ΥΠΟΕΡΓΟ: ΥΠΟΕΡΓΟ 2 «ΠΡΟΓΡΑΜΜΑΤΑ ΚΑΤΑΡΤΙΣΗΣ, ΑΝΑΠΤΥΞΗΣ ΔΕΞΙΟΤΗΤΩΝ, ΕΝΔΥΝΑΜΩΣΗΣ, ΠΙΣΤΟΠΟΙΗΣΗΣ - ΥΛΟΠΟΙΗΣΗ ΜΕ ΙΔΙΑ ΜΕΣΑ, ΕΠΙΜΟΡΦΩΣΗ ΑΠΟ ΤΟ ΕΚΔΔΑ» του Έργου «SUB4. Αναβάθμιση των δεξιοτήτων του ανθρώπινου δυναμικού του Δημόσιου Τομέα» με κωδικό ΟΠΣ ΤΑ 5150174
της Δράσης 16972 ΤΑΑ

ΤΙΤΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ:

PYTHON-ΜΕΡΟΣ Α

ΕΚΠΑΙΔΕΥΤΙΚΟ ΥΛΙΚΟ

Κωδικός εκπαιδευτικού υλικού:

Κωδικός Πιστοποίησης προγράμματος: 1005

Ελλάδα 2.0
Με τη χρηματοδότηση της Ευρωπαϊκής Ένωσης
NextGenerationEU

ΥΠΟΕΡΓΟ: ΥΠΟΕΡΓΟ 2 «ΠΡΟΓΡΑΜΜΑΤΑ ΚΑΤΑΡΤΙΣΗΣ, ΑΝΑΠΤΥΞΗΣ ΔΕΞΙΟΤΗΤΩΝ, ΕΝΔΥΝΑΜΩΣΗΣ, ΠΙΣΤΟΠΟΙΗΣΗΣ - ΥΛΟΠΟΙΗΣΗ ΜΕ ΙΔΙΑ ΜΕΣΑ, ΕΠΙΜΟΡΦΩΣΗ ΑΠΟ ΤΟ ΕΚΔΔΑ» του Έργου «SUB4. Αναβάθμιση των δεξιοτήτων του ανθρώπινου δυναμικού του Δημόσιου Τομέα»

ΤΙΤΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ:

PYTHON-ΜΕΡΟΣ Α

ΟΜΑΔΑ ΕΡΓΑΣΙΑΣ

Μέλη Ομάδας

Συντονιστής:
Ματζαβάκης Ιωάννης

Συγγραφείς:
Γιοχάλας Αλέξανδρος
Γκόγκος Χρήστος
Τράκα Μαρία

Αξιολογητές:
Ζούλιας Εμμανουήλ
Παπαμιχαήλ Γεώργιος

Ελλάδα 2.0
Με τη χρηματοδότηση της Ευρωπαϊκής Ένωσης
NextGenerationEU

ΦΕΒ. 2026

PYTHON-A

ΜΙΑ ΕΙΣΑΓΩΓΗ ΣΤΗΝ PYTHON

Γιοχάλας Αλέξανδρος
Γκόγκος Χρήστος
Τράκα Μαρία

ΕΚΔΔΑ (Εθνικό Κέντρο
Δημόσιας Διοίκησης και
Αυτοδιοίκησης)

https://www.ekdd.gr/

Περιεχόμενα

Εισαγωγή xiii

Περιεχόμενα ανά κεφάλαιο xiv

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ ΣΤΗΝ PYTHON 1

1.1. Προγραμματισμός και γλώσσες προγραμματισμού 1

1.1.1. Μεταγλώττιση και διερμηνεία προγραμμάτων 1

1.2. Η γλώσσα προγραμματισμού Python 2

1.2.1. Οι εκδόσεις της Python 2

1.2.2. Υλοποιήσεις της Python 3

1.2.3. Εγκατάσταση της Python 4

1.2.4. Ιδεατά περιβάλλοντα με venv 7

1.2.5. Ιδεατά περιβάλλοντα με uv 8

1.2.6. Συγγραφή και εκτέλεση προγράμματος με το IDLE 9

1.2.7. Συγγραφή και εκτέλεση προγράμματος με το Visual Studio Code 10

1.2.8. Google Colab 12

1.2.9. Πηγές πληροφόρησης για την Python 13

1.3. Το REPL 14

1.3.1. Παραδείγματα στο REPL με modules της Python Standard Library 15

1.3.2. Άσκηση 1, εντολές στο REPL 17

1.4. Μεταβλητές 17

1.4.1. Τύποι δεδομένων της Python 18

1.4.2. Αριθμητικοί τελεστές 20

1.4.3. Μετατροπές τύπων 21

1.5. Είσοδος και έξοδος τιμών 21

1.6. Σχόλια 23

1.7. Συμβολοσειρές 24

1.7.1. Δεικτοδότηση συμβολοσειρών 24

1.7.2. Τεμαχισμός συμβολοσειρών ... 25
1.7.3. Η συνάρτηση `len()` και διάφορες μέθοδοι συμβολοσειρών ... 26
1.8. Συγκριτικοί τελεστές ... 27
1.9. Λογικοί τελεστές ... 27
1.9.1. Προτεραιότητα ανάμεσα σε αριθμητικούς, συγκριτικούς και λογικούς τελεστές ... 28
1.10. Τελεστές συμβολοσειρών ... 28
1.11. Τελεστές που εφαρμόζονται σε δυαδικά ψηφία ... 29
1.12. Η εντολή επιλογής `if` ... 29
1.12.1. Διπλές ανισότητες σε συνθήκες ... 32
1.12.2. Ο τριαδικός τελεστής στην Python ... 32
1.13. Η εντολή επιλογής `match` ... 33
1.13.1. Άσκηση 2, με την εντολή `if` ... 34
1.14. Η εντολή επανάληψης `while` ... 34
1.15. Η εντολή επανάληψης `for` ... 35
1.16. Οι εντολές `break` και `continue` ... 36
1.17. Η συνάρτηση `range()` ... 37
1.18. Εμφωλευμένες επαναλήψεις ... 37
1.18.1. Άσκηση 3, συνδυασμός εντολών επιλογής και επανάληψης ... 38
1.19. Ερωτήσεις κλειστού τύπου ... 38
1.20. Ασκήσεις προς επίλυση ... 39
ΚΕΦΑΛΑΙΟ 2: ΤΜΗΜΑΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ ΤΗΝ PYTHON ... 41
2.1. Πλεονεκτήματα του τμηματικού προγραμματισμού ... 42
2.2. Κενές συναρτήσεις και συναρτήσεις που επιστρέφουν τιμή ... 43
2.3. Πώς ορίζεται και καλείται μια κενή συνάρτηση ... 43
2.3.1. Ονόματα συναρτήσεων ... 43
2.4. Ορισμός και κλήση μιας συνάρτησης ... 44
2.4.1. Η κλήση μιας συνάρτησης ... 45

2.5. Τα αριστερά περιθώρια στην Python 48

2.6. Συμβολοσειρές τεκμηρίωσης (docstrings) 49

2.7. Σχεδίαση προγραμμάτων ώστε να χρησιμοποιούν συναρτήσεις 50

2.7.1. Παύση της εκτέλεσης μέχρι να πατηθεί το πλήκτρο Enter 50

2.7.2. Η λέξη-κλειδί pass και η χρήση της 51

2.8. Τοπικές μεταβλητές 51

2.8.1. Εμβέλεια και τοπικές μεταβλητές 52

2.8.2. Καθολικές μεταβλητές 54

2.8.3. Συναρτήσεις μέσα σε συναρτήσεις και μη τοπικές μεταβλητές 55

2.9. assert: έλεγχος για μη επιτρεπτές τιμές κατά την εκτέλεση κώδικα 56

2.10. Περνώντας ορίσματα σε συναρτήσεις 57

2.10.1. Εμβέλεια μεταβλητών παραμέτρων 59

2.10.2. Μια σύντομη παρένθεση: έλεγχος της εισόδου 61

2.10.3. Άσκηση 1 62

2.10.4. Περνώντας πολλά ορίσματα 63

2.10.5. Όταν οι συναρτήσεις επιστρέφουν τιμές 64

2.10.6. Επιφέροντας αλλαγές στις παραμέτρους 65

2.10.7. Ορίσματα με λέξη-κλειδί 66

2.10.8. Προαιρετικές παράμετροι συναρτήσεων 68

2.10.9. Συναρτήσεις με μεταβλητό πλήθος ορισμάτων 69

2.10.10. Άσκηση 2 – μεταβλητά ορίσματα θέσης 71

2.10.11. Παράμετροι μόνο με λέξη-κλειδί 71

2.10.12. Κανόνες για το συνδυασμό των διαφόρων τύπων ορισμάτων 72

2.11. Αναδρομή 75

2.11.1. Άσκηση 3 - αναδρομή 76

2.12. Προγραμματισμός: δομημένος, τμηματικός, διαδικασιακός ή αντικειμενοστραφής; 76

2.13. Προχωρημένες δυνατότητες της Python για συναρτήσεις 78

2.14. Ολοκληρωμένα περιβάλλοντα ανάπτυξης (Integrated Development Environments, IDEs) 78

2.15. Ερωτήσεις κλειστού τύπου 82

2.16. Ασκήσεις προς επίλυση 85

ΚΕΦΑΛΑΙΟ 3: ΛΙΣΤΕΣ, ΒΙΒΛΙΟΘΗΚΕΣ ΚΑΙ ΤΜΗΜΑΤΑ 89

3.1. Λίστες 90

3.1.1. Δημιουργία Λίστας 91

3.1.2. Πρόσβαση σε στοιχεία της Λίστας - Δεικτοδότηση Λιστών 97

3.1.3. Τροποποίηση στοιχείων λίστας 101

3.1.4. Λειτουργίες και Μέθοδοι Λιστών 103

3.1.5. Διάσχιση στοιχείων Λίστας (Iteration) 114

3.1.6. Πολυδιάστατοι Πίνακες (Lists of Lists) 117

3.1.7. Αντιγραφή Λιστών 119

3.2. Εγκατάσταση Βιβλιοθηκών με τη χρήση του PIP από το PyPI 121

3.2.1. Τι είναι το PyPI (Python Package Index) 121

3.2.2. Τι είναι το PIP (Pip Installs Packages); 122

3.2.3. Πρακτική Εφαρμογή: Βασικές Εντολές 122

3.2.4. Προβολή Εγκατεστημένων Βιβλιοθηκών 123

3.2.5. Απεγκατάσταση Βιβλιοθήκης 123

3.2.6. Διαδικασία Επαλήθευσης 123

3.2.7. Άσκηση Διαχείρισης Βιβλιοθηκών 123

3.3. Modules και Packages 124

3.3.1. Τι είναι το Module; 124

3.3.2. Οργάνωση κώδικα σε Modules 124

3.3.3. Τρόποι Εισαγωγής (Importing) 125

3.3.4. Άσκηση 20 - Δημιουργία και χρήση Module 125

3.3.5. Τι είναι το Package; 126

3.4. Η Μεταβλητή __name__ και ο Ρόλος της στην Εκτέλεση Κώδικα 130
3.4.1. Γιατί είναι απαραίτητος ο έλεγχος if __name__ == "__main__" 130
3.4.2. Άσκηση 22 με τη χρήση της __name__ 131
3.5. Ερωτήσεις Κλειστού Τύπου 132
3.6. Ασκήσεις προς επίλυση 133
3.6.1. Άσκηση 23. Εύρεση Μικρότερου Αριθμού 133
3.6.2. Άσκηση 24: Πίνακας Βαθμολογιών 133
ΚΕΦΑΛΑΙΟ 4: ΣΥΝΘΕΤΟΙ ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ ΠΕΡΑ ΑΠΟ ΤΙΣ ΛΙΣΤΕΣ 134
4.1. Η Δομή Δεδομένων των Πλειάδων (Tuples) 135
4.1.1. Ορισμός και Χαρακτηριστικά 135
4.1.2. Σύγκριση με Λίστες 136
4.1.3. Δημιουργία Πλειάδας 136
4.1.4. Πρόσβαση σε στοιχεία Πλειάδας – Δεικτοδότηση Πλειάδων 137
4.1.5. Τροποποίηση στοιχείων πλειάδας 139
4.1.6. Αποσυσκευασία (Unpacking) 140
4.1.7. Λειτουργίες και Μέθοδοι Πλειάδων 141
4.1.8. Διάσχιση στοιχείων Πλειάδας (Loop Tuples) 143
4.1.9. Εμβάθυνση στις Πλειάδες (Tuples) 145
4.2. Η Δομή Δεδομένων των Συνόλων (Sets) 145
4.2.1. Ορισμός και Χαρακτηριστικά 146
4.2.2. Σύγκριση με Λίστες και Πλειάδες 147
4.2.3. Δημιουργία Συνόλου 147
4.2.4. Πρόσβαση στα στοιχεία Συνόλου 148
4.2.5. Τροποποίηση στοιχείων Συνόλου 149
4.2.6. Αποσυσκευασία (Unpacking) 149
4.2.7. Λειτουργίες και Μέθοδοι Συνόλων 149
4.2.8. Διάσχιση στοιχείων Συνόλου (Iteration - Loop Sets) 156

4.2.9. Ασκήσεις Συνόλων 156
4.3. Η Δομή Δεδομένων των Λεξικών (Dictionaries) 157
4.3.1. Ορισμός και Χαρακτηριστικά 157
4.3.2. Σύγκριση με Άλλες Δομές Δεδομένων 158
4.3.3. Δημιουργία Λεξικού 159
4.3.4. Πρόσβαση σε στοιχεία Λεξικού και Διαχείριση Εξαιρέσεων 161
4.3.5. Μέθοδοι Λεξικού 163
4.3.6. Διάσχιση στοιχείων Λεξικού 171
4.3.7. Ασκήσεις Λεξικών 175
4.4. Μεταλλαξιμότητα – Μεταβλητότητα (Mutability) και Μη Μεταλλαξιμότητα - Αμεταβλητότητα (Immutability) 176
4.4.1. Εφαρμογή Αμεταβλητότητας - Μη Μεταλλαξιμότητας (Immutability) 176
4.5. Περιφραστικές λίστες και λεξικά (comprehensions) 177
4.5.1. Περιφραστικές Λίστες (List Comprehensions) 177
4.5.2. Περιφραστικά Λεξικά (Dictionary Comprehensions) 179
4.5.3. Ασκήσεις Περιφραστικές λίστες και λεξικά 180
4.6. Επίλυση Προβλημάτων με Λεξικά, Λίστες, Σύνολα και Πλειάδες 180
4.6.1. Πρόβλημα 1: Διαχείριση λίστας εργασιών 180
4.6.2. Πρόβλημα 2: Αποθήκευση γεωγραφικών συντεταγμένων 181
4.6.3. Πρόβλημα 3: Εύρεση κοινών και μοναδικών στοιχείων - Εγγεγραμμένοι Μαθητές 182
4.6.4. Πρόβλημα 4: Τηλεφωνικός κατάλογος επαφών 183
4.6.5. Πρόβλημα 5: Κατάλογος προϊόντων με σταθερές κατηγορίες 184
4.6.6. Πρόβλημα 6: Σύστημα διαχείρισης μαθητών 184
4.7. Μορφοποίηση λεκτικών με διάφορους τρόπους 186
4.7.1. f-strings (Formatted String Literals) 186
4.7.2. Μέθοδος format() 187
4.7.3. Τελεστής % 189

4.7.4. Ασκήσεις Μορφοποίηση λεκτικών με διάφορους τρόπους.......190
4.8. Ερωτήσεις Κλειστού Τύπου.......191
4.9. Ασκήσεις προς επίλυση.......192
ΚΕΦΑΛΑΙΟ 5: ΔΙΑΧΕΙΡΙΣΗ ΑΡΧΕΙΩΝ ΔΙΑΦΟΡΩΝ ΤΥΠΩΝ.......194
5.1. Εργασία με αρχεία.......194
5.1.1. Άνοιγμα αρχείων κειμένου για ανάγνωση.......195
5.1.2. Άνοιγμα αρχείων κειμένου για εγγραφή.......198
5.1.3. Διαγραφή αρχείου.......200
5.2. Άνοιγμα αρχείου με την εντολή `with` – context managers.......201
5.2.1. Άνοιγμα αρχείου για ανάγνωση με την εντολή `with`.......201
5.2.2. Άνοιγμα αρχείου για εγγραφή με την εντολή `with`.......201
5.2.3. Άνοιγμα αρχείου για προσθήκη δεδομένων με την εντολή `with`.......202
5.3. Άδειασμα περιεχομένων αρχείων.......202
5.4. Εργασία με αρχεία τύπου CSV.......203
5.4.1. Δημιουργία αρχείου CSV.......203
5.4.2. Ανάγνωση αρχείου CSV.......205
5.4.3. Άσκηση 1 – εργασία με αρχεία CSV.......209
5.5. Εργασία με αρχεία JSON.......210
5.5.1. Μετατροπή από JSON σε Python.......211
5.5.2. Μετατροπή από Python σε JSON.......212
5.6. Εργασία με αρχεία XML.......215
5.6.1. Ανάγνωση ενός αρχείου XML.......218
5.6.2. Προσθήκη σε αρχείο XML.......219
5.6.3. Μετατροπή αρχείου XML σε αρχείο JSON.......220
5.6.4. Μετατροπή XML σε HTML και έλεγχος εγκυρότητας της XML.......222
5.7. Εργασία με αρχεία excel.......226
5.7.1. Άσκηση 2 – εργασία με αρχεία excel.......229

5.8. Σειριοποίηση και αποσειριοποίηση δεδομένων με το pickle 229
5.8.1. Δημιουργία αρχείου με το pickle 230
5.8.2. Άνοιγμα αρχείου pickle για ανάγνωση και προσθήκη 230
5.8.3. Shelve: λεξικά στο ράφι 232
5.9. Εργασία με καταλόγους αρχείων: οι βιβλιοθήκες os και shutil 234
5.9.1. Διαβάζοντας τα περιεχόμενα ενός φακέλου 235
5.9.2. Δημιουργία και διαγραφή φακέλων και αρχείων 236
5.9.3. Αντιγραφή και μετακίνηση αρχείων και φακέλων 239
5.9.4. Διασχίζοντας ένα μονοπάτι του συστήματος αρχείων με την os.walk 240
5.9.5. Άσκηση 3 – επεξεργασία φακέλων και αρχείων 242
5.10. Ερωτήσεις κλειστού τύπου 242
5.11. Ασκήσεις προς επίλυση 243
ΚΕΦΑΛΑΙΟ 6: ΧΕΙΡΙΣΜΟΣ ΛΑΘΩΝ ΚΑΙ ΕΙΔΙΚΕΣ ΒΙΒΛΙΟΘΗΚΕΣ 245
6.1. Η ανάγκη για χειρισμό λαθών 246
6.2. Εξαιρέσεις 246
6.2.1. Χειρισμός εξαιρέσεων με την try-except-else-finally 247
6.2.2. Η εντολή `raise` 248
6.3. Αμυντικός προγραμματισμός 250
6.3.1. Η εντολή `assert` 250
6.3.2. Παράδειγμα αμυντικού προγραμματισμού με εξαιρέσεις και επανάληψη 251
6.3.3. Άσκηση 1, με πρόκληση και σύλληψη εξαίρεσης 252
6.4. Τυχαιότητα 252
6.4.1. Αναπαραγωγιμότητα αποτελεσμάτων με seed(s) 253
6.4.2. Άσκηση 2, με τυχαίες τιμές 254
6.5. Χειρισμός ημερομηνιών, ωρών και χρονικών σημάνσεων 254
6.5.1. Το module `datetime` 254
6.5.2. Η εξωτερική βιβλιοθήκη `dateutil` 256

6.5.3. Το module calendar 256
6.5.4. Το module time 257
6.5.5. Άσκηση 3, με χειρισμό ημερομηνιών 258
6.6. Jupyter Notebooks 258
6.6.1. Εγκατάσταση και εκκίνηση Jupyter Notebook 259
6.6.2. Βασικά παραδείγματα χρήσης Jupyter Notebook 259
6.7. Ερωτήσεις κλειστού τύπου 261
6.8. Ασκήσεις προς επίλυση 262
ΚΕΦΑΛΑΙΟ 7: ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ 264
7.1. Κλάσεις και αντικείμενα 264
7.1.1. Σταδιακή δημιουργία μιας κλάσης και αντικειμένων της στο REPL 265
7.1.2. Δημιουργία κλάσης και αντικειμένων σε πρόγραμμα 266
7.1.3. Ένα ακόμα παράδειγμα δημιουργίας κλάσης και στιγμιοτύπων της 267
7.2. Dunder μέθοδοι αντικειμένων 269
7.2.1. Παράδειγμα με __str__ και __repr__ 270
7.2.2. Παράδειγμα με την μέθοδο __del__ 271
7.2.3. Παράδειγμα με την μέθοδο __add__ 272
7.2.4. Παράδειγμα με την μέθοδο __call__ 273
7.2.5. Παράδειγμα με την μέθοδο __getitem__ 274
7.2.6. Άσκηση 1 275
7.3. Επίλυση προβλήματος με διαδικασιακό προγραμματισμό και με αντικειμενοστραφή προγραμματισμό 275
7.4. Σύγκριση αντικειμένων 277
7.4.1. Εισαγωγή αντικειμένων σε σύνολα και χρήση αντικειμένων ως κλειδιά λεξικών 278
7.5. Αντιγραφή αντικειμένων (ρηχή και βαθιά αντιγραφή) 279
7.6. Κληρονομικότητα 281
7.6.2. Πολλαπλή κληρονομικότητα 283

7.7. Μεταβλητές κλάσης και μέθοδοι κλάσης 284
7.7.1. Χρήση μεθόδου κλάσης ως εναλλακτικού κατασκευαστή 285
7.8. Στατικές μεταβλητές και στατικές μέθοδοι 285
7.9. Σύνθεση 286
7.10. Ενθυλάκωση 287
7.11. Ιδιότητες (properties) 289
7.12. Ερωτήσεις κλειστού τύπου 290
7.13. Ασκήσεις προς επίλυση 291
ΠΑΡΑΡΤΗΜΑ Α’ - ΛΥΣΕΙΣ ΑΣΚΗΣΕΩΝ 294
Λύσεις ασκήσεων κεφαλαίου 1 294
Άσκηση 1 294
Άσκηση 2 295
Άσκηση 3 296
Λύσεις ασκήσεων κεφαλαίου 2 297
Άσκηση 1 297
Άσκηση 2 298
Άσκηση 3 298
Λύσεις ασκήσεων κεφαλαίου 3 299
Άσκηση 1 299
Άσκηση 2 299
Άσκηση 3 299
Άσκηση 4 299
Άσκηση 5 299
Άσκηση 6 300
Άσκηση 7 300
Άσκηση 8 300
Άσκηση 9 301

Άσκηση 10 ... 301

Άσκηση 11 ... 301

Άσκηση 12 ... 301

Άσκηση 13 ... 302

Άσκηση 14 ... 302

Άσκηση 15 ... 302

Άσκηση 16 ... 302

Άσκηση 17 ... 304

Άσκηση 18 ... 304

Άσκηση 19 ... 304

Άσκηση 20 ... 305

Άσκηση 21 ... 306

Άσκηση 22 ... 306

Λύσεις ασκήσεων κεφαλαίου 4 ... 307

Άσκηση 1 ... 307

Άσκηση 2 ... 307

Άσκηση 3 ... 308

Άσκηση 4 ... 308

Άσκηση 5 ... 309

Άσκηση 6 ... 309

Άσκηση 7 ... 310

Άσκηση 8 ... 310

Άσκηση 9 ... 310

Άσκηση 10 ... 311

Άσκηση 11 ... 311

Λύσεις ασκήσεων κεφαλαίου 5 ... 312

Άσκηση 1 ... 312

Άσκηση 2 313
Άσκηση 3 314
Λύσεις ασκήσεων κεφαλαίου 6 315
Άσκηση 1 315
Άσκηση 2 316
Άσκηση 3 317
Λύσεις ασκήσεων κεφαλαίου 7 318
Άσκηση 1 318
Άσκηση 2 319
Άσκηση 3 320
ΠΑΡΑΡΤΗΜΑ Β’ - ΑΠΑΝΤΗΣΕΙΣ ΕΡΩΤΗΣΕΩΝ ΚΛΕΙΣΤΟΥ ΤΥΠΟΥ 322
ΒΙΒΛΙΟΓΡΑΦΙΑ 323

Εισαγωγή

Η Python είναι μια ισχυρή γλώσσα προγραμματισμού που είναι εύκολη στην εκμάθηση. Για διάφορους λόγους όπως η ευκολία προγραμματισμού, η αναγνωσιμότητα του κώδικα, η ανάπτυξη ισχυρών βιβλιοθηκών ανάλυσης δεδομένων, η αύξηση των επιδόσεων των Η/Υ και άλλους, έχει αναδειχθεί ως η πλέον δημοφιλής γλώσσα προγραμματισμού τα τελευταία χρόνια. Στο πρόγραμμα επιμόρφωσης Python-A γίνεται μια εισαγωγή στα βασικά στοιχεία της γλώσσας όπως οι μεταβλητές, οι τύποι δεδομένων, οι τελεστές, οι εντολές εισόδου/εξόδου, η συγγραφή και εκτέλεση απλών προγραμμάτων, οι εντολές επιλογής, οι εντολές επανάληψης, ο τμηματικός προγραμματισμός, οι βασικές δομές δεδομένων της γλώσσας (λίστες, πλειάδες, σύνολα, λεξικά), ο διαμερισμός προγραμμάτων σε τμήματα, οι βιβλιοθήκες, η διαχείριση αρχείων διαφόρων τύπων, ο χειρισμός λαθών, οι εξαιρέσεις και ο αντικειμενοστραφής προγραμματισμός. Επιπλέον, γίνεται αναφορά στην εγκατάσταση της Python, στη δημιουργία εικονικών περιβαλλόντων, στα σημειωματάρια Jupyter Notebook, σε ενσωματωμένες βιβλιοθήκες της γλώσσας όπως οι `math`, `random`, `os`, `pickle`, `datetime`, `time`, `calendar` και σε εξωτερικές βιβλιοθήκες της γλώσσας όπως η `dateutil` και άλλες.

Το πρόγραμμα επιμόρφωσης Python-A έχει σχεδιαστεί έτσι ώστε να είναι ασύγχρονο, που σημαίνει ότι οι εκπαιδευόμενοι καλούνται να μελετούν πρώτα το υλικό κάθε ενότητας, στη διάρκεια μιας ημερολογιακής εβδομάδας, και κατά τη διάρκεια μελέτης να διατυπώνουν απορίες, ερωτήσεις και σχόλια χρησιμοποιώντας τα ηλεκτρονικά κανάλια επικοινωνίας που διατίθενται μέσω της πλατφόρμας Learning Management System (LMS) του ΕΚΔΔΑ[1]. Σε κάθε ενότητα υπάρχει εκπαιδευτικό υλικό, παραδείγματα κώδικα, λυμένες ασκήσεις, ασκήσεις προς επίλυση και ερωτήσεις ανακεφαλαίωσης.

Η διάρκεια του προγράμματος είναι επτά εβδομάδες και κάθε εβδομάδα μελέτης αντιστοιχεί σε ένα κεφάλαιο του παρόντος συγγράμματος. Οι εκπαιδευόμενοι καλούνται να υποβάλουν λύσεις σε 3 ασκήσεις (μικρά projects). Επίσης, συμμετέχουν στην τελική αξιολόγηση που είναι με τη μορφή ερωτήσεων κλειστού τύπου σε όλη την ύλη της επιμόρφωσης.

[1] https://lms.ekdd.gr

Περιεχόμενα ανά κεφάλαιο

Στο κεφάλαιο 1 περιγράφεται η έννοια της μεταγλώττισης και της διερμηνείας ενός προγράμματος, οι εκδόσεις και οι υλοποιήσεις της Python, η εγκατάσταση της Python, η απευθείας εκτέλεση εντολών και λήψη αποτελεσμάτων στο REPL, η συγγραφή και εκτέλεση απλών προγραμμάτων στο IDLE, οι αριθμητικοί τελεστές, οι μεταβλητές, οι τύποι δεδομένων, τα σχόλια, η είσοδος και η έξοδος με τις συναρτήσεις `input()` και `print()`, οι συγκριτικοί τελεστές, η εντολή επιλογής `if` και οι παραλλαγές της (`elif`, `else`), η νέα εντολή `match`, οι εντολές επανάληψης `while` και `for`, οι εντολές `break` και `continue`, οι λογικοί τελεστές και οι τελεστές που εφαρμόζονται σε δυαδικά ψηφία.

Στο κεφάλαιο 2 εισάγεται η έννοια του *τμηματικού προγραμματισμού* και των *συναρτήσεων* της Python. Αναλύεται ο τρόπος δήλωσης και κλήσης των συναρτήσεων στην Python, τι είναι παράμετροι (parameters) και τι ορίσματα (arguments) και οι διάφοροι τύποι τους που υποστηρίζονται από την Python. Επίσης περιγράφονται οι τοπικές, καθολικές και μη τοπικές μεταβλητές, οι συναρτήσεις μέσα σε συναρτήσεις και η έννοια της. Επίσης, παρουσιάζεται η έννοια της αναδρομής. Γίνεται αναφορά στις έννοιες του δομημένου προγραμματισμού, του τμηματικού προγραμματισμού, του διαδικασιακού προγραμματισμού και του αντικειμενοστραφούς προγραμματισμού και των σχέσεων μεταξύ τους. Τέλος, γίνεται μια σύντομη παρουσίαση για τα Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης IDEs) και ειδικά του VS Code και κάποιων από τις δυνατότητές του.

Στο κεφάλαιο 3 εξετάζονται οι λίστες, ο πιο ευέλικτος και ευρέως χρησιμοποιούμενος σύνθετος τύπος δεδομένων, με ανάλυση σε βάθος της δημιουργίας, της δεικτοδότησης, της διάσχισης, των τμημάτων (slices), της αντιγραφής και των πολυδιάστατων πινάκων. Παρουσιάζεται επίσης το οικοσύστημα βιβλιοθηκών της Python και ο τρόπος εγκατάστασης εξωτερικών πακέτων μέσω του pip και του αποθετηρίου PyPI, που παρέχουν πρόσβαση σε χιλιάδες έτοιμα εργαλεία για κάθε είδους εφαρμογή. Τέλος, αναλύεται η οργάνωση του κώδικα σε modules και packages, με ιδιαίτερη αναφορά στον ρόλο της μεταβλητής __name__ και στη σημασία της για τη δόμηση επαναχρησιμοποιήσιμου και συντηρήσιμου κώδικα.

Στο κεφάλαιο 4 αναλύονται οι υπόλοιπες σύνθετες δομές δεδομένων της Python: οι πλειάδες (tuples), που εγγυώνται την ακεραιότητα σταθερών δεδομένων, τα σύνολα (sets), που διαχειρίζονται μοναδικά στοιχεία και υποστηρίζουν πράξεις συνόλων, και τα λεξικά (dictionaries), που προσφέρουν γρήγορη αναζήτηση μέσω ζευγών κλειδιού-τιμής. Εξετάζεται επίσης η έννοια της μεταβλητότητας και αμεταβλητότητας (mutability/immutability), καθώς και οι περιφραστικές λίστες

και λεξικά (comprehensions) και οι τρόποι μορφοποίησης λεκτικών (f-strings, format(), %). Το κεφάλαιο ολοκληρώνεται με χαρακτηριστικά προβλήματα που αναδεικνύουν την αποτελεσματική αξιοποίηση των δομών αυτών στην πράξη.

Στο κεφάλαιο 5 εισάγεται η εργασία με αρχεία στην Python. Παρουσιάζεται το άνοιγμα αρχείων κειμένου για ανάγνωση, εγγραφή ή επεξεργασία, το σώσιμο των αλλαγών και το κλείσιμο των αρχείων. Στη συνέχεια περιγράφεται η χρήση, επεξεργασία και κάποιες μετατροπές μεταξύ διαφόρων τύπων αρχείων κειμένου: CSV, JSON, XML, excel. Κατόπιν περιγράφονται οι `pickle` και `shelve` για αποθήκευση και ανάγνωση αντικειμένων στην Python. Τέλος, γίνεται μια εισαγωγή στο πώς αλληλοεπιδρά η Python με το λειτουργικό σύστημα και το σύστημα αρχείων μέσω στην modules `os` και `shutil` για τη δημιουργία, μετονομασία, διαγραφή και αντιγραφή φακέλων και αρχείων και τη διάσχιση υποδέντρου του συστήματος αρχείων για εργασία με τα περιεχόμενά του.

Στο κεφάλαιο 6 περιγράφεται η ανάγκη για χειρισμό λαθών που ανακύπτει στα προγράμματα, ο αμυντικός προγραμματισμός, οι εξαιρέσεις, οι τύποι εξαιρέσεων, η σύλληψη και η πρόκληση εξαιρέσεων. Επιπλέον, δίνονται παραδείγματα χρήσης της βιβλιοθήκης `random`, του χειρισμού δεδομένων ημερομηνιών και χρόνου με τις βιβλιοθήκες `datetime`, `dateutil`, `time` και `calendar` καθώς και εκτέλεσης κώδικα με ταυτόχρονη τεκμηρίωση του σε Jupyter Notebooks.

Στο κεφάλαιο 7 γίνεται μια εισαγωγή στον αντικειμενοστραφή προγραμματισμό με την Python. Ειδικότερα, περιγράφονται οι έννοιες των κλάσεων, των ιδιοτήτων και των μεθόδων των αντικειμένων, του `self`, των μεθόδων `__init__`, `__str__`, `__add__`, `__call__`, `__getitem__`, της σύγκρισης αντικειμένων, της αντιγραφής αντικειμένων (ρηχή και βαθιά αντιγραφή) της χρήσης αντικειμένων ως κλειδιά λεξικών, της κληρονομικότητας, της πολλαπλής κληρονομικότητας, της σύνθεσης, της ενθυλάκωσης, των getters, setters και της αντικατάστασής τους από ιδιότητες (properties), των στατικών μεθόδων, των μεθόδων κλάσεων και των μεταβλητών κλάσεων ή αλλιώς στατικών μεταβλητών.

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ ΣΤΗΝ PYTHON

Σε αυτό το κεφάλαιο θα επιχειρηθεί μια εισαγωγή στα βασικά στοιχεία της γλώσσας Python. Θα περιγραφεί η εγκατάσταση της Python, και πως μπορούν να γραφούν και να εκτελεστούν σύντομα προγράμματα Python στο περιβάλλον IDLE που εγκαθίσταται με την εγκατάσταση της Python ή σε κάποιο περιβάλλον IDE (Integrated Development Environment) όπως το Visual Studio Code. Σύντομα αποσπάσματα κώδικα θα εκτελούνται στο REPL (Read Evaluate Print Loop), που και αυτό εγκαθίσταται με την εγκατάσταση της Python και δίνει τη δυνατότητα άμεσης εκτέλεσης κώδικα. Το REPL είναι ένα διαδραστικό περιβάλλον προγραμματισμού που διαβάζει την είσοδο του χρήστη, την αξιολογεί, εκτυπώνει αποτελέσματα και επαναλαμβάνει τον κύκλο, επιτρέποντας την εξερεύνηση των δυνατοτήτων της Python. Επίσης, θα περιγραφεί η χρήση του Google Colab για την εκτέλεση αποσπασμάτων κώδικα Python στο cloud της Google, ακόμα και χωρίς τοπική εγκατάσταση της Python.

1.1. Προγραμματισμός και γλώσσες προγραμματισμού

Προγραμματισμός είναι η διαδικασία δημιουργίας προγραμμάτων. Ένα πρόγραμμα είναι η περιγραφή της σειράς βημάτων που επιτελούν έναν σκοπό, όπως τον υπολογισμό ενός αποτελέσματος. Τα προγράμματα γράφονται σε γλώσσες προγραμματισμού όπως είναι η Python η Java, η C, η C++, η Rust κ.α. Ο κώδικας ενός προγράμματος ονομάζεται πηγαίος κώδικας (source code) του προγράμματος και αποτελεί την υλοποίηση ενός αλγορίθμου σύμφωνα με τους συντακτικούς κανόνες που ισχύουν στην γλώσσα προγραμματισμού στην οποία έχει γραφεί.

1.1.1. Μεταγλώττιση και διερμηνεία προγραμμάτων

Ο πηγαίος κώδικας (δηλαδή ο κώδικας .py ή .ipynb για την Python) δεν μπορεί να εκτελεστεί απευθείας από ένα υπολογιστικό σύστημα, αλλά θα πρέπει πρώτα να μεταφραστεί σε γλώσσα μηχανής. Υπάρχουν δύο βασικές κατηγορίες μεταφραστικών προγραμμάτων οι μεταγλωττιστές (compilers) και οι διερμηνευτές (interpreters) και κάθε γλώσσα προγραμματισμού διαθέτει είτε μεταγλωττιστή είτε διερμηνευτή, με την Python να διαθέτει διερμηνευτή. Γενικά ισχύει ότι οι μεταγλωττιστές μεταφράζουν όλο τον πηγαίο κώδικα σε γλώσσα μηχανής, δημιουργώντας ένα εκτελέσιμο, το οποίο εκτελείται στη συνέχεια, τροφοδοτούμενο κάθε φορά με είσοδο που διαφοροποιεί την εκτέλεσή του και τα αποτελέσματα που παράγει. Από την άλλη μεριά οι διερμηνευτές μεταφράζουν και εκτελούν γραμμή προς γραμμή τον πηγαίο κώδικα. Θα πρέπει να σημειωθεί ότι η παραπάνω περιγραφή αποτελεί μια απλοποίηση, με τους σχεδιαστές της κάθε

γλώσσας προγραμματισμού να επιλέγουν το βαθμό που η γλώσσα βρίσκεται πλησιέστερα στο υπόδειγμα της μεταγλώττισης ή της διερμηνείας.

1.2. Η γλώσσα προγραμματισμού Python

Η Python δημιουργήθηκε από τον Guido Van Rossum το 1989 στο Centrum Wiskunde & Informatica στην Ολλανδία. Αποτέλεσε μετεξέλιξη της πειραματικής γλώσσας προγραμματισμού ABC που έδινε έμφαση στην αναγνωσιμότητα και στην απλότητα του κώδικα, στοιχεία που μεταφέρθηκαν και στην Python. O Guido Van Rossum διατήρησε τον ρόλο του BDFL (Benevolent Dictator For Life) της Python μέχρι το 2018, οπότε και αποσύρθηκε διατηρώντας έναν συμβουλευτικό ρόλο με τις απόψεις του για την γλώσσα να έχουν βαρύνουσα σημασία. Πλέον, η εξέλιξη και η γενικότερη διακυβέρνηση της Python καθορίζεται από ένα πενταμελές συμβούλιο καθοδήγησης (steering council) με συγκεκριμένες αρμοδιότητες, που εκλέγεται περιοδικά, όπως περιγράφεται στο PEP 13 (PEP=Python Enhancement Proposal).

1.2.1. Οι εκδόσεις της Python

Η έκδοση της Python που χρησιμοποιείται σήμερα είναι η έκδοση 3. Στο σχετικά πρόσφατο παρελθόν υπήρχε και η έκδοση 2, που έφτασε στο λεγόμενο τέλος ζωής (end of life) στις 1/1/2020 που σημαίνει ότι η έκδοση αυτή της Python δεν εξελίχθηκε μετά από αυτή την ημερομηνία. Ωστόσο, υπάρχουν πολλές εφαρμογές λογισμικού που έχουν γραφεί σε Python 2 ενώ είναι πιθανό να συναντήσει κανείς βιβλιοθήκες που είναι σε Python 2. Ο κώδικας αυτός δεν μπορεί συνήθως να χρησιμοποιηθεί απευθείας στην Python 3 διότι υπάρχουν κάποιες μικρές αλλά σημαντικές διαφορές που δημιουργούν ασυμβατότητες. Μια σύνοψη των σημαντικότερων διαφορών μεταξύ της Python 2 και της Python 3 παρουσιάζεται στον Πίνακας 1.

Χαρακτηριστικό	Python 2	Python 3
Εκτύπωση	`print "Hello" # η print είναι εντολή)`	`print("Hello") # η print είναι συνάρτηση`
Διαίρεση (/)	`5/2 # ακέραια διαίρεση με αποτέλεσμα 2` `5.0/2 # πραγματική διαίρεση με αποτέλεσμα 2.5`	`5/2 # πραγματική διαίρεση με αποτέλεσμα 2.5` `5//2 # ακέραια διαίρεση με αποτέλεσμα 2`
Συμβολοσειρές	`"abc" # λεκτικό με κωδικοποίηση bytes` `u"abc" # λεκτικό με κωδικοποίηση unicode`	`"abc" # λεκτικό με κωδικοποίηση unicode` `b"abc" # λεκτικό με κωδικοποίηση bytes`

Είσοδος	raw_input() # επιστρέφει λεκτικό με την είσοδο του χρήστη input() # πραγματοποιεί αποτίμηση κώδικα	Η raw_input() αφαιρέθηκε στην Python 3 input() # επιστρέφει λεκτικό με την είσοδο του χρήστη
range	range() # επιστρέφει λίστα xrange() # οκνηρή αποτίμηση λίστας	range(x) # οκνηρή αποτίμηση λίστας Η xrange() αφαιρέθηκε στην Python 3
Εξαιρέσεις	except ValueError, e:	except ValueError as e:

Μέθοδοι λεξικών	d.keys() # επιστρέφει λίστα
	d.items() # επιστρέφει λίστα
		d.keys() # dict_keys (view)
		d.items() # dict_items (view)

Unicode	Δεν είναι το default (πρέπει να χρησιμοποιηθεί το u"...")	Είναι το default, όλα τα λεκτικά είναι Unicode
Διάσχιση	Συναρτήσεις όπως οι map(), zip(), filter(), επιστρέφουν λίστες	Οι συναρτήσεις map(), zip(), filter() είναι iterators

Πίνακας 1 – Σημαντικές διαφορές Python 2 και Python 3.

1.2.2. Υλοποιήσεις της Python

Η κύρια υλοποίηση της Python είναι η CPython αλλά υπάρχουν και άλλες υλοποιήσεις όπως η PyPy, η Cython, η IronPython, η Jython, η MicroPython, η CircuitPython η RustPython, η Nuitka κ.α.

Η CPython είναι η υλοποίηση αναφοράς της Python και μπορεί να μεταφορτωθεί από το https://www.python.org/. Έχει γραφεί στη γλώσσα προγραμματισμού C, μεταγλωττίζει αρχεία πηγαίου κώδικα Python (.py) σε .pyc και στη συνέχεια τα εκτελεί διερμηνευόμενα, εντολή προς εντολή στην εικονική μηχανή CPython. Παρουσιάζει υψηλή σταθερότητα και διαθέτει μεγάλο αριθμό εξωτερικών βιβλιοθηκών. Ωστόσο, έχει σχετικά χαμηλή ταχύτητα εκτέλεσης προγραμμάτων, κυρίως λόγω διερμηνείας αλλά και λόγω του GIL (Global Interpreter Lock) που είναι ένας μηχανισμός του διερμηνευτή που επιτρέπει μόνο σε ένα νήμα (thread) τη φορά να εκτελεί κώδικα, απλοποιώντας τη διαχείριση μνήμης αλλά περιορίζοντας τον πραγματικό παραλληλισμό που μπορεί να επιτευχθεί σε πολυπύρηνα συστήματα.

Η PyPy είναι η δεύτερη πιο σημαντική υλοποίηση της Python και μπορεί να μεταφορτωθεί από το https://pypy.org/. Για ορισμένα σενάρια εφαρμογών παράγει σημαντικά ταχύτερο κώδικα από ότι η CPython. Ωστόσο, παρουσιάζει θέματα συμβατότητας, ακόμα και σε καθιερωμένες βιβλιοθήκες της Python, ειδικά όταν πρόκειται για τις πλέον πρόσφατες εκδόσεις τους, όπως στις βιβλιοθήκες numpy, pandas, scikit-learn κ.α.

Η Cython είναι ένα υπερσύνολο της γλώσσας Python με προαιρετικές δηλώσεις τύπων δεδομένων (στην Python ο προγραμματιστής δεν υποχρεούται να δηλώσει τον τύπο δεδομένων των

μεταβλητών). Μπορεί να χρησιμοποιηθεί για επιτάχυνση αριθμητικών υπολογισμών. Απαιτεί ένα επιπλέον βήμα μεταγλώττισης και είναι λιγότερο δυναμική σε σχέση με την κλασική Python.

Στο παρόν σύγγραμμα θα χρησιμοποιηθεί η υλοποίηση CPython στην έκδοση 3 της γλώσσας Python. Ο τρόπος ονομασίας των εκδόσεων ακολουθεί την λεγόμενη canonical version μορφή που είναι MAJOR.MINOR.MICRO, όπου για παράδειγμα στην έκδοση 3.14.2 είναι: MAJOR=3, MINOR=14, MICRO=2. Αλλαγή στην MAJOR έκδοση σημαίνει ότι ενδεχόμενα παραβιάζεται η συμβατότητα με προηγούμενες εκδόσεις. Δεν υπάρχει σχεδιασμός για αλλαγή της MAJOR έκδοσης της Python, αλλά όλες οι μεγάλης κλίμακας αλλαγές αναμένεται να συμβούν εντός της MAJOR έκδοσης 3. Αλλαγή στη MINOR έκδοση εισάγει νέα χαρακτηριστικά στη γλώσσα και βελτιώσεις απόδοσης. Συνήθως συμβαίνει περίπου μια ανά έτος, διατηρεί την προς τα πίσω συμβατότητα και μπορεί να εισάγει νέες βιβλιοθήκες στις τυπικές βιβλιοθήκες της γλώσσας. Για παράδειγμα, στην έκδοση 3.6 προστέθηκαν τα f-strings ως ένας απλούστερος τρόπος μορφοποίησης κειμένου που περιέχει εκφράσεις, στην έκδοση 3.10 προστέθηκε η εντολή match ως απλούστερος και «καθαρότερος» τρόπος συγγραφής κώδικα που στην προηγούμενη έκδοση (3.9) θα έπρεπε να γραφεί με την εντολή if, και στην έκδοση 3.11 ενσωματώθηκαν αλλαγές που επιταχύνουν την εκτέλεση του κώδικα. Τέλος, αλλαγές στην MICRO έκδοση διορθώνουν λάθη (bug fixes) και δεν προσθέτουν νέα χαρακτηριστικά στη γλώσσα.

1.2.3. Εγκατάσταση της Python

Η ύπαρξη ενός ορθά εγκατεστημένου περιβάλλοντος Python, που θα μπορεί εύκολα να αναδημιουργηθεί, βοηθά στην εκμάθηση της γλώσσας και επιτρέπει συνεπή συμπεριφορά σε διαφορετικά συστήματα (π.χ. Windows, MacOS και Linux) και σε επιμέρους εφαρμογές που πρόκειται να αναπτυχθούν με την Python. Υπάρχουν πολλοί τρόποι με τους οποίους μπορεί να γίνει τοπική εγκατάσταση της Python σε ένα σύστημα, όπως με μεταφόρτωση και εκτέλεση του Python Install Manager από το https://www.python.org, με εγκατάσταση της διανομής Anaconda ή με εγκατάσταση της ελαφρότερης διανομής miniconda, με το εργαλείο uv και με άλλους τρόπους.

Στη συνέχεια θα περιγραφούν τρόποι εγκατάστασης της Python στα Windows, στο Linux και στο MacOS με έμφαση στην ευκολία. Επίσης, αξίζει να αναφερθεί ότι υπάρχει η δυνατότητα να εγκατασταθεί η Python στα Windows μέσα από το WSL (Windows Subsystem for Linux). Σε αυτή την περίπτωση εφόσον έχει εγκατασταθεί το WSL και κάποια εικονική μηχανή Linux, τότε μέσα στην εικονική μηχανή ακολουθείται ο τρόπος εγκατάστασης της Python για Linux που περιγράφεται παρακάτω.

Εύκολη εγκατάσταση της Python στα Windows (α' τρόπος)

Από το https://www.python.org μπορεί να μεταφορτωθεί ο “Python install manager” (π.χ., αρχείο python-manager-25.2.msix) και στη συνέχεια να γίνει η εγκατάσταση της Python με εκτέλεσή του. Κατά την εγκατάσταση, στην ερώτηση “Add command directory to your path now? [y/N]”, προτείνεται η συμπλήρωση της απάντησης y για να μπορεί να εντοπίζεται η python απευθείας από τη γραμμή εντολών (Command Prompt). Στις υπόλοιπες ερωτήσεις της εγκατάστασης επιλέγονται οι προκαθορισμένες απαντήσεις, πατώντας το πλήκτρο Enter. Ο έλεγχος ορθής εγκατάστασης μπορεί να γίνει εισάγοντας σε ένα νέο παράθυρο γραμμής εντολών είτε την εντολή python είτε την εντολή `py`. Το άνοιγμα ενός παραθύρου γραμμής εντολών μπορεί να γίνει ψάχνοντας για `cmd` στο μενού εκκίνησης των Windows ή πατώντας Win+R, πληκτρολογώντας `cmd` και στη συνέχεια Enter. Εφόσον, η εγκατάσταση έχει γίνει σωστά, η εκτέλεση της εντολής python θα εμφανίσει το REPL, όπως φαίνεται στο απόσπασμα οθόνης Εικόνα 1. Η έξοδος από το REPL μπορεί να γίνει στη συνέχεια γράφοντας `exit()` ή `exit` χωρίς παρενθέσεις από την έκδοση 3.13 και μετά.

```
C:\Users\auser>python
Python 3.14.2 (tags/v3.14.2:df79316, Dec  5 2025, 17:18:21) [MSC v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Εικόνα 1 – Έλεγχος ορθής εγκατάστασης της Python: ενεργοποίηση και απενεργοποίηση του REPL από τη γραμμή εντολών.

Δύο εναλλακτικοί τρόποι ελέγχου της ορθής εγκατάστασης της Python είναι οι ακόλουθοι:

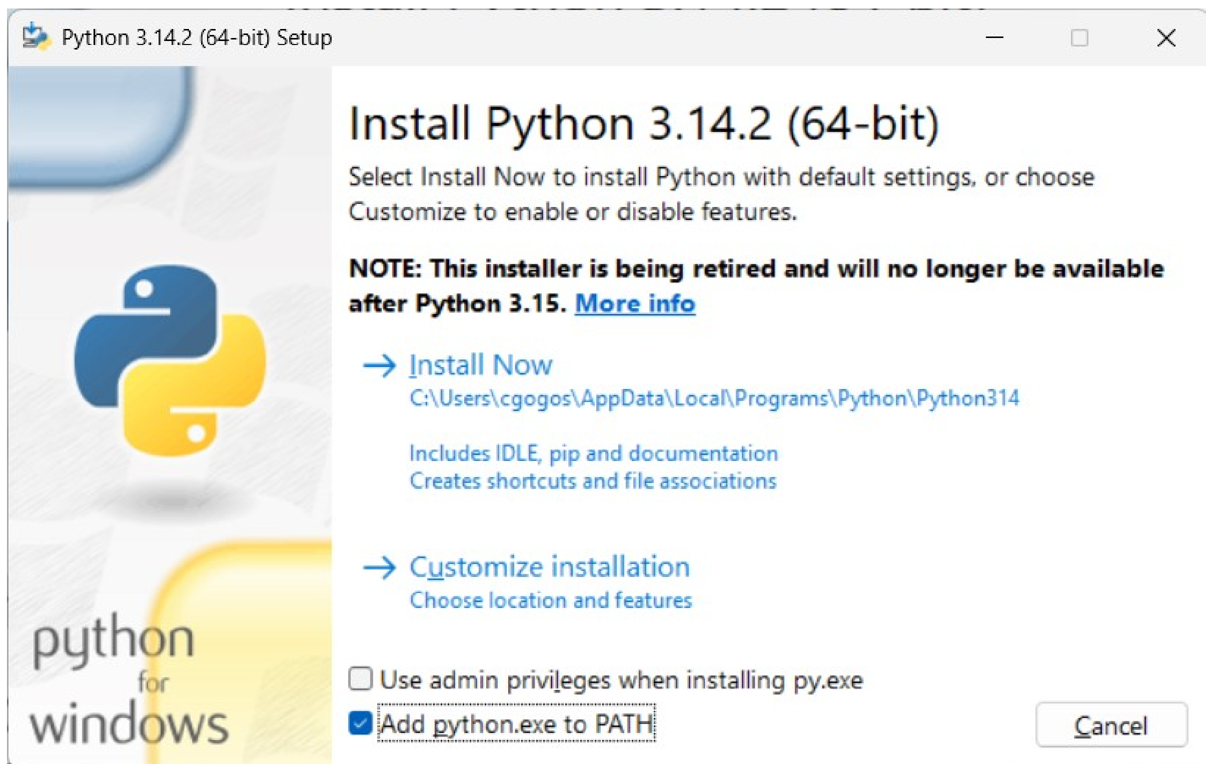
- Εκτέλεση της εντολής `python -V` ή της εντολής `python --version` που επιστρέφουν την έκδοση της Python που έχει εγκατασταθεί.
- Εκτέλεση της εντολής `python -c "import math; print(math.pi)"` που εκτελεί τον κώδικα μέσα στα εισαγωγικά και επιστρέφει την τιμή `3.141592653589793`.

Η εγκατάσταση της Python με τον Python install manager κρύβει, όπως πρέπει, την απευθείας πρόσβαση από τη γραμμή εντολών στο pip που είναι το βασικό εργαλείο εγκατάστασης εξωτερικών βιβλιοθηκών. Για να μπορεί να χρησιμοποιηθεί το pip θα πρέπει να δημιουργηθεί πρώτα σε έναν φάκελο ένα ιδεατό περιβάλλον (virtual environment) και μέσα σε αυτό να γίνει η εγκατάσταση βιβλιοθηκών. Η δημιουργία ιδεατών περιβαλλόντων θα περιγραφεί στη συνέχεια στις παραγράφους 1.2.3 και 1.2.4.

Εύκολη εγκατάσταση της Python στα Windows (β’ τρόπος)

Από το https://www.python.org μπορεί επίσης να μεταφορτωθεί ένας standalone installer όπως ο python-3.14.2-amd64.exe και να εκτελεστεί, επιλέγοντας “Add python.exe to PATH” και στη συνέχεια “Install now”. Θα πρέπει να δοθεί προσοχή έτσι ώστε να μεταφορτωθεί ο standalone installer που αντιστοιχεί στην αρχιτεκτονική του συστήματος για το οποίο γίνεται εγκατάσταση.

Οπότε, αν ο υπολογιστής έχει επεξεργαστή αρχιτεκτονικής INTEL, τότε το όνομα του αρχείου τελειώνει σε amd64 ενώ αν πρόκειται για υπολογιστή με επεξεργαστή αρχιτεκτονικής ARM το όνομα του αρχείου τελειώνει σε arm64. Αν και αυτός ο τρόπος εγκατάστασης πρόκειται να αποσυρθεί μετά την έκδοση 3.15 όπως φαίνεται στην Εικόνα 2, είναι ένας απλός τρόπος εγκατάστασης βολικός για πειραματισμό καθώς εγκαθιστά και δίνει απευθείας πρόσβαση μαζί με την Python στο pip και στο IDLE.



Εικόνα 2 – Αρχική οθόνη εγκατάστασης της Python στα Windows με τον standalone installer.

Εύκολη εγκατάσταση της Python στο Linux

Οι διανομές Linux (π.χ. Ubuntu, Debian, Mint, Fedora, Arch Linux) περιέχουν ήδη προεγκατεστημένη κάποια έκδοση της Python. Η δε έκδοση της Python που υπάρχει εγκατεστημένη μπορεί να διαπιστωθεί με την εντολή `python3 --version`. Η έκδοση της python που εκτελείται με την εντολή `python3`, δεν συνίσταται να αλλάξει καθώς μπορεί να χρησιμοποιείται από το ίδιο το σύστημα, αλλά αυτό δεν σημαίνει ότι δεν μπορεί να προστεθούν νέες εκδόσεις της Python που θα εκτελούνται είτε όταν θα δίνεται η εντολή `python` είτε με εντολές της μορφής `python3.14`. Ακολουθούν οι εντολές που πρέπει να δοθούν από το τερματικό για να επιτευχθεί αυτό:

```
sudo apt update
sudo apt upgrade -y
sudo apt install -y software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
```

```
sudo apt install -y python3.14 python3.14-venv python3.14-dev
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.14 50
sudo update-alternatives --config python
python --version
```

Α. 1 – Εγκατάσταση της έκδοσης 3.14 της Python σε περιβάλλον Linux (Ubuntu 24.04 LTS).

Εύκολη εγκατάσταση της Python στο MacOS

Όπως και στο Linux η Python είναι ήδη εγκατεστημένη στο MacOS. Ένας εύκολος τρόπος εγκατάστασης νέων εκδόσεων της Python στο MacOS είναι μέσω του homebrew (https://brew.sh/) που είναι ένας package manager που θα πρέπει να εγκατασταθεί πρώτα. Εφόσον υπάρχει εγκατεστημένο το homebrew, οι εντολές που θα πρέπει να δοθούν για την εγκατάσταση, για παράδειγμα της έκδοσης 3.14 της Python, είναι οι ακόλουθες:

```
brew update
brew install python@3.14
```

Α. 2 – Εγκατάσταση της έκδοσης 3.14 της Python σε MacOS, μέσω homebrew.

1.2.4. Ιδεατά περιβάλλοντα με `venv`

Συνίσταται η δημιουργία και χρήση των λεγόμενων ιδεατών περιβαλλόντων (virtual environments) στην Python έτσι ώστε η εγκατάσταση επιπλέον βιβλιοθηκών να μην επιβαρύνει τη βασική εγκατάσταση της Python και να είναι δυνατόν να επιλεγούν συγκεκριμένες εκδόσεις βιβλιοθηκών που μπορεί να απαιτούνται σε κάθε περίπτωση. Η δημιουργία ενός νέου ιδεατού περιβάλλοντος σε έναν φάκελο (folder) ή αλλιώς κατάλογο (directory), έστω με όνομα A γίνεται με την ακόλουθη εντολή που θα πρέπει να δοθεί από τη γραμμή εντολών με τρέχοντα φάκελο τον A, προκειμένου να δημιουργήσει ένα virtual environment με όνομα `myenv`.

```
python -m venv myenv
```

Η ενεργοποίηση του ιδεατού περιβάλλοντος γίνεται με την ακόλουθη εντολή στα Windows, (στη γραμμή εντολών):

```
myenv\Scripts\activate
```

Στο δε Linux και στο MacOS η εντολή ενεργοποίησης του ιδεατού περιβάλλοντος είναι:

```
source myenv/bin/activate
```

Μετά την ενεργοποίηση, το prompt δείχνει το όνομα του ιδεατού περιβάλλοντος μέσα σε παρενθέσεις, για παράδειγμα `(myenv)`. Η εγκατάσταση των εξωτερικών βιβλιοθηκών που εδώ ονομάζονται packages πλέον μπορεί να γίνει, με το pip, στο απομονωμένο ιδεατό περιβάλλον που έχει δημιουργηθεί. Το pip είναι ο προεπιλεγμένος διαχειριστής πακέτων της Python που χρησιμοποιείται για την εγκατάσταση, αναβάθμιση και αφαίρεση βιβλιοθηκών από αποθετήρια

όπως το PyPI (Python Package Index, https://pypi.org/) . Για παράδειγμα οι ακόλουθες εντολές εγκαθιστούν την εξωτερική βιβλιοθήκη requests και στη συνέχεια εκτελούν κώδικα που εμφανίζει την έκδοση της requests:

```
pip install requests
python -c "import requests; print(requests.__version__)"
```

Η απενεργοποίηση του ιδεατού περιβάλλοντος γίνεται με την εντολή deactivate τόσο σε Windows, όσο και σε Linux, MacOS.

1.2.5. Ιδεατά περιβάλλοντα με uv

Εναλλακτικά του venv, συνίσταται η χρήση του εργαλείου uv, το οποίο αποτελεί ένα σύγχρονο, ιδιαίτερα γρήγορο package και environment manager για την Python. Το uv επιτρέπει τη δημιουργία και διαχείριση ιδεατών περιβαλλόντων, καθώς και την εγκατάσταση βιβλιοθηκών, με απλούστερες και ταχύτερες εντολές, διατηρώντας την απομόνωση από τη βασική εγκατάσταση της Python και επιτρέποντας τον ακριβή έλεγχο εκδόσεων.

Αρχικά, θα πρέπει να εγκατασταθεί το uv που γίνεται δίνοντας μια μόνο εντολή εγκατάστασης από powershell γραμμή εντολών σε Windows ή από μια γραμμή εντολών σε Lixux, MacOS. Οδηγίες για την εγκατάστασή του uv βρίσκονται στο https://docs.astral.sh/uv/getting-started/installation/.

Η αρχικοποίηση ενός νέου Python project σε έναν φάκελο, έστω με όνομα A, γίνεται από τη γραμμή εντολών με τρέχοντα φάκελο τον A, με την ακόλουθη εντολή:

```
uv init
```

Η εντολή αυτή δημιουργεί τα βασικά αρχεία του project (όπως το pyproject.toml) και προετοιμάζει το περιβάλλον για τη διαχείριση εξαρτήσεων από συγκεκριμένες βιβλιοθήκες και εκδόσεις τους. Η προσθήκη εξωτερικών βιβλιοθηκών στο project γίνεται με την εντολή uv add, η οποία καταγράφει τις εξαρτήσεις στο pyproject.toml. Για παράδειγμα, η ακόλουθη εντολή προσθέτει τη βιβλιοθήκη requests:

```
uv add requests
```

Εφόσον είναι επιθυμητό μπορεί να δηλωθεί η έκδοση της Python που θα χρησιμοποιηθεί στο project με την ακόλουθη εντολή (π.χ. για την εγκατάσταση της έκδοσης 3.14 της Python):

```
uv python pin 3.14
```

Εφόσον έχει δηλωθεί η έκδοση της Python, η δημιουργία του ιδεατού περιβάλλοντος και η εγκατάσταση τυχόν βιβλιοθηκών γίνεται με την εντολή:

```
uv sync
```

Το uv χρησιμοποιεί την έκδοση της Python που ήδη υπάρχει εγκατεστημένη, αλλιώς αν η συγκεκριμένη έκδοση δεν υπάρχει, τότε το uv προχωρά στην αυτόματη εγκατάστασή της.

Η ενεργοποίηση και απενεργοποίηση του ιδεατού περιβάλλοντος γίνεται με τον ίδιο τρόπο όπως και με το venv. Το προκαθορισμένο όνομα του φακέλου στον οποίο δημιουργείται το virtual environment είναι .venv, οπότε οι εντολή ενεργοποίησης για τα Windows είναι:

```
.venv\Scripts\activate
```

ενώ για Linux και MacOS είναι:

```
source .venv/bin/activate
```

Από τη στιγμή που έχει ενεργοποιηθεί το ιδεατό περιβάλλον μπορεί να ελεγχθεί η ορθή εγκατάσταση με την εντολή uv run python --version που θα πρέπει να επιστρέψει την έκδοση της Python ή δίνοντας την ακόλουθη εντολή:

```
python -c "import requests; print(requests.__version__)"
```

που θα επιστρέψει την έκδοση της εξωτερικής βιβλιοθήκης requests που εγκαταστάθηκε με τις εντολές που δόθηκαν παραπάνω.

1.2.6. Συγγραφή και εκτέλεση προγράμματος με το IDLE

Το IDLE (Integrated Development and Learning Environment) είναι ένα ελαφρύ περιβάλλον ανάπτυξης της Python, που διανέμεται μαζί με την τυπική εγκατάσταση της γλώσσας και έχει σχεδιαστεί κυρίως για εκπαιδευτική χρήση και για αρχάριους προγραμματιστές. Παρέχει ένα δικό του REPL για άμεση εκτέλεση εντολών, έναν βασικό επεξεργαστή κειμένου για συγγραφή κώδικα με επισήμανση σύνταξης, αυτόματες εσοχές και έλεγχο λαθών, καθώς και δυνατότητα εκτέλεσης και αποσφαλμάτωσης προγραμμάτων σε ένα ενιαίο γραφικό περιβάλλον. Παρότι δεν προορίζεται για μεγάλα ή σύνθετα προγράμματα, το IDLE μπορεί να χρησιμοποιηθεί για εκμάθηση της Python καθώς απαιτεί ελάχιστες ρυθμίσεις και προσφέρει άμεση ανατροφοδότηση στον χρήστη.

Αν η εγκατάσταση έχει γίνει στα Windows είναι πιθανό να έχει δημιουργηθεί εικονίδιο εφαρμογής για το IDLE από το οποίο μπορεί να γίνει η εκκίνησή του. Σε διαφορετική περίπτωση το IDLE μπορεί να εκκινηθεί με την εντολή:

```
python -m idlelib
```

Η αρχική οθόνη του IDLE φαίνεται στην Εικόνα 3

```
IDLE Shell 3.14.2
File  Edit  Shell  Debug  Options  Window  Help
Python 3.14.2 (main, Dec 17 2025, 21:10:36) [MSC v.1944 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>> import math
>>> x = math.sqrt(2)
>>> x
1.4142135623730951
>>> x, x**2, x**4
(1.4142135623730951, 2.0000000000000004, 4.000000000000001)
>>> |
Ln: 9  Col: 0
```

Εικόνα 3 – Η αρχική οθόνη, με το REPL, του IDLE.

Από το μενού του IDLE Shell με την επιλογή File → New File ανοίγει ο επεξεργαστής κειμένου όπου μπορεί να εισαχθεί ένα πρόγραμμα σαν αυτό που φαίνεται στην Εικόνα 4. Η εκτέλεση με επιλογή από το μενού Run → Run module, θα ζητήσει πρώτα τον ορισμό ενός ονόματος αρχείου (π.χ. test.py) για το αρχείο κώδικα που δημιουργήθηκε και η εκτέλεση θα συνεχιστεί στο IDLE Shell, όπως φαίνεται στην ίδια εικόνα.

```
test.py - C:/Users/cgogos/Desktop/test.py (3.14.2)
File  Edit  Format  Run  Options  Window  Help
print("Παράδειγμα στο IDLE!")
x = int(input("Δώσε έναν αριθμό: "))
print("Διπλάσιος:", 2 * x)
Ln: 4  Col: 0
```

```
IDLE Shell 3.14.2
File  Edit  Shell  De
>>> x = math.sqr
>>> x
1.4142135623
>>> x, x**2, x**
(1.414213562
>>>
=================== RESTART: C:/Users/cgogos/Desktop/test.py ===================
Παράδειγμα στο IDLE!
Δώσε έναν αριθμό: 21
Διπλάσιος: 42
>>>
Ln: 8  Col: 0
```

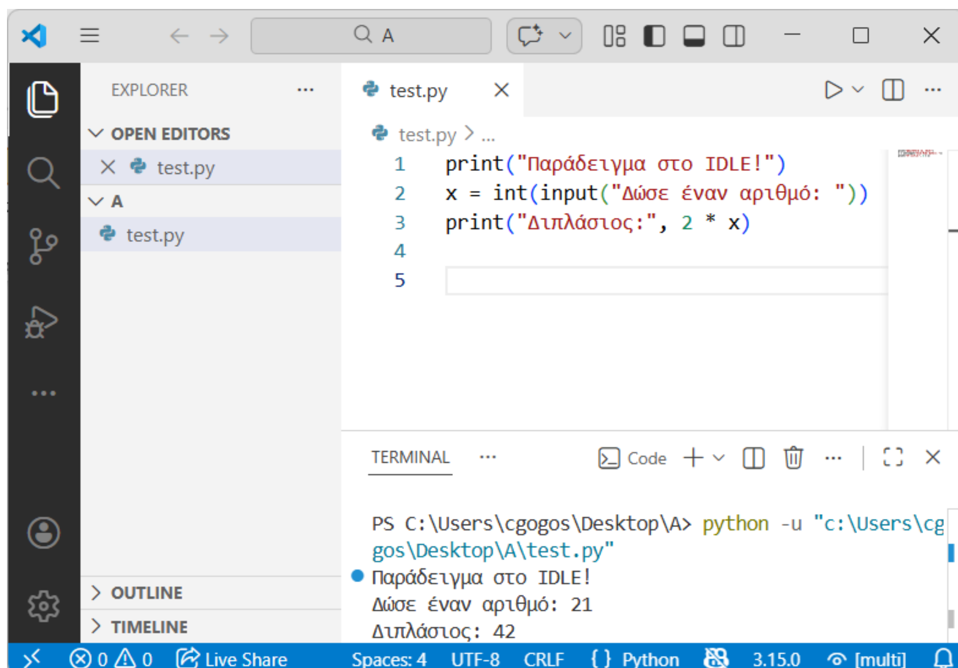
Εικόνα 4 – Εκτέλεση κώδικα που έχει γραφεί στον επεξεργαστή κειμένου του IDLE.

1.2.7. Συγγραφή και εκτέλεση προγράμματος με το Visual Studio Code

Το Visual Studio Code (VSC) είναι ένας σύγχρονος, ελαφρύς αλλά ιδιαίτερα ισχυρός επεξεργαστής κειμένου από την Microsoft, που χρησιμοποιείται για ανάπτυξη λογισμικού σε διάφορες γλώσσες προγραμματισμού, με έμφαση στην επεκτασιμότητα και στην παροχή διευκολύνσεων και εργαλείων στον προγραμματιστή. Βασικές δυνατότητες του VSC είναι η επισήμανση σύνταξης, η αυτόματη συμπλήρωση, το ενσωματωμένο τερματικό, η δυνατότητα debugging και άλλα. Μέσω

επεκτάσεων μπορεί να μετατραπεί σε πλήρες IDE με υποστήριξη για Git, εικονικά περιβάλλοντα, linters, εργαλεία ανάλυσης κώδικα και άλλα. Χρησιμοποιείται τόσο στην εκπαίδευση όσο και σε επαγγελματικά έργα, από μικρά προγράμματα έως μεγάλα έργα λογισμικού που αναπτύσσονται από πολλούς συνεργαζόμενους προγραμματιστές.

Το VSC είναι δωρεάν και εγκαθίσταται εύκολα σε Windows, Linux και MacOS ακολουθώντας τις οδηγίες στην επίσημη σελίδα του. Από τη στιγμή που έχει εγκατασταθεί μπορεί να ανοίξει έναν φάκελο στον οποίο υπάρχουν τα αρχεία που πρόκειται να επεξεργαστούν και να εκτελεστούν. Αν ο φάκελος είναι κενός τα αρχεία κώδικα μπορούν να δημιουργηθούν. Συνεπώς, αν ο χρήστης έχει δημιουργήσει έναν φάκελο με όνομα A, για παράδειγμα στην επιφάνεια εργασίας, τότε με την επιλογή μενού File → Open Folder μπορεί να επιλεγεί ο φάκελος A. Στη συνέχεια, μέσα από το VSC, μπορεί να δημιουργηθεί ένα νέο αρχείο με την επιλογή File → New File και να δοθεί ως όνομα αρχείου, για παράδειγμα, `test.py`. Συμπληρώνοντας τον κώδικα που φαίνεται στην Εικόνα 5 μπορεί να επιλεγεί να γίνει αποθήκευση του αρχείου `test.py` και στη συνέχεια να επιλεγεί Run → Run Without Debugging προκειμένου να εκτελεστεί ο κώδικας. Κατά την πρώτη συγγραφή και εκτέλεση κώδικα `.py`, το VSC ανιχνεύει ότι πρόκειται για κώδικα Python και προτείνει την εγκατάσταση βιβλιοθηκών που επιτρέπουν την εκτέλεση του, που ο χρήστης πρέπει να αποδεχθεί. Επιπλέον, η ρύθμιση File → Autosave του VSC είναι χρήσιμο να ενεργοποιηθεί διότι το αρχείο κώδικα θα αποθηκεύεται πριν εκτελεστεί, αποφεύγοντας την εκτέλεση διαφορετικού κώδικα από αυτόν που εμφανίζεται στην οθόνη.



Εικόνα 5 – Εκτέλεση κώδικα Python που έχει γραφεί στο Visual Studio Code.

1.2.8. Google Colab

Το Google Colab (Colaboratory) είναι ένα διαδικτυακό περιβάλλον ανάπτυξης βασισμένο σε Jupyter Notebooks (βλ. κεφ. 6), το οποίο επιτρέπει την εκτέλεση κώδικα Python απευθείας από τον φυλλομετρητή, χωρίς να απαιτείται τοπική εγκατάσταση της Python ή των βιβλιοθηκών της. Διατίθεται από τη Google και για να έχει κανείς πρόσβαση σε αυτό θα πρέπει να διαθέτει λογαριασμό Google. Παρέχει έτοιμο περιβάλλον με προεγκατεστημένες δημοφιλείς βιβλιοθήκες για μηχανική μάθηση και ανάλυση δεδομένων (όπως NumPy, pandas, matplotlib, TensorFlow και PyTorch), καθώς και πρόσβαση σε υπολογιστικούς πόρους όπως GPUs (Graphical Processing Units) και TPUs (Tensor Processing Units) για επιτάχυνση υπολογισμών. Το Colab διευκολύνει τη συνεργασία, επιτρέποντας σε πολλούς χρήστες να εργάζονται στο ίδιο notebook σε πραγματικό χρόνο, ενώ η αποθήκευση αρχείων γίνεται στο Google Drive. Αποτελεί ένα εργαλείο που μπορεί να βοηθήσει στη γρήγορη πρωτοτυποποίηση και παρουσίαση αποτελεσμάτων, ιδιαίτερα σε περιβάλλοντα διδασκαλίας και έρευνας.

Για τη δημιουργία ενός Google Colab σημειωματαρίου, ο χρήστης συνδέεται με το λογαριασμό του Google, μεταβαίνει στη σελίδα του Google Colab (https://colab.research.google.com/) και επιλέγει τη δημιουργία νέου σημειωματαρίου (notebook), οπότε ανοίγει ένα διαδραστικό περιβάλλον στο οποίο το έγγραφο αποτελείται από κελιά. Σε κάθε κελί μπορεί να γράψει κώδικα Python ή κείμενο (Markdown), και η εκτέλεση του κώδικα γίνεται πατώντας το κουμπί εκτέλεσης (▶) δίπλα στο κελί ή με το συνδυασμό πλήκτρων Shift+Enter, οπότε το αποτέλεσμα εμφανίζεται αμέσως κάτω από το κελί, επιτρέποντας την τμηματική εκτέλεση και τον πειραματισμό με τον κώδικα. Για παράδειγμα στην Εικόνα 6 παρουσιάζεται ένα Google Colab σημειωματάριο όπου σε ένα κελί του έχει συμπληρωθεί κώδικας Python που δημιουργεί 100 τυχαίες τιμές, υπολογίζει βασικά στατιστικά μεγέθη για τις τιμές αυτές και απεικονίζει τις τιμές γραφικά σε ένα ιστόγραμμα. Αξίζει να σημειωθεί ότι η εξωτερική βιβλιοθήκη matplotlib που αναλαμβάνει το σχεδιασμό του γραφήματος είναι ήδη προεγκατεστημένη και χρησιμοποιείται χωρίς να προηγηθούν εντολές εγκατάστασης.

```python
import random
import matplotlib.pyplot as plt
# Δημιουργία 100 τυχαίων αριθμών
numbers = [random.randint(1, 100) for _ in range(100)]
# Υπολογισμός βασικών στατιστικών
print("Ελάχιστο:", min(numbers))
print("Μέγιστο:", max(numbers))
print("Μέσος όρος:", sum(numbers) / len(numbers))
# Οπτικοποίηση
plt.figure(figsize=(2, 1.5))
plt.hist(numbers, bins=10)
plt.title("Κατανομή τυχαίων αριθμών")
plt.xlabel("Τιμή")
plt.ylabel("Συχνότητα")
plt.show()
```

```
Ελάχιστο: 2
Μέγιστο: 100
Μέσος όρος: 52.58
```

Εικόνα 6 – Ένα Google Colab σημειωματάριο που εμφανίζει κώδικα Python, αποτελέσματα και ένα γράφημα.

Εναλλακτικές λύσεις του Google Colab, για διατήρηση σημειωματαρίων στο cloud, είναι τα Kaggle Notebooks (https://www.kaggle.com/code), το Binder (https://mybinder.org/), το try-jupyter (https://jupyter.org/try-jupyter/lab/index.html) και άλλες.

1.2.9. Πηγές πληροφόρησης για την Python

Υπάρχουν πολλές πηγές πληροφόρησης για την Python. Ορισμένες από τις πλέον σημαντικές είναι η επίσημη τεκμηρίωση της γλώσσας και το επίσημο tutorial της γλώσσας. Ακολουθεί μια σχετική λίστα:

- https://docs.python.org/3/, επίσημη τεκμηρίωση της γλώσσας.
- https://docs.python.org/3/tutorial/index.html, επίσημο tutorial της Python.
- https://openstax.org/details/books/introduction-python-programming, βιβλίο για εισαγωγή στον προγραμματισμό από τον οργανισμό OpenStax.
- https://learnxinyminutes.com/python/, σύντομο tutorial για την Python.
- https://codapi.org/try/python/, σύντομο tutorial για την Python με εκτελέσιμο κώδικα.

- https://genepy.org/, ασκήσεις σε Python με αυτόματο έλεγχο ορθότητας και παρακολούθηση προόδου.

Επίσης, υπάρχουν πολλά βιβλία που έχουν γραφεί για την Python, όπως είναι τα [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13] που παρατίθενται στη βιβλιογραφία και τα οποία διατίθενται ελεύθερα από τους δημιουργούς τους.

1.3. Το REPL

Εφόσον έχει προηγηθεί η εγκατάσταση της Python, το REPL μπορεί να ξεκινήσει απλά πληκτρολογώντας στην γραμμή εντολών την εντολή python. Τότε, θα εμφανιστεί ένα περιβάλλον αλληλεπίδρασης του χρήστη με την Python παρόμοιο με το απόσπασμα κώδικα Α. 3. Στο περιβάλλον αυτό ο χρήστης μπορεί να εισάγει εντολές στην προτροπή >>> και να λάβει αποτελέσματα.

```
C:\Users\User>python
Python 3.14.2 (tags/v3.14.2:df79316, Dec  5 2025, 17:18:21) [MSC v.1944 64 bit (AMD64)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Α. 3 - Ενεργοποίηση του REPL.

Τα αρχικά REPL σημαίνουν:

- Read – λήψη εισόδου από τον χρήστη, δηλαδή εισαγωγή python κώδικα και δεδομένων από τον χρήστη.
- Evaluate – αποτίμηση κώδικα.
- Print – εκτύπωση αποτελεσμάτων.
- Loop – επανάληψη της διαδικασίας μέχρι να γίνει έξοδος από τον χρήστη.

Το REPL είναι χρήσιμο για γρήγορο πειραματισμό, αποσφαλμάτωση κώδικα, δοκιμή συναρτήσεων βιβλιοθηκών, κλήση συστήματος βοήθειας, π.χ. `>>> help(str)` και άλλα. Ειδικότερα, επιτρέπει την άμεση εκτέλεση κώδικα, χωρίς ανάγκη να γραφεί πλήρες πρόγραμμα (script). Η πλοήγηση στο ιστορικό των εντολών γίνεται με τα βελάκια πάνω και κάτω του πληκτρολογίου, ενώ επιτρέπεται αλλά και διευκολύνεται η εισαγωγή εντολών πολλαπλών γραμμών. Η εκκαθάριση της οθόνης γίνεται με Ctrl+L ενώ η έξοδος από το REPL γίνεται πληκτρολογώντας `exit()` ή με Ctrl+D. Από την έκδοση 3.13 της Python το REPL διαθέτει syntax highlighting και η έξοδος μπορεί να γίνει πληκτρολογώντας `exit`, χωρίς παρενθέσεις.

Το REPL μπορεί να χρησιμοποιηθεί ως μια ισχυρή αριθμομηχανή που μπορεί να παράξει αποτελέσματα με πολύ μεγάλο αριθμό ψηφίων και υψηλή ακρίβεια. Στο απόσπασμα κώδικα Α. 4 φαίνονται μερικά παραδείγματα αριθμητικών πράξεων. Στα παραδείγματα αυτά ο τελεστής ** είναι ο τελεστής της ύψωσης σε δύναμη, ενώ για πράξεις με δεκαδικούς με μεγάλη ακρίβεια χρησιμοποιείται το Decimal.

```
>>> 2**100
1267650600228229401496703205376
>>> 1/7
0.14285714285714285
>>> from decimal import Decimal, getcontext
>>> getcontext().prec = 50
>>> Decimal(1)/Decimal(7)
Decimal('0.14285714285714285714285714285714285714285714285714')
```

Α. 4 – Αριθμητικές πράξεις στο REPL.

1.3.1. Παραδείγματα στο REPL με modules της Python Standard Library

Στο REPL μπορούν να χρησιμοποιηθούν modules της Python είτε από την τυπική βιβλιοθήκη της (Python Standard Library) είτε από εξωτερικές βιβλιοθήκες που έχουν εγκατασταθεί. Στη συνέχεια παρουσιάζονται παραδείγματα από τα modules math, random, datetime και calendar της τυπικής βιβλιοθήκης της Python.

Το module math περιέχει βασικές μαθηματικές συναρτήσεις και σταθερές. Για να χρησιμοποιηθούν οι συναρτήσεις και οι σταθερές του, το module πρέπει να εισαχθεί με το import math. Στο απόσπασμα κώδικα Α. 5 παρουσιάζονται μερικά παραδείγματα, όπου pi είναι ο αριθμός π, e είναι η μαθηματική σταθερά e (αριθμός του Euler), inf είναι το άπειρο, sqrt() είναι η συνάρτηση τετραγωνικής ρίζας, factorial() είναι η συνάρτηση παραγοντικού και comb() είναι η συνάρτηση εύρεσης πλήθους συνδυασμών από ένα σύνολο στοιχείων για ένα συγκεκριμένο πλήθος επιλεχθέντων στοιχείων.

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>> math.inf
inf
>>> math.sqrt(2)
1.4142135623730951
>>> math.factorial(5)
120
>>> math.factorial(50)
30414093201713378043612608166064768844377641568960512000000000000
>>> math.comb(6,2)
15
```

Α. 5 – Σταθερές και συναρτήσεις του module math.

Το module random περιέχει συναρτήσεις για παραγωγή τυχαίων τιμών. Στα παραδείγματα στο απόσπασμα κώδικα A. 6, η συνάρτηση uniform() επιστρέφει μια τυχαία πραγματική τιμή εντός δύο ορίων, η συνάρτηση randint() επιστρέφει μια ακέραια τιμή ανάμεσα σε δύο ακέραια όρια (συμπεριλαμβανομένων των δύο ορίων) και η choice() επιστρέφει μια τυχαία τιμή από τις τιμές της λίστας που δέχεται ως όρισμα.

```
>>> import random
>>> random.uniform(0,1)
0.12609970351309063
>>> random.uniform(0,1)
0.40458192428704964
>>> random.randint(1,6)
2
>>> random.randint(1,6)
3
>>> random.choice(['κορώνα', 'γράμματα'])
'κορώνα'
>>> random.choice(['κορώνα', 'γράμματα'])
'γράμματα'
```
A. 6 – Συναρτήσεις του module random.

Το module datetime επιτρέπει τον εύκολο χειρισμό ημερομηνιών που μπορούν να περιέχουν και πληροφορία χρόνου. Στα παραδείγματα στο απόσπασμα κώδικα A. 7 δημιουργείται ένα αντικείμενο ημερομηνίας με την date() και ένα αντικείμενο ημερομηνίας και ώρας με την datetime(). Επίσης, χρησιμοποιείται η συνάρτηση timedelta() για δημιουργία μιας νέας ημερομηνίας με βάση μια υπάρχουσα ημερομηνία και μια χρονική διαφορά που εκφράζεται σε αριθμό ημερών και ωρών.

```
>>> import datetime
>>> d = datetime.date(1990, 5, 17)
>>> d
datetime.date(1990, 5, 17)
>>> dt = datetime.datetime(1990, 5, 17, 14, 30)
>>> dt
datetime.datetime(1990, 5, 17, 14, 30)
>>> delta = datetime.timedelta(days=10, hours=5)
>>> dt + delta
datetime.datetime(1990, 5, 27, 19, 30)
>>> dt.year, dt.month, dt.day
(1990, 5, 17)
```
A. 7 – Συναρτήσεις του module datetime.

Ως ένα ακόμα παράδειγμα module, παρουσιάζεται το calendar που μπορεί να χρησιμοποιηθεί για την εύκολη δημιουργία ενός ημερολογίου όπως φαίνεται στο απόσπασμα κώδικα A. 8.

```
>>> import calendar
>>> cal = calendar.month(2026, 1)
>>> print(cal)
    January 2026
Mo Tu We Th Fr Sa Su
          1  2  3  4
```

```
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```
Α. 8 – Το module calendar.

1.3.2. Άσκηση 1, εντολές στο REPL

Εκτελέστε τις ακόλουθες εντολές (Α. 9) στο REPL, μια προς μια και παρατηρήστε τα αποτελέσματα που λαμβάνετε. Για όποια εντολή χρειαστεί, αναζητήστε στο διαδίκτυο για ποιο λόγο εμφανίζονται τα αποτελέσματα που επιστρέφονται.

```
>>> 2 + 3 * 4
>>> "Python"[::-1]
>>> [x**2 for x in range(5)]
>>> import random; [random.randint(1,6) for _ in range(5)]
>>> random.choice(["πέτρα", "ψαλίδι", "χαρτί"])
>>> sorted([7, 2, 3, 1, 6])
>>> import this
>>> import __hello__; __hello__.main()
>>> from __future__ import braces
>>> import antigravity
```
Α. 9 – Μερικά παραδείγματα εντολών που μπορούν να δοθούν στο REPL.

Η λύση της άσκησης υπάρχει στο Παράρτημα Α'.

1.4. Μεταβλητές

Στην Python οι μεταβλητές (variables) είναι ονοματισμένες αναφορές προς αντικείμενα που βρίσκονται στη μνήμη του υπολογιστή. Λόγω του dynamic typing, δεν δηλώνονται τύποι δεδομένων, ενώ ο τύπος δεδομένων των μεταβλητών μπορεί να αλλάζει καθώς εκτελείται ο κώδικας. Σε άλλες γλώσσες προγραμματισμού (π.χ. C, Java), που είναι static typing γλώσσες, οι τύποι των μεταβλητών δηλώνονται και δεν μπορούν να αλλάξουν κατά την εκτέλεση του προγράμματος. Στην Python η ανάθεση τιμών σε μεταβλητές γίνεται με τον τελεστή =, ο δε τύπος της μεταβλητής προκύπτει από τον τύπο του αντικειμένου στο δεξί μέρος της ανάθεσης.

Οι μεταβλητές εκτός από τιμή (value), έχουν όνομα (name), τύπο (type) και αναγνωριστικό (id). Η συνάρτηση `type()` επιστρέφει το τρέχοντα τύπο μιας μεταβλητής ενώ η συνάρτηση `id()` επιστρέφει το τρέχοντα αναγνωριστικό θέσης μνήμης μιας μεταβλητής. Τόσο ο τύπος όσο και το αναγνωριστικό μιας μεταβλητής μπορεί να αλλάζει κατά την εκτέλεση ενός προγράμματος όπως φαίνεται και στο παράδειγμα του αποσπάσματος κώδικα Α. 10.

```
>>> x = 42
>>> type(x)
<class 'int'>
>>> id(x)
140714427840920
```

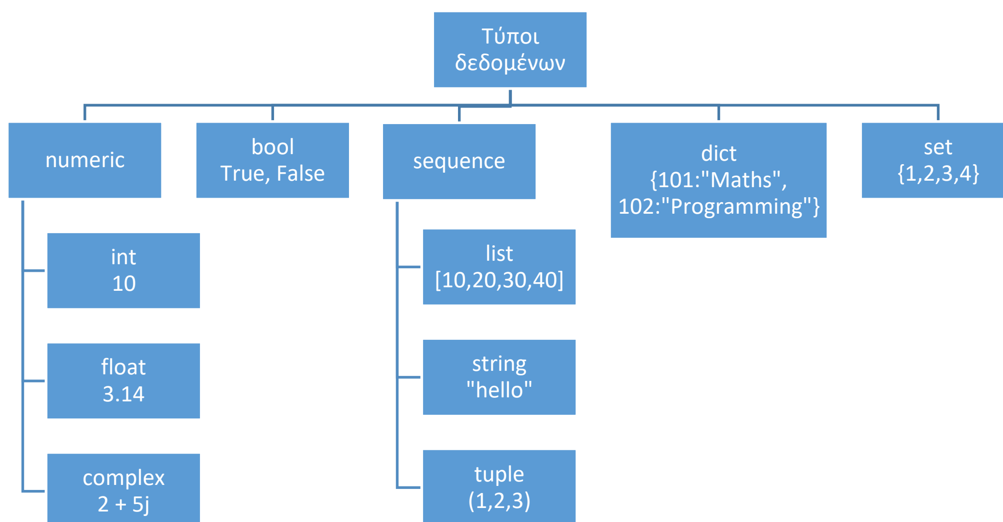
```
>>> x = "python"
>>> type(x)
<class 'str'>
>>> id(x)
2024405253648
```

Α. 10 – Αλλαγή τύπου δεδομένων μεταβλητής κατά την εκτέλεση διαδοχικών εντολών.

Τα ονόματα των μεταβλητών ξεκινούν με γράμμα ή με κάτω παύλα _ (underscore) και μπορούν να περιέχουν γράμματα ψηφία και την κάτω παύλα. Αν και μπορούν να χρησιμοποιηθούν ελληνικοί χαρακτήρες στα ονόματα των μεταβλητών, δεν συνίσταται κάτι τέτοιο. Στις μεταβλητές, αλλά και γενικότερα στα λεγόμενα αναγνωριστικά, υπάρχει διάκριση πεζών κεφαλαίων.

1.4.1. Τύποι δεδομένων της Python

Ένας τύπος (type) είναι ένα όνομα για μια συλλογή από τιμές με κοινά χαρακτηριστικά. Η Python διαθέτει διάφορους ενσωματωμένους τύπους δεδομένων που χρησιμοποιούνται για την αναπαράσταση και αποθήκευση διαφόρων μορφών δεδομένων. Οι πλέον βασικοί από αυτούς παρουσιάζονται στην Εικόνα 7.



Εικόνα 7 – Βασικοί τύποι δεδομένων της Python.

Η κατηγορία αριθμητικών τύπων (numeric) περιλαμβάνει τους ακέραιους αριθμούς (`int`), τους αριθμούς κινητής υποδιαστολής (`float`), που αναπαριστούν πραγματικούς αριθμούς με δεκαδικό μέρος, και τους μιγαδικούς αριθμούς (`complex`) που αποτελούνται από πραγματικό και φανταστικό μέρος. Στο απόσπασμα κώδικα Α. 11 παρουσιάζεται η ανάθεση αριθμητικών τιμών σε μεταβλητές καθώς και ορισμένες αριθμητικές πράξεις μαζί με τα αποτελέσματά τους.

```
>>> a = 10
>>> b = 3.14
>>> c = 2 + 5j
>>> print("a =", a, "τύπος:", type(a))
```

```
a = 10 τύπος: <class 'int'>
>>> print("b =", b, "τύπος:", type(b))
b = 3.14 τύπος: <class 'float'>
>>> print("c =", c, "τύπος:", type(c))
c = (2+5j) τύπος: <class 'complex'>
>>> a * 2
20
>>> b / 2
1.57
>>> c + (1 - 2j)
(3+3j)
```
Α. 11 – Παραδείγματα με αριθμητικούς τύπους δεδομένων.

Ο τύπος δεδομένων `bool` λαμβάνει 2 μόνο τιμές, `True` και `False`. Στο απόσπασμα κώδικα Α. 12 παρουσιάζεται το πως προκύπτει μια λογική τιμή (`False`) από μια σύγκριση τιμών.

```
>>> a, b = 1, 2
>>> c = a > b
>>> c
False
>>> type(c)
<class 'bool'>
```
Α. 12 – Παράδειγμα με τον λογικό τύπο δεδομένων bool.

Η κατηγορία τύπων ακολουθία (sequence) υπονοεί ότι στους συγκεκριμένους τύπους, λίστα (`list`), συμβολοσειρά (`str`) και πλειάδα (`tuple`), υπάρχει σειρά στα στοιχεία που περιέχουν. Δηλαδή κάθε στοιχείο κατέχει συγκεκριμένη θέση μέσα στη δομή και μπορεί να προσπελαστεί μέσω ενός δείκτη (index). Οι δείκτες στην Python ξεκινούν από το 0, γεγονός που σημαίνει ότι το πρώτο στοιχείο μιας ακολουθίας έχει δείκτη 0, το δεύτερο 1 κ.ο.κ. Στους τύπους ακολουθίας υποστηρίζονται επίσης λειτουργίες όπως η προσπέλαση στοιχείων, η αποκοπή τμημάτων (slicing), η διάσχισή τους (π.χ. με την εντολή `for`) και ο υπολογισμός του πλήθους των στοιχείων με τη συνάρτηση `len()`. Στο απόσπασμα κώδικα Α. 13 φαίνονται παραδείγματα προσπέλασης στοιχείων σε λίστα, συμβολοσειρά και πλειάδα με χρήση δεικτών. Οι συμβολοσειρές περιγράφονται αναλυτικότερα στην παράγραφο 1.7, οι λίστες στην παράγραφο 3.1 και οι πλειάδες στην παράγραφο 4.1.

```
>>> numbers = [10,20,30,40]
>>> numbers[0], numbers[2]
(10, 30)
>>> text = "hello"
>>> text[0], text[2]
('h', 'l')
>>> point3d = (1,2,3)
>>> point3d[0], point3d[2]
(1, 3)
```
Α. 13 – Παραδείγματα με τύπους δεδομένων ακολουθίας.

Δύο ακόμα βασικοί τύποι δεδομένων είναι το σύνολο (`set`) και το λεξικό (`dict`) που περιγράφονται στις παραγράφους 4.2 και 4.3, αντίστοιχα. Εδώ, αρκεί να αναφερθεί ότι τα σύνολα

περιέχουν τιμές χωρίς διπλότυπα, ενώ τα λεξικά περιέχουν ζεύγη τιμών της μορφής κλειδί: τιμή και χρησιμοποιούνται πολύ συχνά στην πράξη. Στο απόσπασμα κώδικα A. 14 παρουσιάζεται ένα απλό παράδειγμα χρήσης συνόλου και λεξικού με το λεξικό να έχει δύο ζεύγη τιμών με κλειδιά 101 και 102 αντίστοιχα.

```
>>> s = {1,2,3,3,3,4}
>>> s, type(s)
({1, 2, 3, 4}, <class 'set'>)
>>> d = {101:'Maths', 102:'Programming'}
>>> d[101], d[102], type(d)
('Maths', 'Programming', <class 'dict'>)
```

A. 14 – Παράδειγμα με σύνολο και λεξικό.

1.4.2. Αριθμητικοί τελεστές

Οι αριθμητικοί τελεστές είναι οι πράξεις των μαθηματικών. Οι αριθμητικοί τελεστές της Python, μαζί με παραδείγματα πράξεων, φαίνονται στον Πίνακας 2. Ιδιαίτερη αναφορά πρέπει να γίνει στον τελεστή ύψωσης σε δύναμη που είναι ο ** και στους τελεστές πηλίκου ακέραιας διαίρεσης και υπολοίπου ακέραιας διαίρεσης που είναι οι // και % αντίστοιχα, καθώς δεν αντιστοιχούν σε σύμβολα των μαθηματικών αλλά έχουν επιλεγεί λόγω του έχει επικρατήσει η χρήση τους και προκειμένου να μπορούν να γραφούν με ευκολία στο πληκτρολόγιο.

Σύμβολο αριθμητικού τελεστή	Πράξη	Παράδειγμα στο REPL
+	Πρόσθεση	`>>> 1 + 2` `3`
-	Αφαίρεση	`>>> 1 - 2` `-1`
*	Πολλαπλασιασμός	`>>> 2 * 3` `6`
/	Διαίρεση	`>>> 10 / 4` `2.5`
//	Πηλίκο ακέραιας διαίρεσης	`>>> 10 // 3` `3`
%	Υπόλοιπο ακέραιας διαίρεσης	`>>> 10 % 3` `1`
**	Ύψωση σε δύναμη	`>>> 2 ** 10` `1024`

Πίνακας 2 – Αριθμητικοί τελεστές.

Για την προτεραιότητα των αριθμητικών τελεστών ισχύουν οι γνωστοί κανόνες προτεραιότητας των αριθμητικών πράξεων, δηλαδή, πρώτα γίνονται οι πράξεις εντός παρενθέσεων, μετά γίνεται η ύψωση σε δύναμη, μετά ο πολλαπλασιασμός και η διαίρεση και τέλος η πρόσθεση και η αφαίρεση. Σε πράξεις με τελεστές ίδιας προτεραιότητας, στην πρόσθεση/αφαίρεση, καθώς και στον πολλαπλασιασμό/διαίρεση οι πράξεις γίνονται από αριστερά προς τα δεξιά, ενώ στην ύψωση σε δύναμη οι πράξεις γίνονται από δεξιά προς τα αριστερά.

1.4.3. Μετατροπές τύπων

Ο τύπος δεδομένων μιας μεταβλητής μπορεί να αλλάζει αυτόματα εφόσον αυτό επιβάλλεται από τις πράξεις που γίνονται. Στο ακόλουθο παράδειγμα στο απόσπασμα κώδικα Α. 15 η μεταβλητή `x` σταδιακά αλλάζει τύπο δεδομένων από ακέραιο (`int`) σε πραγματικό (`float`) και στη συνέχεια σε μιγαδικό (`complex`). Αυτός ο τρόπος μετατροπής τύπου δεδομένων ονομάζεται υπονοούμενη μετατροπή (implicit data type conversion).

```
>>> x = 2
>>> type(x)
<class 'int'>
>>> x = x / 4
>>> x
0.5
>>> type(x)
<class 'float'>
>>> x = - x
>>> x = x**0.5
>>> x
(4.329780281177467e-17+0.7071067811865476j)
>>> type(x)
<class 'complex'>
```

Α. 15 – Υπονοούμενη μετατροπή τύπου δεδομένων.

Εκτός από την υπονοούμενη μετατροπή τύπου υπάρχει και η εξαναγκασμένη μετατροπή τύπου δεδομένων (explicit data type conversion). Αυτό συμβαίνει σε περιπτώσεις που μπορεί να είναι επιθυμητή η αλλαγή τύπου μιας μεταβλητής (π.χ. από ακέραια σε χαρακτήρες, από πραγματική σε ακέραια κ.α.). Η διαδικασία ονομάζεται type casting και ορισμένα παραδείγματα type casting παρουσιάζονται στο απόσπασμα κώδικα Α. 16.

```
>>> x = 10
>>> type(x)
<class 'int'>
>>> x = str(x)
>>> x, type(x)
('10', <class 'str'>)
>>> y = 10 / 3
>>> y, type(y)
(3.3333333333333335, <class 'float'>)
>>> y = int(y)
>>> y, type(y)
(3, <class 'int'>)
```

Α. 16 – Εξαναγκασμένη μετατροπή τύπου δεδομένων.

1.5. Είσοδος και έξοδος τιμών

Η είσοδος τιμών από τον χρήστη γίνεται με τη συνάρτηση `input()`. Η κλήση της συνάρτησης `input()` δημιουργεί μια κατάσταση αναμονής από τον χρήστη να εισάγει δεδομένα από το πληκτρολόγιο. Τα δεδομένα που εισάγει ο χρήστης επιστρέφονται ως λεκτικά, οπότε αν χρειάζεται

να χρησιμοποιηθούν στη συνέχεια ως ακέραιες τιμές ή ως πραγματικές τιμές θα πρέπει να γίνει η κατάλληλη μετατροπή τύπου, όπως στο παράδειγμα στο απόσπασμα κώδικα Α. 17.

```
>>> x = input()
7
>>> x, type(x)
('7', <class 'str'>)
>>> x = int(input())
7
>>> x, type(x)
(7, <class 'int'>)
```

Α. 17 – Είσοδος τιμών από τον χρήστη.

Η `input()` μπορεί να δέχεται ως όρισμα ένα λεκτικό που χρησιμοποιείται ως μήνυμα προς τον χρήστη, όπως για παράδειγμα στο απόσπασμα κώδικα Α. 18. Με αυτό τον τρόπο μπορεί να γραφεί κώδικας που καθοδηγεί τον χρήστη για τις τιμές που πρέπει να εισάγει όπως στη συνέχεια.

```
>>> x = float(input("Εισήγαγε μια πραγματική τιμή: "))
Εισήγαγε μια πραγματική τιμή: 3.14159
>>> x, type(x)
(3.14159, <class 'float'>)
```

Α. 18 – Είσοδος τιμής από τον χρήστη με μήνυμα προτροπής.

Από την άλλη μεριά, η έξοδος μπορεί να γίνει με τη συνάρτηση `print()` που χρησιμοποιείται για την εμφάνιση των τιμών των μεταβλητών, εκφράσεων και μηνυμάτων στην οθόνη. Η `print()` μπορεί να δέχεται πολλαπλά ορίσματα που εμφανίζονται το ένα δίπλα στο άλλο με ένα κενό χαρακτήρα ως διαχωριστικό. Επίσης, μπορούν να χρησιμοποιηθούν οι παράμετροι `sep` και `end` για να τροποποιήσουν τη συμπεριφορά της `print()`. Η `sep` καθορίζει το διαχωριστικό που θα εμφανίζεται ανάμεσα στις τιμές που εκτυπώνονται, ενώ η end καθορίζει τι θα χρησιμοποιηθεί στο τέλος της εκτύπωσης. Η προκαθορισμένη τιμή για την παράμετρο `sep` είναι ένα διάστημα (`' '`) ενώ για το `end` είναι η αλλαγή γραμμής (`'\n'`). Ένα παράδειγμα χρήσης της `print()` παρουσιάζεται στο απόσπασμα κώδικα Α. 19.

```
>>> a, b = 1, 2
>>> print(a, a +b, "message")
1 3 message
>>> print(a, a +b, "message", sep="-")
1-3-message
>>> print(a, a +b, "message", end="\n\n")
1 3 message

>>>
```

Α. 19 – Εκτύπωση με την print.

Συνηθίζεται μια συμβολοσειρά να προετοιμάζεται με κατάλληλη μορφοποίηση έτσι ώστε να περιέχει τιμές μεταβλητών και εκφράσεων σε προκαθορισμένες θέσεις πριν εμφανιστεί στην οθόνη με την `print()`. Η μορφοποίηση μπορεί να γίνει με διάφορους τρόπους όπως οι ακόλουθοι:

- Με το σύμβολο % που είναι ο παλαιότερος τρόπος (υπάρχει από την Python 2).
- Με τη μέθοδο συμβολοσειρών `format()`.
- Με f-strings που είναι νεότερος και βολικότερος τρόπος. Μια περιγραφή των δυνατοτήτων των f-strings βρίσκεται στο https://docs.python.org/3/tutorial/inputoutput.html.

Στο απόσπασμα κώδικα παρουσιάζεται ένα παράδειγμα που εμφανίζει το ίδιο αποτέλεσμα και με τους τρεις παραπάνω τρόπους.

```
>>> name, age = "Κατερίνα", 27
>>> s1 = "Το όνομα είναι %s και η ηλικία είναι %s" % (name, age)
>>> s2 = "Το όνομα είναι {} και η ηλικία είναι {}".format(name, age)
>>> s3 = f"Το όνομα είναι {name} και η ηλικία είναι {age}"
>>> s1
'Το όνομα είναι Κατερίνα και η ηλικία είναι 27'
>>> s2
'Το όνομα είναι Κατερίνα και η ηλικία είναι 27'
>>> s3
'Το όνομα είναι Κατερίνα και η ηλικία είναι 27'
```

Α. 20 – Μορφοποίηση λεκτικών.

1.6. Σχόλια

Τα σχόλια (comments) στον κώδικα, αγνοούνται από τον διερμηνευτή της Python και χρησιμοποιούνται για να παρέχουν εξήγηση σε σημεία του κώδικα που ο προγραμματιστής κρίνει ότι απαιτείται. Η έναρξη σχολίων στην Python γίνεται με το σύμβολο #. Μια καλή πρακτική είναι το σχόλιο να μην εξηγεί το τι κάνει ο κώδικας αλλά γιατί το κάνει. Επίσης, τα σχόλια θα πρέπει να είναι καθαρά διατυπωμένα, ενημερωμένα και σύντομα, ενώ θα πρέπει να αποφεύγονται περιττά σχόλια.

Τα σχόλια μπορούν να είναι inline, μιας γραμμής ή μπλοκ όπως φαίνεται στον κώδικα Κ. 1. Στην περίπτωση των σχολίων μπλοκ, είτε χρησιμοποιούνται πολλαπλά συνεχόμενα σχόλια μιας γραμμής, είτε μπορεί εναλλακτικά να χρησιμοποιηθεί ένα λεκτικό πολλαπλών γραμμών που τοποθετείται εντός διπλών ή απλών εισαγωγικών που επαναλαμβάνονται 3 φορές στην αρχή και στο τέλος του μπλοκ κειμένου.

```
foo() #  Σχόλιο inline

# Σχόλιο μιας γραμμής
bar()

# Σχόλιο τύπου μπλοκ
# Σχόλιο τύπου μπλοκ
buzz()

"""
```

```
Σχόλια τύπου block
Σχόλια τύπου block
"""
qux()
```
Κ. 1 – Σχόλια 3 τύπων.

1.7. Συμβολοσειρές

Συμβολοσειρά (string) είναι μια ακολουθία χαρακτήρων που οριοθετείται (αρχή και τέλος) με ειδικά σύμβολα. Για παράδειγμα μια συμβολοσειρά είναι το `'Hello World!'` που οριοθετείται με απλά εισαγωγικά. Στην Python ως σύμβολα οριοθέτησης συμβολοσειρών εκτός από τα απλά εισαγωγικά μπορούν να χρησιμοποιηθούν τα διπλά εισαγωγικά (π.χ. `"Hello World!"`), τα τρία απλά εισαγωγικά (π.χ. `'''Hello World!'''`), και τα τρία διπλά εισαγωγικά (π.χ. `"""Hello World!"""`). Τα τριπλά εισαγωγικά χρησιμοποιούνται για συμβολοσειρές που καταλαμβάνουν πολλές γραμμές. Μπορεί να χρησιμοποιηθεί συνδυασμός από απλά και διπλά εισαγωγικά έτσι ώστε να υπάρχουν μέσα στη συμβολοσειρά είτε απλά είτε διπλά εισαγωγικά, εφόσον απαιτείται. Οι συμβολοσειρές αναφέρονται συχνά και ως αλφαριθμητικά ή λεκτικά και από την Python 3 κάθε συμβολοσειρά αποθηκεύεται εσωτερικά ως Unicode που σημαίνει ότι μπορούν να χρησιμοποιηθούν χαρακτήρες σε ελληνικά και άλλες γλώσσες, σύμβολα μαθηματικών και άλλα σύμβολα καθώς και emojis. Στο απόσπασμα κώδικα Α. 21 παρουσιάζονται μερικά παραδείγματα δημιουργίας συμβολοσειρών.

```
>>> s1 = 'Ο κώδικας δουλεύει μέχρι να τον δείξεις σε άλλον.'
>>> s2 = "Ο κώδικας δουλεύει μέχρι να τον δείξεις σε άλλον."
>>> s3 = '''Ο κώδικας δουλεύει
... μέχρι να τον δείξεις
... σε άλλον.'''
>>> s4 = """Ο κώδικας δουλεύει
... μέχρι να τον δείξεις
... σε άλλον."""
>>> s5 = 'Ο κώδικας "δουλεύει" μέχρι να τον δείξεις σε άλλον.'
>>> s1, s2, s3, s4, s5
('Ο κώδικας δουλεύει μέχρι να τον δείξεις σε άλλον.', 'Ο κώδικας δουλεύει μέχρι να τον
δείξεις σε άλλον.', 'Ο κώδικας δουλεύει \nμέχρι να τον δείξεις \nσε άλλον.', 'Ο κώδικας
δουλεύει \nμέχρι να τον δείξεις \nσε άλλον.', 'Ο κώδικας "δουλεύει" μέχρι να τον δείξεις σε
άλλον.')
```
Α. 21 – Ορισμός συμβολοσειρών με απλά, διπλά και τριπλά εισαγωγικά.

1.7.1. Δεικτοδότηση συμβολοσειρών

Οι χαρακτήρες που περιέχονται σε μια συμβολοσειρά έχουν δείκτες με τους οποίους μπορούν να αναφερθούν. Η δεικτοδότηση (indexing) ξεκινά για τον πρώτο χαρακτήρα από τα αριστερά από το μηδέν και αυξάνεται κατά ένα για κάθε χαρακτήρα προς τα δεξιά. Ένα βολικό χαρακτηριστικό είναι ότι υπάρχει και εναλλακτική δεικτοδότηση από το τέλος προς την αρχή με αρνητικούς δείκτες, που

ξεκινά με τον πλέον δεξιό χαρακτήρα να έχει ως δείκτη το -1 και ο δείκτης μειώνεται κατά ένα για κάθε χαρακτήρα προς τα αριστερά. Ένα παράδειγμα της δεικτοδότησης συμβολοσειρών φαίνεται στην Εικόνα 8 για τη συμβολοσειρά `'Λειτουργεί στην Python μου.'` και στο απόσπασμα κώδικα Α. 22.

Λ	ε	ι	τ	ο	υ	ρ	γ	ε	ί		σ	τ	η	ν		P	y	t	h	o	n		μ	ο	υ	.
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Εικόνα 8 – Οι δείκτες από την αρχή προς το τέλος και από το τέλος προς την αρχή σε μια συμβολοσειρά.

```
>>> s = 'Λειτουργεί στην Python μου.'
>>> s[0], s[9], s[15], s[26], s[-1], s[-12], s[-18], s[-27]
('Λ', 'ί', ' ', '.', '.', ' ', 'ί', 'Λ')
```

Α. 22 – Προσπέλαση μεμονωμένων χαρακτήρων συμβολοσειράς.

1.7.2. Τεμαχισμός συμβολοσειρών

Ο τεμαχισμός (slicing) συμβολοσειρών επιστρέφει ένα τμήμα μιας συμβολοσειράς με βάση το μοτίβο `[<αρχή>:<τέλος>:<βήμα>]` για το οποίο ισχύει ότι:

- Οι `<αρχή>`, `<τέλος>`, `<βήμα>` πρέπει να είναι ακέραιοι.
- Αν παραλείπεται η `<αρχή>` υπονοείται η αρχή της συμβολοσειράς, δηλαδή η τιμή δείκτη `0`.
- Αν παραλείπεται το `<τέλος>` υπονοείται το τέλος της συμβολοσειράς, δηλαδή η τιμή `len(s)` για τη συμβολοσειρά `s`.
- Αν παραλείπεται το `<βήμα>` υπονοείται η τιμή `1`.

Ένα σημείο που αξίζει προσοχής είναι ότι το τμήμα `s[start:end]`, ξεκινά από το `start` αλλά δεν περιέχει το χαρακτήρα στη θέση end, σταματά δηλαδή στο `end - 1`, δηλαδή είναι συμπεριληπτικό (inclusive) του `start`, αλλά όχι (exclusive) του end.

Ορισμένα παραδείγματα τεμαχισμού συμβολοσειράς παρουσιάζονται στον Πίνακας 3.

Τεμαχισμός	Ερμηνεία	Παράδειγμα για τη συμβολοσειρά: s = 'Λειτουργεί στην Python μου.'
`s[start:end:step]`	Τμήμα της s από τον δείκτη `start` μέχρι και τον δείκτη `end - 1`	`s[16:21]` → `'Python'`
`s[start:]`	Τμήμα της s από τον δείκτη `start` μέχρι το τέλος	`s[16:]` → `'Python μου.'`
`s[:end]`	Τμήμα της s από την αρχή μέχρι και τον δείκτη `end - 1`	`s[:10]` → `'Λειτουργεί'`
`s[:]`	Όλη τη συμβολοσειρά	`s[:]` → `'Λειτουργεί στην Python μου.'`
`s[::-1]`	Λήψη αντίστροφης συμβολοσειράς	`s[::-1]` → `'.υομ nohtyP νητσ ίεγρυοτιεΛ'`

Πίνακας 3 – Τεμαχισμός συμβολοσειρών.

1.7.3. Η συνάρτηση `len()` και διάφορες μέθοδοι συμβολοσειρών

Η συνάρτηση `len()` επιστρέφει το μήκος της συμβολοσειράς που δέχεται ως όρισμα. Δεν είναι η μοναδική built-in συνάρτηση της Python που μπορεί να εφαρμοστεί σε συμβολοσειρά. Υπάρχουν και άλλες συναρτήσεις που δέχονται ως όρισμα μια συμβολοσειρά όπως η `list()`, η `tuple()` και η `set()` που δημιουργούν από μια συμβολοσειρά μια δομή δεδομένων λίστας, πλειάδας και συνόλου αντίστοιχα με περιεχόμενο τους χαρακτήρες της συμβολοσειράς, οι συναρτήσεις `int()` και `float()` που μετατρέπουν μια συμβολοσειρά σε ακέραιο και πραγματικό αντίστοιχα, και άλλες. Παραδείγματα των συναρτήσεων που αναφέρθηκαν παρουσιάζονται στο απόσπασμα κώδικα Α. 23.

```
>>> s = "H Python είναι απλή!"
>>> len(s)
20
>>> list(s)
['H', ' ', 'P', 'y', 't', 'h', 'o', 'n', ' ', 'ε', 'ί', 'ν', 'α', 'ι', ' ', 'α', 'π', 'λ',
'ή', '!']
>>> tuple(s)
('H', ' ', 'P', 'y', 't', 'h', 'o', 'n', ' ', 'ε', 'ί', 'ν', 'α', 'ι', ' ', 'α', 'π', 'λ',
'ή', '!')
>>> set(s)
{' ', 'ή', 'o', 'α', '!', 'π', 'H', 'ε', 'ν', 'λ', 't', 'n', 'P', 'h', 'ι', 'y', 'ί'}
>>> int("6174")
6174
>>> float("3.14")
3.14
```

Α. 23 – Συναρτήσεις συμβολοσειρών.

Εκτός από τις συναρτήσεις που δέχονται ως όρισμα μια συμβολοσειρά υπάρχει μεγάλος αριθμός από μεθόδους, δηλαδή συναρτήσεις που καλούνται με τον τελεστή τελείας μετά από ένα αντικείμενο συμβολοσειράς, που επιτελούν πολλές λειτουργίες πάνω σε αλφαριθμητικά. Ενδεικτικά εδώ αναφέρονται οι ακόλουθες μέθοδοι συμβολοσειρών:

- `upper()`
- `lower()`
- `find(str, start, end)`
- `count(str, start, end)`
- `index(value)`
- `replace(old, new, count)`

Παραδείγματα χρήσης των παραπάνω μεθόδων παρουσιάζονται στο απόσπασμα κώδικα Α. 24.

```
>>> s = "Python Programming"
>>> s.upper()
'PYTHON PROGRAMMING'
>>> s.lower()
'python programming'
>>> s.find("Pro")
7
>>> s.count("m")
```

```
2
>>> s.index("P")
0
>>> s.replace("Python", "Java")
'Java Programming'
```
Α. 24 – Παραδείγματα εφαρμογής μεθόδων σε λεκτικά.

Για την πλήρη λίστα των διαθέσιμων μεθόδων για συμβολοσειρές μπορεί κανείς να ανατρέξει στο https://docs.python.org/3/library/stdtypes.html#string-methods.

1.8. Συγκριτικοί τελεστές

Οι συγκριτικοί τελεστές της Python φαίνονται στον Πίνακας 4.

Συγκριτικός τελεστής	Ερμηνεία
`a == b`	Έλεγχος αν οι τιμές των μεταβλητών a και b είναι ίσες
`a != b`	Έλεγχος αν οι τιμές των μεταβλητών a και b είναι διαφορετικές
`a < b`	Έλεγχος αν η τιμή της a είναι μικρότερη από την τιμή της b
`a > b`	Έλεγχος αν η τιμή της a είναι μεγαλύτερη από την τιμή της b
`a <= b`	Έλεγχος αν η τιμή της a είναι μικρότερη ή ίση από την τιμή της b
`a >= b`	Έλεγχος αν η τιμή της a είναι μεγαλύτερη ή ίση από την τιμή της b
`a is b`	Έλεγχος αν η a και η b αναφέρονται στο ίδιο αντικείμενο
`a in b`	Έλεγχος αν η a υπάρχει στη b (π.χ. `'y' in 'Python'` επιστρέφει `True`)

Πίνακας 4 – Συγκριτικοί τελεστές.

Για τον τελεστή `is` αξίζει να αναφερθεί ότι η σύγκριση γίνεται με βάση την τιμή που επιστρέφει η κλήση της συνάρτησης `id()` για καθένα από τα a και b. Η συνάρτηση `id()` επιστρέφει έναν ακέραιο που αναγνωρίζει μοναδικά το αντικείμενο στο οποίο αναφέρεται το όρισμά της.

1.9. Λογικοί τελεστές

Οι λογικοί τελεστές που υποστηρίζει η Python παρουσιάζονται στον Πίνακας 5.

Λογικός τελεστής	Ερμηνεία	Πίνακας αληθείας

and	λογικό ΚΑΙ (σύζευξη)	`False and False → False`
		`False and True → False`
		`True and False → False`
		`True and True → True`

or	λογικό Ή (διάζευξη)	`False or False → False`
		`False or True → True`
		`True or False → True`
		`True or True → True`

not	άρνηση	`not False → True` `not True → False`

Πίνακας 5 – Λογικοί τελεστές.

1.9.1. Προτεραιότητα ανάμεσα σε αριθμητικούς, συγκριτικούς και λογικούς τελεστές

Η προτεραιότητα σε εκφράσεις με διάφορα είδη τελεστών είναι:

1. Αριθμητικοί τελεστές (υψηλότερη)
2. Συγκριτικοί τελεστές
3. Λογικοί τελεστές (χαμηλότερη)

Έτσι, στην έκφραση `1+2 > 3 or 10**2 == 5 * 20` η σειρά εκτέλεσης των πράξεων θα είναι:

- Πρώτα οι αριθμητικοί τελεστές: `10**2`, `5*20`, `1+2`,
- μετά οι συγκριτικοί τελεστές: `3 > 3`, `100 == 100`,
- και τέλος ο λογικός τελεστής: `False or True` που θα δώσει ως τελικό αποτέλεσμα `True`.

1.10. Τελεστές συμβολοσειρών

Στην Python υπάρχουν τελεστές που μπορούν να εφαρμοστούν σε συμβολοσειρές. Ο τελεστής `+` μπορεί να χρησιμοποιηθεί για τη συνένωση δύο συμβολοσειρών. Ο τελεστής `*` μπορεί να συνδυάσει από την μια πλευρά μια συμβολοσειρά και από την άλλη έναν ακέραιο και θα προκαλέσει επανάληψη της συμβολοσειράς τόσες φορές όσες ο ακέραιος. Οι τελεστές `in` και `not in` ελέγχουν αν μια συμβολοσειρά υπάρχει μέσα σε μια άλλη συμβολοσειρά. Επίσης, μπορούν να χρησιμοποιηθούν όλοι οι συγκριτικοί τελεστές (π.χ. `==`, `<`) για τη λεξικογραφική σύγκριση συμβολοσειρών. Μια ακόμα ομάδα τελεστών που μπορούν να χρησιμοποιηθούν με συμβολοσειρές είναι οι τελεστές επαυξημένης ανάθεσης `+=` και `*=` καθώς και ο τελεστής ανάθεσης `=`. Παραδείγματα χρήσης των τελεστών που αναφέρθηκαν παρουσιάζονται στο απόσπασμα κώδικα Α. 25.

```
>>> "Γεια σου " + "κόσμε!"
'Γεια σου κόσμε!'
>>> "Χα" * 3
'ΧαΧαΧα'
>>> "Py" in "Python"
True
>>> "Java" not in "Python"
True
>>> "ΑΑΒ" < "ΑΑΓ"   # λεξικογραφική σύγκριση
True
>>> s = "Τέλος"
>>> s += " παραδειγμάτων"
>>> s
'Τέλος παραδειγμάτων'
```

Α. 25 – Παραδείγματα με τελεστές που μπορούν να εφαρμοστούν σε συμβολοσειρές.

1.11. Τελεστές που εφαρμόζονται σε δυαδικά ψηφία

Μια ακόμα ομάδα τελεστών είναι οι λεγόμενοι bitwise τελεστές που εφαρμόζονται σε ακέραιους αριθμούς λαμβάνοντας υπόψη τη δυαδική τους μορφή. Οι τελεστές αυτοί παρουσιάζονται στον Πίνακας 6.

<table>
<tr><th>Τελεστής</th><th>Περιγραφή</th><th>Παράδειγμα (στο REPL)</th><th>Δυαδική αναπαράσταση</th></tr>
<tr><td>&</td><td>Δυαδικό ΚΑΙ – πρέπει και οι 2 δυαδικά ψηφία να είναι 1 για να είναι το αποτέλεσμα 1, αλλιώς το αποτέλεσμα είναι 0</td><td>>>> 5 & 3
1</td><td>101 ==> 5
011 ==> 3
---
001 ==> 1</td></tr>
<tr><td>|</td><td>Δυαδικό Ή – αρκεί ένα από τα 2 δυαδικά ψηφία να είναι 1 για να είναι το αποτέλεσμα 1, αλλιώς το αποτέλεσμα είναι 0</td><td>>>> 5 | 3
7</td><td>101 ==> 5
011 ==> 3
---
111 ==> 7</td></tr>
<tr><td>^</td><td>Αποκλειστικό Ή – πρέπει τα 2 δυαδικά ψηφία να είναι διαφορετικά μεταξύ τους για να είναι το αποτέλεσμα 1, αλλιώς το αποτέλεσμα είναι 0</td><td>>>> 5 ^ 3
6</td><td>101 ==> 5
011 ==> 3
---
110 ==> 6</td></tr>
<tr><td>~</td><td>Δυαδικό ΌΧΙ – είναι μονομελής τελεστής και πραγματοποιεί την πράξη του συμπληρώματος ως προς 2</td><td>>>> ~5
-6</td><td>00000101 ==> 5

11111010 ==> -6</td></tr>
<tr><td><<</td><td>Αριστερή ολίσθηση για ένα πλήθος θέσεων – η ολίσθηση κατά 1 ψηφίο προς τα αριστερά προκαλεί διπλασιασμό του αριθμού</td><td>>>> 5 << 2
20</td><td>00101 ==> 5

10100 ==> 20</td></tr>
<tr><td>>></td><td>Δεξιά ολίσθηση για ένα πλήθος θέσεων – η ολίσθηση κατά 1 ψηφίο προς τα δεξιά προκαλεί ακέραιο υποδιπλασιασμό του αριθμού</td><td>>>> 5 >> 1
2</td><td>101 ==> 5
---
010 ==> 2</td></tr>
</table>

Πίνακας 6 – Τελεστές δυαδικών ψηφίων.

1.12. Η εντολή επιλογής `if`

Η εντολή επιλογής `if` επιτρέπει την εκτέλεση ενός μπλοκ εντολών με βάση την αποτίμηση μιας συνθήκης. Μπλοκ εντολών είναι μια ή περισσότερες εντολές που βρίσκονται στο ίδιο επίπεδο εσοχής (indentation) ή αλλιώς κατακόρυφης στοίχισης και που εκτελούνται μαζί. Σε άλλες γλώσσες προγραμματισμού όπως για παράδειγμα στη C η αρχή και το τέλος των μπλοκ εντολών σηματοδοτείται από τις αγκύλες `{` και `}`. Ωστόσο, στην Python έχει γίνει η επιλογή της χρήσης εσοχών για καθαρότητα κώδικα και για έμμεση αποφυγή γραφής κώδικα με πολλαπλά

εμφωλευμένα μπλοκ που καθιστούν τον κώδικα δύσκολο στην κατανόηση και εκσφαλμάτωση. Η γενική σύνταξη της `if` στην Python είναι η ακόλουθη:

```
if <συνθήκη_1>:
  <μπλοκ_εντολών_1>
elif <συνθήκη_2>:
  <μπλοκ_εντολών_2>
elif <συνθήκη3>:
  <μπλοκ_εντολών_3>
...
else:
  <μπλοκ_εντολών_n>
```

Η εντολή `if` λειτουργεί ως εξής: Αν η `<συνθήκη1>` είναι αληθής τότε εκτελείται το `<μπλοκ εντολών 1>` και η εκτέλεση συνεχίζει με όποια εντολή βρίσκεται μετά την `if`. Αν η `<συνθήκη1>` είναι ψευδής τότε εξετάζεται η `<συνθήκη2>` και αν είναι αληθής εκτελούνται οι εντολές του `<μπλοκ εντολών 2>` και η εκτέλεση συνεχίζει μετά την `if`. Αν η `<συνθήκη2>` είναι ψευδής τότε εξετάζεται η `<συνθήκη3>` κ.ο.κ. Αν καμία από τις συνθήκες δεν είναι αληθής και υπάρχει το τμήμα `else` της εντολής `if`, τότε εκτελείται το `<μπλοκ εντολών n>` πριν η εκτέλεση συνεχιστεί με τις εντολές μετά την `if`. Συνεπώς, το πολύ ένα από τα μπλοκ εντολών της `if` θα εκτελεστεί, που θα είναι το πρώτο από την αρχή για το οποίο η αντίστοιχη συνθήκη θα είναι αληθής.

Παράδειγμα 1, με την εντολή `if`

Στη συνέχεια, στον κώδικα Κ. 2 παρουσιάζεται ένα παράδειγμα χρήσης της εντολής `if` σε ένα πρόγραμμα που δέχεται μια ακέραια τιμή από τον χρήστη και εμφανίζει το πρόσημο της τιμής.

```
x = input("Εισάγετε έναν αριθμό: ")
x = int(x)
if x > 0:
    print("Θετικός")
elif x < 0:
    print("Αρνητικός")
else:
    print("Μηδέν")
```

Κ. 2 – Ένα παράδειγμα με την εντολή if.

Ένα παράδειγμα εκτέλεσης του κώδικα Κ. 2 όπου ο χρήστης εισάγει την τιμή 57 παρουσιάζεται στην έξοδο Ε. 1 που ακολουθεί.

```
Εισάγετε έναν αριθμό: 57
Θετικός
```

Ε. 1 – Έξοδος εκτέλεσης προγράμματος.

Παράδειγμα 2, με την εντολή `if`

Ως ένα δεύτερο παράδειγμα παρουσιάζεται ένα πρόγραμμα που δέχεται δύο βαθμούς από τον χρήστη και υπολογίζει το μέσο όρο τους. Αν οι βαθμοί απέχουν πάνω από 2 μονάδες τότε ο μικρότερος βαθμός αλλάζει έτσι ώστε να απέχει 2 μονάδες από τον μεγαλύτερο βαθμό. Έτσι αν οι βαθμοί είναι 16 και 17 τότε ο μέσος όρος είναι (16+17) / 2 = 16.5, ενώ αν οι βαθμοί είναι 10 και 20, τότε ο βαθμός 10 γίνεται 18 και ο μέσος όρος (18 + 20) / 2 = 19.

Το πρόγραμμα που υλοποιεί την παραπάνω λογική παρουσιάζεται στον κώδικα Κ. 3, ενώ ένα παράδειγμα εκτέλεσης του προγράμματος φαίνεται στην έξοδο Ε. 2.

```
a = float(input("Δώσε τον πρώτο βαθμό: "))
b = float(input("Δώσε το δεύτερο βαθμό: "))
if a > b:
    a, b = b, a  # στο a βρίσκεται η μικρότερη τιμή και στο b η μεγαλύτερη τιμή

if b - a > 2:
    a = b - 2
    c = (a + b) / 2
    print(f"Ο προσαρμοσμένος μέσος όρος είναι {c}")
else:
    c = (a + b) / 2
    print(f"Ο μέσος όρος είναι {c}")
```
Κ. 3 – Προσαρμογή βαθμών και υπολογισμός μέσου όρου.

```
Δώσε τον πρώτο βαθμό: 10
Δώσε το δεύτερο βαθμό: 20
Ο προσαρμοσμένος μέσος όρος είναι 19.0
```
Ε. 2 – Παράδειγμα εκτέλεσης όπου χρειάστηκε να γίνει προσαρμογή βαθμού.

Μια εντολή `if` μπορεί να βρίσκεται εμφωλευμένη μέσα σε ένα μπλοκ μιας άλλης εντολής `if` προκειμένου να εξυπηρετήσει την αλγοριθμική λογική που απαιτείται. Στο ακόλουθο παράδειγμα παρουσιάζεται μια τέτοια περίπτωση.

Παράδειγμα 3, με την εντολή `if`

Στο πρόγραμμα που υλοποιείται στον κώδικα Κ. 4, ο χρήστης εισάγει την ηλικία που δηλώνει ότι έχει και το αν διαθέτει ταυτότητα (ΝΑΙ/ΌΧΙ) και εμφανίζεται μήνυμα σχετικά με το αν επιτρέπεται η είσοδός του σε μια εκδήλωση. Η είσοδος στην εκδήλωση επιτρέπεται σε κάποιο άτομο αν η ηλικία είναι από 18 και πάνω και αν διαθέτει ταυτότητα. Στον κώδικα, ο έλεγχος για το αν διαθέτει ταυτότητα είναι εμφωλευμένος μέσα στην περίπτωση που η συνθήκη `age >= 18` είναι αληθής.

```
age = int(input("Εισήγαγε ηλικία: "))
has_id_response = input("Έχει ταυτότητα (ΝΑΙ/ΌΧΙ); ")
```

```python
has_id = has_id_response.strip() in ["ΝΑΙ", "ναι", 'N', 'ν']

if age >= 18:
    if has_id:
        print("Επιτρέπεται η είσοδος.")
    else:
        print("Απαιτείται ταυτότητα.")
else:
    print("Ανήλικος - δεν επιτρέπεται η είσοδος.")
```
Κ. 4 – Έλεγχος ηλικίας.

Στην έξοδο Ε. 3 φαίνεται ένα παράδειγμα εκτέλεσης του κώδικα.

```
Εισήγαγε ηλικία: 19
Έχει ταυτότητα (ΝΑΙ/ΌΧΙ): ΝΑΙ
Επιτρέπεται η είσοδος.
```
Ε. 3 – Παράδειγμα εκτέλεσης.

1.12.1. Διπλές ανισότητες σε συνθήκες

Η Python υποστηρίζει έναν συντομότερο τρόπο γραφής διπλών ανισοτήτων, όπως χρησιμοποιούνται και στα μαθηματικά. Για παράδειγμα η συνθήκη που ελέγχει ότι μια θερμοκρασία είναι μεγαλύτερη από 18 αλλά μικρότερη ή ίση του 26 μπορεί να γραφεί ως `t > 18 and t <= 26` ή ισοδύναμα ως `18 < t <= 26`. Συνήθως χρησιμοποιείται ο δεύτερος τρόπος (διπλή ανισότητα), για συντομία.

1.12.2. Ο τριαδικός τελεστής στην Python

Ο τριαδικός τελεστής (ternary operator ή conditional expression) είναι ένα σύντομος τρόπος να γραφεί μια εντολή `if/else` σε μια γραμμή. Η σύνταξή του είναι η ακόλουθη:

`τιμή_αν_αληθής if συνθήκη else τιμή_αν_ψευδής`

Πρόκειται για έναν συνοπτικό τρόπο που καθιστά τον κώδικα ευανάγνωστο για απλές συνθήκες, ενώ για αποτύπωση συνθετότερης λογικής η χρήση του δεν συνίσταται. Για παράδειγμα στο ακόλουθο απόσπασμα κώδικα (Κ. 5) χρησιμοποιείται ο τριαδικός τελεστής για να αναθέσει στη μεταβλητή `status` την τιμή `"Ενήλικος"` ή `"Ανήλικος"` ανάλογα με την τιμή της μεταβλητής `age`.

```python
age = 20
status = "Ενήλικος" if age >= 18 else "Ανήλικος"
print(status)  # Ενήλικος
```
Κ. 5 – Εφαρμογή τριαδικού τελεστή για εύκολα κατανοητό κώδικα.

1.13. Η εντολή επιλογής match

Η εντολή match προστέθηκε στην Python 3.10, είναι γνωστή ως structural pattern matching (δομικό ταίριασμα προτύπου) και προσφέρει έναν καθαρό και εκφραστικό τρόπο επιλογής ροής εκτέλεσης, παρόμοιο με την εντολή switch που υπάρχει σε άλλες γλώσσες προγραμματισμού όπως η C. Η βασική ιδέα είναι ότι η τιμή που δίνεται στο match συγκρίνεται με μοτίβα που μπορούν να αντιστοιχούν σε ένα σύνολο τιμών και το πρόγραμμα εκτελεί το πρώτο case για οποίο μοτίβο ταιριάζει. Η βασική σύνταξη της εντολής match είναι:

```
match <έκφραση>:
    case <μοτίβο_1> [if <φρουρός_1>]:
        <μπλοκ_εντολών_1>
    case <μοτίβο_2> [if <φρουρός_2>]:
        <μπλοκ_εντολών_2>
    ...
    case _:
        <μπλοκ_εντολών_ν>
```

Η γενική μορφή της match περιλαμβάνει ένα match <έκφραση>: και διαδοχικά case <pattern>:, με προαιρετικές συνθήκες if, τα λεγόμενα guards (φρουρούς), ενώ το case _: λειτουργεί ως επιλογή catch-all, δηλαδή ενεργοποιείται στην περίπτωση που δεν ταιριάξει το μοτίβο σε καμία από τις προηγούμενες περιπτώσεις. Τα case(s) εξετάζονται στη σειρά από πάνω προς τα κάτω, εκτελείται μόνο το πρώτο που ταιριάζει και δεν υπάρχει fall-through, δηλαδή αν ταιριάξει ένα μοτίβο θα εκτελεστεί το μπλοκ εντολών του και στη συνέχεια η εκτέλεση θα συνεχίσει με τις εντολές μετά από την εντολή match. Τα if <guard> είναι προαιρετικά και με αυτά μπορεί να προστεθεί σε κάθε περίπτωση μια πρόσθετη συνθήκη που πρέπει να ισχύει.

Παράδειγμα με την εντολή match

Στο παράδειγμα του κώδικα Κ. 6 ορίζεται μια εντολή μετακίνησης ως command = "move", 5 και στη συνέχεια χρησιμοποιείται η δομή match για να αναλύσει τη μορφή και το περιεχόμενό της με βάση προκαθορισμένα μοτίβα. Κάθε case αποδομεί την εντολή μετακίνησης στο είδος της εντολής και στην παράμετρό της και ελέγχει με if guards πρόσθετες συνθήκες όπως αν ο αριθμός των βημάτων είναι θετικός ή αρνητικός ή αν η κατεύθυνση είναι έγκυρη. Η εκτέλεση σταματά στην πρώτη περίπτωση που ταιριάζει, ενώ το case _ λειτουργεί ως γενική περίπτωση για οποιαδήποτε μη αναγνωρίσιμη εντολή.

```
command = "move", 5
```

```
match command:
    case "move", steps if steps > 0:
        print(f"Μετακίνηση μπροστά κατά {steps} βήματα")
    case "move", steps if steps < 0:
        print(f"Μετακίνηση πίσω κατά {abs(steps)} βήματα")
    case "stop",:
        print("Στάση")
    case "turn", direction if direction in ("left", "right"):
        print(f"Στροφή προς {direction}")
    case _:
        print("Μη έγκυρη εντολή")
```
Κ. 6 – Παράδειγμα με την εντολή match.

1.13.1. Άσκηση 2, με την εντολή `if`

Γράψτε πρόγραμμα που να δέχεται από τον χρήστη έναν ακέραιο αριθμό και να εμφανίζει με κατάλληλο μήνυμα σε ποια από τις ακόλουθες περιπτώσεις βρίσκεται ο αριθμός: άρτιος και θετικός, άρτιος και αρνητικός, περιττός και θετικός, περιττός και αρνητικός, μηδέν.

Η λύση της άσκησης υπάρχει στο Παράρτημα Α'.

1.14. Η εντολή επανάληψης `while`

Η εντολή `while` είναι η μια από τις δύο εντολές επανάληψης που διαθέτει η Python (η άλλη είναι η εντολή `for`). Η σύνταξή της είναι η ακόλουθη:

```
while <συνθήκη>:
  <μπλοκ_εντολών_1>
[else:
  <μπλοκ_εντολών_2>]
```

Η εντολή `while` επιτρέπει την επαναληπτική εκτέλεση ενός μπλοκ κώδικα (το `<μπλοκ_εντολών_1>`) για όσο ισχύει μια συνθήκη. Όταν η συνθήκη γίνει ψευδής, τότε η εκτέλεση συνεχίζεται με τις εντολές μετά την `while`. Στην Python έχει προστεθεί στην `while` και το προαιρετικό τμήμα `else:`. Αν υπάρχει σε μια εντολή `while` το `else:` και το `<μπλοκ_εντολών_2>`, τότε αυτό θα εκτελεστεί αν η συνθήκη γίνει ψευδής και μόνο τότε, δηλαδή αν δεν γίνει έξοδος από την επανάληψη νωρίτερα με την εντολή `break` που θα αναφερθεί στη συνέχεια.

Παράδειγμα με την εντολή `while`

Το πρόγραμμα του κώδικα Κ. 7 δέχεται μια ακέραια τιμή και πραγματοποιεί αντίστροφη μέτρηση ξεκινώντας από την τιμή αυτή μέχρι να φτάσει στην τιμή 0. Στο τέλος εμφανίζει το μήνυμα «Τέλος εκτέλεσης».

```
x = int(input("Δώσε μια θετική ακέραια τιμή: "))
while x >= 0:
    print(x, end=" ")
    x -= 1
print("\nΤέλος εκτέλεσης")
```

Κ. 7 – Αντίστροφη μέτρηση.

Ένα παράδειγμα εξόδου για μια εκτέλεση του προγράμματος εμφανίζεται στην έξοδο Ε. 4.

```
Δώσε μια θετική ακέραια τιμή: 10
10 9 8 7 6 5 4 3 2 1 0
Τέλος εκτέλεσης
```

Ε. 4 – Παράδειγμα με την εντολή while.

1.15. Η εντολή επανάληψης for

Η εντολή for είναι η εντολή επανάληψης που χρησιμοποιείται συχνότερα στον προγραμματισμό με την Python. Η σύνταξή της είναι η ακόλουθη:

```
for <μεταβλητή> in <ακολουθία>:
    <μπλοκ_εντολών_1>
[else:
    <μπλοκ_εντολών_2>]
```

Η εντολή for επιτρέπει την επαναληπτική εκτέλεση του ίδιου μπλοκ κώδικα για κάθε στοιχείο μιας ακολουθίας, δηλαδή ενός iterable όπως για παράδειγμα μια συμβολοσειρά, μια λίστα κ.α. Γενικά, η for χρησιμοποιείται όταν το πλήθος των επαναλήψεων είναι γνωστό εκ των προτέρων. Το τμήμα else της εντολής λειτουργεί όπως και στη while, δηλαδή το <μπλοκ_εντολών_2> εκτελείται όταν η έξοδος από την επανάληψη προκληθεί λόγω εξάντλησης των στοιχείων της ακολουθίας και όχι αν προκληθεί πρόωρη έξοδος από την επανάληψη (με την break).

Παράδειγμα με την εντολή for

Το πρόγραμμα του κώδικα Κ. 8 εμφανίζει κάθε ψηφίο του αριθμού 1234567890, χωρισμένο το ένα από το άλλο με ένα κενό και μετά την εμφάνιση όλων των ψηφίων εμφανίζει το μήνυμα «Τέλος».

```
a = 1234567890
for x in str(a):
    print(x, end=" ")
```

```
print("\nΤέλος")
```
Κ. 8 – Παράδειγμα διάσχισης ψηφίων ακεραίου αριθμού με την εντολή for.

Η εκτέλεση του προγράμματος θα παράξει την έξοδο Ε. 5.

```
1 2 3 4 5 6 7 8 9 0
Τέλος
```
Ε. 5 – Έξοδος εκτέλεσης.

1.16. Οι εντολές break και continue

Δύο εντολές που συναντώνται συχνά στις εντολές επανάληψης while και for είναι η break και η continue. Η μεν εντολή break διακόπτει την εκτέλεση του βρόχου επανάληψης στον οποίο βρίσκεται και προκαλεί τη συνέχεια εκτέλεσης με την πρώτη εντολή μετά το βρόχο. Η δε εντολή continue διακόπτει την εκτέλεση της τρέχουσας επανάληψης του βρόχου επανάληψης στον οποίο βρίσκεται και προκαλεί τη συνέχεια εκτέλεσης της επόμενης επανάληψης του βρόχου. Η συμπεριφορά των δύο αυτών εντολών φαίνεται στον κώδικα Κ. 9 και στην έξοδό του Ε. 6. Στην περίπτωση του πρώτου βρόχου μόλις η μεταβλητή c λάβει την τιμή "3", εκτελείται η εντολή break και ο βρόχος τερματίζει, ενώ στην περίπτωση του επόμενου βρόχου μόλις η μεταβλητή c λάβει την τιμή "3", εκτελείται η εντολή continue και η συγκεκριμένη επανάληψη τερματίζεται πρόωρα και ο κώδικας συνεχίζει με την επόμενη επανάληψη, δηλαδή με το c να έχει την τιμή "4".

```
s = "12345"
print("Break")
for c in s:
    if c == "3":
        break
    print(c, end=" ")

print("\nContinue")
for c in s:
    if c == "3":
        continue
    print(c, end=" ")
```
Κ. 9 – Επίδειξη λειτουργίας των εντολών break και continue.

```
Break
1 2
Continue
1 2 4 5
```
Ε. 6 – Έξοδος κώδικα με break και continue.

1.17. Η συνάρτηση range()

Μια συνάρτηση που χρησιμοποιείται συχνά με την εντολή επανάληψης for είναι η range(). Η συνάρτηση range(start, end, step) επιτρέπει τη δημιουργία ακολουθιών ακεραίων αριθμών. Το όρισμα start συμπεριλαμβάνεται στην ακολουθία, ενώ αν παραληφθεί υπονοείται το 0. Το όρισμα end πρέπει υποχρεωτικά να συμπληρωθεί, και η τιμή του δεν συμπεριλαμβάνεται στην ακολουθία. Τέλος, το όρισμα step καθορίζει το βήμα της ακολουθίας (start, start + step, start + 2 * step, ...) και αν παραληφθεί υπονοείται το 1. Ένα σημαντικό χαρακτηριστικό της range() είναι ότι δεν επιστρέφει λίστα αλλά ένα διασχίσιμο (iterable) αντικείμενο το οποίο παράγει τιμές όταν αυτό απαιτηθεί, γεγονός που το καθιστά αποδοτικό από πλευράς δέσμευσης πόρων μνήμης. Μερικά παραδείγματα στο REPL με τη range() ακολουθούν στον απόσπασμα κώδικα Α. 26. Για την εμφάνιση του αποτελέσματος κάθε κλήσης της range() χρησιμοποιείται η μετατροπή του σε λίστα με το list().

```
>>> r1 = range(10)
>>> r2 = range(1,10)
>>> r3 = range(1,10,2)
>>> r4 = range(10,0,-1)
>>> for r in r1, r2, r3, r4:
...     print(list(r))
...
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 3, 5, 7, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Α. 26 – Παραδείγματα με τη συνάρτηση range().

1.18. Εμφωλευμένες επαναλήψεις

Οι εμφωλευμένες επαναλήψεις (nested loops) προκύπτουν όταν ένας βρόχος επανάληψης τοποθετείται μέσα στο σώμα ενός άλλου βρόχου και χρησιμοποιούνται για την επεξεργασία δομών δεδομένων με περισσότερες από μία διαστάσεις ή για την εκτέλεση υπολογισμών που απαιτούν κάποιας μορφής συνδυασμό τιμών. Στις εμφωλευμένες επαναλήψεις για κάθε επανάληψη του εξωτερικού βρόχου, ο εσωτερικός βρόχος εκτελείται πλήρως, γεγονός που αυξάνει σημαντικά τον συνολικό αριθμό εκτελέσεων και επηρεάζει άμεσα την υπολογιστική πολυπλοκότητα του προγράμματος. Στον κώδικα Κ. 10 παρουσιάζεται ένα πρόγραμμα που εκτυπώνει δύο φορές την προπαίδεια των αριθμών από 1 έως και 5, με εμφωλευμένες επαναλήψεις, δύο φορές, μια με εντολές for και μια με εντολές while.

```
for i in range(1, 6):
    for j in range(1, 6):
        print(f"{i * j:2}", end=" ")
    print()
```

```
print("-" * 14)

i = 1
while i <= 5:
    j = 1
    while j <= 5:
        print(f"{i * j:2}", end=" ")
        j += 1
    print()
    i += 1
```

Κ. 10 – Εμφωλευμένες επαναλήψεις με for και με while.

```
 1  2  3  4  5
 2  4  6  8 10
 3  6  9 12 15
 4  8 12 16 20
 5 10 15 20 25
--------------
 1  2  3  4  5
 2  4  6  8 10
 3  6  9 12 15
 4  8 12 16 20
 5 10 15 20 25
```

Ε. 7 – Έξοδος προπαίδειας από 1 μέχρι και 5.

1.18.1. Άσκηση 3, συνδυασμός εντολών επιλογής και επανάληψης

Γράψτε κώδικα που να επιλύει το πρόβλημα «μάντεψε τον αριθμό» όπου ζητείται ο εντοπισμός ενός τυχαίου ακέραιου αριθμού από το 1 μέχρι το 100 που επιλέγεται τυχαία από τον Η/Υ, με 6 το πολύ προσπάθειες του χρήστη. Για κάθε επιλογή του χρήστη θα λαμβάνει ανατροφοδότηση σχετικά με το αν η επιλογή του πέτυχε τη ζητούμενη τιμή ή αν είναι μικρότερη ή μεγαλύτερη. Αν ο χρήστης εντοπίσει τη ζητούμενη τιμή σε κάποια από τις 6 προσπάθειες θα ανακηρύσσεται νικητής, αλλιώς θα έχει χάσει το παιχνίδι.

Η λύση της άσκησης βρίσκεται στο Παράρτημα Α'.

1.19. Ερωτήσεις κλειστού τύπου

1. Τι αποτέλεσμα δίνει η έκφραση `"abc"[::-1]`;

   A. "abc"
   B. "bac"
   C. "cba"
   D. Σφάλμα

2. Τι αποτέλεσμα δίνει η έκφραση `10 // 3`;

A. `3.33`
B. `0.3`
C. `3`
D. Σφάλμα

3. Ποιο είναι το αποτέλεσμα του `list(range(3))`;
   A. `[1, 2, 3]`
   B. `[0, 1, 2]`
   C. `[0, 1, 2, 3]`
   D. `[3]`

4. Ποιο είναι το αποτέλεσμα του `"a" + "b" * 3`;
   A. `"aaab"`
   B. `"a3b"`
   C. `"a b b b"`
   D. `"abbb"`

5. Ποια είναι η έξοδος του `print("A" in "CAT")`;
   A. `True`
   B. `False`
   C. `None`
   D. `"A"`

1.20. Ασκήσεις προς επίλυση

1. Υπολογίστε μια προσέγγιση του αριθμού π = 3.14159..., κάνοντας την πράξη $\frac{1}{4}\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}\right)$, αναθέστε το αποτέλεσμα στη μεταβλητή `pi`. Ποια η διαφορά της τιμής που υπολογίσατε από την αποθηκευμένη σταθερά για το `pi` στο module `math`;

2. Γράψτε πρόγραμμα που να υπολογίζει το εμβαδόν και την περίμετρο ενός τριγώνου με δεδομένες τις πλευρές του a, b, c. Το πρόγραμμα θα ζητά από τον χρήστη να εισάγει τις τρεις πλευρές και θα ελέγχει αν μπορούν να σχηματίσουν τρίγωνο (κάθε πλευρά θα πρέπει να είναι μικρότερη από το άθροισμα των άλλων δύο). Αν σχηματίζουν τρίγωνο, που ισχύει όταν κάθε πλευρά έχει θετική τιμή και είναι μικρότερη από το άθροισμα των δύο άλλων πλευρών, να υπολογίζεται η περίμετρος και το εμβαδόν του τριγώνου. Αν οι τιμές που δόθηκαν δεν σχηματίζουν τρίγωνο, να εμφανίζεται κατάλληλο μήνυμα. Για το εμβαδόν E να χρησιμοποιηθεί ο τύπος του Ήρωνα:

$$E = \sqrt{p*(p-a)*(p-b)*(p-c)}$$

όπου p είναι η ημιπερίμετρος του τριγώνου, δηλαδή $p = \frac{a+b+c}{2}$

3. Γράψτε πρόγραμμα που να δέχεται από τον χρήστη ένα έτος και να εμφανίζει το εάν είναι δίσεκτο ή όχι. Δίσεκτο είναι ένα έτος που δεν διαιρείται με το 100, αλλά διαιρείται με το 4. Επίσης δίσεκτα είναι τα έτη που διαιρούνται με το 400.

4. Έστω μια τυπική σκακιέρα (8 x 8). Αν ξεκινήσουμε από την κάτω αριστερή θέση και διασχίσουμε τη σκακιέρα από αριστερά προς τα δεξιά και από κάτω προς τα πάνω τοποθετώντας 1 κόκκο ρυζιού στην πρώτη θέση και το διπλάσιο αριθμό κόκκων της προηγούμενης σε κάθε επόμενη θέση, να βρεθεί το πλήθος κόκκων ρυζιού που θα υπάρχουν συνολικά στην σκακιέρα. Αν ένα κιλό ρυζιού έχει περίπου 40.000 κόκκους, πόσα κιλά ρυζιού θα υπάρχουν στην σκακιέρα.

5. Γράψτε κώδικα που να επιλέγει τυχαία αριθμούς στο διάστημα 0 έως και 1 με τη συνάρτηση `random.uniform()` μέχρι το άθροισμά τους να γίνει μεγαλύτερο από 1. Επαναλάβατε το πείραμα 1.000.000 φορές. Κατά μέσο όρο πόσες φορές χρειάστηκε να επιλεγούν τιμές για να ξεπεράσει το άθροισμα την τιμή 1; Συγκρίνατε τον αριθμό αυτό με τη σταθερά `math.e`.

6. Γράψτε κώδικα που να υλοποιεί τη διαδικασία Kaprekar, δηλαδή, να δέχεται έναν τετραψήφιο αριθμό που να μην έχει όλα τα ψηφία ίδια, να διατάσσει τα ψηφία του σε αύξουσα σειρά και σε φθίνουσα σειρά, να αφαιρεί το μεγαλύτερο από το μικρότερο και να επαναλαμβάνει τη διαδικασία με το αποτέλεσμα, μέχρι να μην προκύπτει αλλαγή στο αποτέλεσμα. Ποιος είναι ο αριθμός με τον οποίο σταματά η επανάληψη;

ΚΕΦΑΛΑΙΟ 2: ΤΜΗΜΑΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ ΤΗΝ PYTHON

Από τα πρώτα βήματα του προγραμματισμού υπολογιστών έγινε φανερό ότι είναι πολύ χρήσιμη η τεχνική του χωρισμού ενός πολύπλοκου προβλήματος σε μικρότερα, αυτόνομα τμήματα. Με αυτό τον τρόπο αντί ενός μεγάλου και πολύπλοκου προβλήματος έχουν να λυθούν περισσότερα μικρότερα και απλούστερα προβλήματα.

Η τεχνική αυτή χρησιμοποιούνταν πολύ πριν τους υπολογιστές και τον προγραμματισμό σε πολλούς και διαφορετικούς τομείς και είναι γνωστή με το όνομα «διαίρει και κυρίευε» (divide and conquer, πιο γνωστό στην Ελλάδα σαν «διαίρει και βασίλευε»).

Στον προγραμματισμό αυτά τα τμήματα είναι ακολουθίες εντολών που εκτελούν μια αυτόνομη λειτουργία και ονομάζονται συναρτήσεις (functions)[2]. Μα αυτό τον τρόπο ένα μεγάλο πρόγραμμα μπορεί να χωριστεί σε πολλές μικρές συναρτήσεις που όταν εκτελεστούν με τη σειρά που έχουμε ορίσει, στο τέλος θα έχει λυθεί ολόκληρο το πρόβλημα.

Για παράδειγμα, αν υποτεθεί ότι πρέπει να γραφεί ένα πρόγραμμα για ένα ηλεκτρονικό κατάστημα που να υπολογίζει την τελική αξία μιας παραγγελίας. Μια πιθανή ανάλυση σε συναρτήσεις θα μπορούσε να είναι η ακόλουθη:

- Μια συνάρτηση που επιστρέφει την τιμή ενός προϊόντος χωρίς ΦΠΑ.
- Μια συνάρτηση που επιστρέφει το ποσοστό του ΦΠΑ, ανάλογα με την κατηγορία που ανήκει το προϊόν.
- Μια συνάρτηση που υπολογίζει την τιμή λιανικής, που συμπεριλαμβάνει τον ΦΠΑ.
- Μια συνάρτηση που επιστρέφει τη μείωση τιμής που πιθανόν δικαιούται η/ο πελάτης/ισσα λόγω ενεργής προσφοράς.
- Μια συνάρτηση που υπολογίζει το κόστος των μεταφορικών.
- Μια συνάρτηση που λαμβάνει υπόψη όλα τα παραπάνω και υπολογίζει το τελικό συνολικό κόστος της παραγγελίας.

Οι πρώτες τρεις συναρτήσεις θα χρειαστεί να εκτελεστούν μια φορά για κάθε προϊόν της παραγγελίας.

---

[2] Να σημειωθεί εδώ ότι ο αγγλικός όρος function έχει την έννοια της *συνάρτησης* στα μαθηματικά, αλλά και της *λειτουργίας* (που επιτελεί κάτι ή κάποιο άτομο) γενικότερα.

Αυτός ο τρόπος χωρισμού ενός προβλήματος σε συναρτήσεις για τα διάφορα τμήματα ονομάζεται τμηματικός προγραμματισμός (modular programming) και το πρόγραμμα τμηματοποιημένο πρόγραμμα.

2.1. Πλεονεκτήματα του τμηματικού προγραμματισμού

Ο χωρισμός ενός προγράμματος σε συναρτήσεις προσφέρει τα παρακάτω πλεονεκτήματα:

- **Απλούστερος κώδικας:** Ο χωρισμός ενός προγράμματος σε συναρτήσεις τείνει να κάνει τον κώδικα απλούστερο και πιο εύκολο στην κατανόηση. Πολλές μικρές συναρτήσεις είναι ευκολότερο να διαβαστούν και να γίνουν κατανοητές από μια μεγάλη ακολουθία εντολών.
- **Επαναχρησιμοποίηση κώδικα:** Οι συναρτήσεις βοηθούν στη μείωση της ύπαρξης πολλαπλών αντιγράφων του ίδιου κώδικα σε ένα πρόγραμμα. Μετατρέποντας σε συνάρτηση μια συγκεκριμένη λειτουργία που εκτελείται σε πολλά σημεία ενός προγράμματος, ο κώδικας της λειτουργίας υπάρχει μόνο μια φορά στον κώδικα, αλλά μπορεί εκτελεστεί όσες φορές χρειάζεται. Το να γράφεται ο κώδικας που εκτελεί μια λειτουργία μόνο μια φορά, αλλά να χρησιμοποιείται όποτε χρειάζεται ονομάζεται *επαναχρησιμοποίηση κώδικα*.
- **Καλύτερες δοκιμές ορθής λειτουργίας:** Η δοκιμή ορθής λειτουργίας και η εκσφαλμάτωση γίνονται ευκολότερα και πιο απλά όταν κάθε λειτουργία ενός προγράμματος υλοποιείται από μια συνάρτηση. Κάθε συνάρτηση μπορεί να ελεγχθεί, ώστε να προσδιοριστεί αν εκτελεί τη λειτουργία ορθά. Έτσι γίνεται ευκολότερη η απομόνωση και η διόρθωση πιθανών λαθών.
- **Γρηγορότερη ανάπτυξη κώδικα:** Ο χωρισμός σε αυτόνομες μονάδες και η υλοποίησή τους με συναρτήσεις βοηθάει πολύ όταν ένα άτομο ή μια ομάδα αναπτύσσουν πολλά προγράμματα. Σε μια τέτοια περίπτωση γίνεται φανερό ότι υπάρχουν αρκετές εργασίες που μπορεί να είναι κοινές σε πολλά προγράμματα, όπως το να εμφανίζουν ένα κείμενο βοήθειας για τη χρήση που προγράμματος, το να ζητάνε κάποια διαπιστευτήρια, να ελέγχουν κάποια δεδομένα που δέχονται σαν είσοδο, κλπ. Το να χρησιμοποιηθούν κάποιες συναρτήσεις που γράφτηκαν σε προηγούμενο χρόνο και εκτελούν την επιθυμητή λειτουργία είναι η λογική επιλογή. Δεν έχει ιδιαίτερο νόημα να γράφεται κώδικας για τις ίδιες λειτουργίες πολλές φορές.
- **Διευκόλυνση της ομαδικής εργασίας:** Στην περίπτωση ομάδας εργασίας, ο χωρισμός σε συναρτήσεις όπου η κάθε μια εκτελεί μια αυτόνομη λειτουργία επιτρέπει την ανάθεση του

γραψίματος διαφορετικών συναρτήσεων σε διαφορετικά άτομα. Αυτό επιτρέπει και την γρηγορότερη ανάπτυξη λόγω της παράλληλης εργασίας των μελών.

2.2. Κενές συναρτήσεις και συναρτήσεις που επιστρέφουν τιμή

Υπάρχουν δύο είδη συναρτήσεων: αυτές που επιστρέφουν κάποια τιμή και αυτές που δεν επιστρέφουν τιμή (κενές συναρτήσεις, void).

Όταν καλείται μια συνάρτηση που δεν επιστρέφει τιμή, απλά εκτελεί τις εντολές που την αποτελούν και τερματίζει.

Στην πραγματικότητα, *όλες* οι συναρτήσεις επιστρέφουν κάτι. Στην περίπτωση μιας κενής συνάρτησης, επιστρέφεται η τιμή None.

Όταν καλείται μια συνάρτηση που επιστρέφει τιμή, αυτή εκτελεί τις εντολές που την αποτελούν, αλλά όταν τερματίζει επιστρέφει μια τιμή στην εντολή που την κάλεσε. Παράδειγμα τέτοιας συνάρτησης που υπάρχει στην Python είναι η input, η οποία διαβάζει τα δεδομένα που δίνονται από το πληκτρολόγιο και τα επιστρέφει σαν μια συμβολοσειρά (string). Μια άλλη τέτοια συνάρτηση είναι η int, που επιστρέφει την τιμή του ορίσματος με το οποίο κλήθηκε σαν ακέραιο αριθμό.

2.3. Πώς ορίζεται και καλείται μια κενή συνάρτηση

Ο κώδικας που περιγράφει τις λειτουργίες που (θα) εκτελεί μια συνάρτηση ονομάζεται *ορισμός* της συνάρτησης. Όταν διαβάζει το πρόγραμμα ο διερμηνευτής (interpreter) της Python τις συναρτήσεις τις ελέγχει μόνο για την ύπαρξη συγκεκριμένων λαθών, χωρίς να εκτελείται ο κώδικάς τους. Ο κώδικάς τους εκτελείται μόνο όταν κατά την εκτέλεση του προγράμματος κάποια εντολή καλέσει τη συνάρτηση.

Σημείωση: υπάρχουν γλώσσες, όπως για παράδειγμα η Pascal, στις οποίες οι συναρτήσεις που δεν επιστρέφουν κάποια τιμή ονομάζονται *διεργασίες* (procedures) και είναι ξεχωριστές προγραμματιστικές οντότητες.

2.3.1. Ονόματα συναρτήσεων

Το πρώτο πράγμα που εμφανίζεται στον ορισμό μιας συνάρτησης είναι το όνομά της. Αυτό δίνει και τον τρόπο να κληθεί η συνάρτηση. Ένα καλό όνομα συνάρτησης συνήθως είναι αρκετά περιγραφικό, ώστε να δίνει μια γενική ιδέα της λειτουργίας που επιτελεί η συνάρτηση. Αυτό βέβαια δεν είναι υποχρεωτικό, αλλά είναι εξαιρετικά βοηθητικό για την κατανόηση ενός προγράμματος. Ακριβώς το ίδιο ισχύει, βέβαια, και για τα ονόματα μεταβλητών.

Τα ονόματα των συναρτήσεων μοιράζονται τους ίδιους κανόνες ονοματοδοσίας με τις μεταβλητές:

- Το όνομα της συνάρτησης δεν πρέπει να είναι ίδιο με το όνομα κάποιας λέξης-κλειδιού της Python.
- Δεν πρέπει να περιέχει κενά.
- Ο πρώτος χαρακτήρας πρέπει να είναι γράμμα του αγγλικού αλφάβητου (πεζό ή κεφαλαίο) ή ο χαρακτήρας της κάτω παύλας («_»).
- Οι υπόλοιποι χαρακτήρες του ονόματος μπορούν να περιέχουν εκτός από τους παραπάνω και τα ψηφία 0 έως 9.
- Οι πεζοί και οι κεφαλαίοι χαρακτήρες είναι διαφορετικοί.

Είναι πολύ συνηθισμένο τα ονόματα συναρτήσεων να περιέχουν κάποιο ρήμα ή κάποια ρηματική μορφή, ακριβώς επειδή οι συναρτήσεις κάνουν κάποια ενέργεια. Έτσι είναι αναμενόμενο μια συνάρτηση που ονομάζεται calculate_final_price να υπολογίζει την τελική τιμή από κάτι, ενώ ότι η print_date θα τυπώσει κάποια ημερομηνία.

2.4. Ορισμός και κλήση μιας συνάρτησης

Για να δημιουργηθεί μια συνάρτηση πρέπει να γραφεί ο ορισμός της. Ο ορισμός μιας συνάρτησης στην Python έχει την παρακάτω γενική μορφή:

```
def όνομα_συνάρτησης(παράμετρος1, παράμετρος2, …):
    εντολή1
    εντολή2
    εντολή3
    …
```

Η πρώτη γραμμή του ορισμού ονομάζεται *κεφαλίδα* της συνάρτησης και ορίζει την αρχή της συνάρτησης. Ξεκινάει με τη λεξη-κλειδί `def` και ακολουθείται από το όνομα που δίνεται στη συνάρτηση, ένα ζευγάρι παρενθέσεις και μια άνω και κάτω τελεία (colon). Αν η συνάρτηση έχει παραμέτρους που πρέπει να πάρουν τιμές για να εκτελέσει η συνάρτηση τη λειτουργία της, αυτές δηλώνονται μέσα στις παρενθέσεις χωρισμένες με κόμμα. (Οι παράμετροι θα αναλυθούν από το υποκεφάλαιο 2.10 και μετά.)

Από την επόμενη γραμμή ακολουθεί ένα τμήμα με εντολές (block), μια ομάδα εντολών που έχουν κοινό το ότι η εκτέλεσή τους (με τη συγκεκριμένη σειρά εκτέλεσης) υλοποιεί τη λειτουργία που είναι επιθυμητό να κάνει η συνάρτηση. Αυτές οι εντολές εκτελούνται κάθε φορά που εκτελείται η συνάρτηση. Συνήθως αποκαλούνται και κύριο σώμα της συνάρτησης.

Στο παραπάνω υπόδειγμα συνάρτησης, είναι φανερό ότι όλες οι εντολές του μπλοκ έχουν επιπλέον περιθώριο και είναι πιο μέσα από την πρώτη εντολή (κεφαλίδα). Αυτό είναι γενική απαίτηση στην Python: ο διερμηνευτής χρησιμοποιεί το επιπλέον περιθώριο για να ξεχωρίσει πού αρχίζει και πού τελειώνει ένα μπλοκ. Ταυτόχρονα, όμως, αυτή η απαίτηση κάνει τα προγράμματα που γράφονται σε Python πολύ πιο ευανάγνωστα για τον άνθρωπο και διευκολύνει την κατανόηση της λειτουργίας τους.

Στον κώδικα Κ. 11 εμφανίζεται ένα απλό παράδειγμα συνάρτησης**Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**.

```
def knights_message():
    print(`We are the knights who say Ni!`)
    print(`We are the keepers of the sacred words:`)
    print(Ni, Peng, and Neee-Wom.`)
```

Κ. 11 - Παράδειγμα συνάρτησης στην Python.

Όπως φαίνεται, η συνάρτηση ονομάζεται `knights_message` και αποτελείται από ένα μπλοκ τριών γραμμών που, όταν εκτελεστούν, θα τυπώσουν ένα μήνυμα.

Το παραπάνω παράδειγμα από μόνο του δεν μπορεί να κάνει κάτι, μια και αποτελεί μόνο τον ορισμό της συνάρτησης. Χρειάζεται και κάποιος κώδικας που θα την καλεί, ώστε να εκτελεστεί η συνάρτηση.

2.4.1. Η κλήση μιας συνάρτησης

Για να κληθεί η παραπάνω συνάρτηση από ένα πρόγραμμα, χρησιμοποιείται η παρακάτω εντολή:

```
knights_message()
```

Όταν καλείται μια συνάρτηση, ο διερμηνευτής μεταβαίνει στη συνάρτηση και εκτελεί τις εντολές που την αποτελούν. Όταν φτάσει στο τέλος του μπλοκ της συνάρτησης, ο διερμηνευτής επιστρέφει πίσω στο σημείο του προγράμματος από όπου κλήθηκε η συνάρτηση και συνεχίζει την εκτέλεση του προγράμματος από εκείνο το σημείο. Αυτό συνήθως αναφέρεται σαν «η συνάρτηση επιστρέφει».

Στον κώδικα Κ. 12 φαίνεται ένα σύντομο πρόγραμμα που καλεί τη συνάρτηση και στο Ε. 8 το αποτέλεσμα της κλήσης.

```python
# Παράδειγμα κενής (void) συνάρτησης.
# Ορισμός της συνάρτησης
def knights_message():
    print('We are the knights who say Ni!')
    print('We are the keepers of the sacred words:')
    print('Ni, Peng, and Neee-Wom.')

# Κλήση της συνάρτησης
knights_message()
```
Κ. 12 - Κενή (void) συνάρτηση και κλήση της.

```
We are the knights who say Ni!
We are the keepers of the sacred words:
Ni, Peng, and Neee-Wom.
```
Ε. 8 - Έξοδος από την εκτέλεση της συνάρτησης.

Όταν ο διερμηνευτής της Python ξεκινάει να διαβάζει και να αναλύει το πρόγραμμα, αγνοεί τα σχόλια των δύο πρώτων γραμμών.

Κατόπιν, διαβάζει την εντολή `def` της τρίτης γραμμής και δημιουργεί στη μνήμη μια συνάρτηση με το όνομα `knights_message`, η οποία περιέχει τις εντολές του μπλοκ των επόμενων τριών γραμμών. Σε αυτή τη φάση, η συνάρτηση απλά δημιουργείται, δεν εκτελείται.

Στη συνέχεια διαβάσει μια κενή γραμμή και ένα σχόλιο, τα οποία επίσης αγνοεί.

Τέλος, εκτελεί την εντολή στην τελευταία γραμμή, οπότε καλεί τη συνάρτηση. Η συνάρτηση `knights_message` εκτελείται και τυπώνονται οι τρεις γραμμές εξόδου που φαίνονται στο **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**.

Το παράδειγμα είναι ένα μικρό πρόγραμμα που περιέχει μόνο μια συνάρτηση. Συνήθως στα προγράμματα ορίζονται πολλές συναρτήσεις.

Δεν είναι ασυνήθιστο (αλλά κάθε άλλο παρά υποχρεωτικό) να υπάρχει μια συνάρτηση η οποία καλείται όταν ξεκινάει το πρόγραμμα και καλεί τις άλλες συναρτήσεις του προγράμματος όπως και όταν χρειάζεται, ακολουθώντας τις εντολές που περιέχονται στο κύριο σώμα της. Κατά κάποιον τρόπο περιέχει τη βασική λογική του προγράμματος και δίνει ένα σκελετό της όλης λογικής του.

Καθαρά για λόγους σύμβασης, πολλές φορές η συνάρτηση αυτή ονομάζεται main, αλλά τόσο η ύπαρξή της, όσο και το όνομά της με κανέναν τρόπο δεν είναι υποχρεωτικά[3] .

Στον κώδικα Κ. 13 φαίνεται ένα παράδειγμα με δύο συναρτήσεις, τη main και τη knights_message και στο Ε. 9 η έξοδος που παράγει.

```
# Πρόγραμμα με δύο συναρτήσεις
# Η συνάρτηση main
def main():
    print('Who are you?')
    knights_message()
    print('Those who hear them, seldom live to tell the tale.')

# Η συνάρτηση knights_message
def knights_message():
    print('We are the knights who say Ni!')
    print('We are the keepers of the sacred words:')
    print('Ni, Peng, and Neee-Wom.')

# Κλήση της συνάρτησης main
main()
```

Κ. 13 - Πρόγραμμα με δύο συναρτήσεις.

```
Who are you?
We are the knights who say Ni!
We are the keepers of the sacred words:
Ni, Peng, and Neee-Wom.
Those who hear them, seldom live to tell the tale.
```

Ε. 9 - Η έξοδος του προγράμματος.

Ο ορισμός της συνάρτησης main βρίσκεται αμέσως μετά τις δύο πρώτες γραμμές με τα σχόλια, ενώ ακολουθεί ο ορισμός της knights_message.

Η εντολή που καλεί τη συνάρτηση main είναι στην τελευταία γραμμή του προγράμματος. Η πρώτη εντολή στη main καλεί τη συνάρτηση print, που τυπώνει το αλφαριθμητικό «Who are you?». Κατόπιν καλεί τη συνάρτηση knights_message.

Αυτό κάνει τον διερμηνευτή να ξεκινήσει να εκτελεί τις εντολές της συνάρτησης knights_message.

---

[3] Αυτό προέρχεται από τη γλώσσα C, όπου πρέπει να υπάρχει μια συνάρτηση με το όνομα main (που παίζει το ρόλο αυτού που σε άλλες γλώσσες ονομάζεται «κύριο πρόγραμμα») και από την αρχή της ξεκινάει η εκτέλεση του προγράμματος.

Μόλις τις εκτελέσει, ο διερμηνευτής επιστρέφει στη συνάρτηση `main` και συνεχίζει με την εντολή που ακολουθεί την κλήση της συνάρτησης `knights_message`. Αυτή είναι η εντολή που τυπώνει το «Those who hear them, seldom live to tell the tale.», που είναι και η τελευταία εντολή της συνάρτησης `main`, οπότε τερματίζεται η εκτέλεση του προγράμματος.

Σημείωση: Μια συνηθισμένη έκφραση όταν ένα πρόγραμμα καλεί μια συνάρτηση, είναι ότι ο *έλεγχος* του προγράμματος περνάει σε αυτή τη συνάρτηση.

2.5. Τα αριστερά περιθώρια στην Python

Στην Python, οι γραμμές ενός μπλοκ πρέπει να έχουν μεγαλύτερο αριστερό περιθώριο (εσοχή) από το αμέσως εξωτερικό τους μπλοκ. Για τις συναρτήσεις, αυτό σημαίνει ότι οι εντολές που ακολουθούν την κεφαλίδα πρέπει να έχουν μεγαλύτερο αριστερό περιθώριο από την κεφαλίδα και ότι η τελευταία εντολή που έχει αυτό το μεγαλύτερο περιθώριο είναι και η τελευταία εντολή της συνάρτησης.

Υπάρχουν και κάποιοι επιπλέον κανόνες που πρέπει να ακολουθούνται. Κάθε γραμμή πρέπει να αρχίζει με τον ίδιο αριθμό κενών, διαφορετικά προκύπτουν λάθη. Σε κάποιες περιπτώσεις μπορεί ο διερμηνευτής να βγάλει μήνυμα λάθους (πράγμα που σημαίνει την ύπαρξη κάποιου συντακτικού λάθους), ενώ σε άλλες περιπτώσεις μπορεί να εκτελείται το πρόγραμμα (άρα είναι συντακτικά σωστό), αλλά δεν δίνει σωστά αποτελέσματα επειδή τα λάθος περιθώρια αλλοιώνουν τη λογική δομή του προγράμματος.

Σαν χαρακτήρες για τα περιθώρια είναι δεκτά τα κενά και οι χαρακτήρες στηλοθέτησης (tab). Σε κάθε πρόγραμμα καλό είναι να χρησιμοποιείται μόνο το ένα είδος χαρακτήρα, όχι και οι δύο, μια και μπορεί να προκληθούν λάθη στη λογική δομή του προγράμματος.

Πολλοί διορθωτές κειμένου και περιβάλλοντα ανάπτυξης που αναγνωρίζουν την Python εισάγουν αυτόματα τα περιθώρια. Αυτό το κάνει και ο διορθωτής IDLE. Η άνω και κάτω τελεία που βρίσκεται στο τέλος της κεφαλίδας του ορισμού μιας συνάρτησης βοηθάει να αναγνωρίσουν ότι η επόμενη γραμμή πρέπει να είναι πιο μέσα.

Στην Python το πιο συνηθισμένο μέγεθος εσοχής είναι τα τέσσερα κενά. Μπορεί να χρησιμοποιηθεί άλλος αριθμός, αρκεί να υπάρχει συνέπεια σε όλον τον κώδικα.

Επίσης, πρέπει να ληφθεί υπόψη ότι αν το πρόγραμμα που γράφεται θα δημοσιευθεί π.χ. σε κάποιο δημόσιο αποθετήριο όπως το GitHub ή το GitLab, καλό είναι να ακολουθεί τη συνήθη πρακτική του αποθετηρίου.

Αν γράφεται κάποιο πρόγραμμα συνεργατικά με άλλα άτομα, είναι ένα από τα πράγματα που πρέπει να συμφωνηθεί από πριν από όλα άτομα που αναπτύσσουν το πρόγραμμα.

Τέλος, γενικά εδώ και πολλά χρόνια οι χαρακτήρες tab δεν χρησιμοποιούνται γενικά για τα περιθώρια, ανεξαρτήτως γλώσσας προγραμματισμού για διάφορους λόγους. Στους περισσότερους διορθωτές κειμένου και τα περιβάλλοντα ανάπτυξης μπορεί να ρυθμιστεί κάθε φορά που πατιέται το πλήκτρο Tab να το αντικαθιστά στο κείμενο με ένα προκαθορισμένο αριθμό κενών.

2.6. Συμβολοσειρές τεκμηρίωσης (docstrings)

Στην Python είναι συχνή η χρήση κώδικα με τη μορφή βιβλιοθηκών ή modules, που είτε έχουν εγκατασταθεί μαζί με την Python, είτε εγκαθίστανται σε δεύτερο χρόνο, π.χ. με χρήση της εντολής `pip`.

Η κλήση της `help` με όρισμα το όνομα μιας συνάρτησης (ή μεθόδου) μιας βιβλιοθήκης εμφανίζει ένα κείμενο που περιγράφει τη λειτουργία και τον τρόπο χρήσης της συνάρτησης. Αυτό το κείμενο είναι μια *συμβολοσειρά τεκμηρίωσης* (docstring). Πρόκειται για μια συμβολοσειρά πολλών γραμμών (άρα περικλείεται από τριπλά εισαγωγικά) που τοποθετείται αμέσως μετά τη γραμμή ορισμού της συνάρτησης. Εκτός από την κλήση της `help`, το κείμενο εμφανίζεται και αν κληθεί η συνάρτηση με τη μορφή `<όνομα_συνάρτησης>.__doc__`.

Η χρήση των συμβολοσειρών τεκμηρίωσης δεν είναι υποχρεωτική, είναι απλά μια σύμβαση, αλλά χρησιμοποιούνται ευρύτατα και συνιστάται έντονα.

Το παράδειγμα Κ. 14 συνοψίζει τα παραπάνω:

```
def doc_function():
    """
    Αυτή η συνάρτηση απλά τυπώνει αυτή τη συμβολοσειρά τεκμηρίωσης.
    """
    print(doc_function.__doc__)

doc_function()
help(doc_function)
```
Κ. 14 - Συνάρτηση με docstring.

Εκτελώντας το παράγει την έξοδο Ε. 10:

```
Αυτή η συνάρτηση απλά τυπώνει αυτή τη συμβολοσειρά τεκμηρίωσης.


Help on function doc_function in module __main__:

```

```
doc_function()
    Αυτή η συνάρτηση απλά τυπώνει αυτή τη συμβολοσειρά τεκμηρίωσης.
```

Ε. 10 -Έξοδος προγράμματος με συνάρτηση με docstring.

2.7. Σχεδίαση προγραμμάτων ώστε να χρησιμοποιούν συναρτήσεις

Στον προγραμματισμό, για το χωρισμό ενός αλγόριθμου σε συναρτήσεις χρησιμοποιείται ευρύτατα η τεχνική σχεδίασης «από πάνω προς τα κάτω» (top-down design). Με αυτή την τεχνική το αρχικό πρόβλημα χωρίζεται σε μικρότερα σχετικά αυτόνομα τμήματα και κάθε τέτοιο αυτόνομο τμήμα χωρίζεται και αυτό σε μικρότερα αυτόνομα κομμάτια. Η διαδικασία συνεχίζεται μέχρι να φτάσει στο σημείο να υπάρχουν αυτόνομα τμήματα που μπορούν να υλοποιηθούν από μια σχετικά μικρή συνάρτηση.

Σε αυτή τη θεώρηση, η κορυφή («πάνω») είναι ολόκληρο το πρόβλημα/αλγόριθμος, ενώ κάθε βήμα ανάλυσης σε μικρότερα τμήματα πηγαίνει ένα επίπεδο προς τα «κάτω».

Η τεχνική αυτή ονομάζεται και «stepwise refinement» (κάπως δύσκολο να μεταφραστεί, βηματική/βαθμιαία εκλέπτυνση, ίσως), όνομα που έδωσε ο δημιουργός της Pascal, της Modula και Modula 2 και πολλών άλλων και εξαιρετικός επιστήμονας Niklaus Wirth.

Στα διαγράμματα ροής (flowcharts) οι συναρτήσεις έχουν το δικό τους σύμβολο, ένα ορθογώνιο με επιπλέον κάθετες γραμμές αριστερά και δεξιά, όπως φαίνεται στην Εικόνα 9.

Εικόνα 9 - Σύμβολο συνάρτησης στα διαγράμματα ροής

Αν στη σχεδίαση κάποιου προγράμματος χρησιμοποιούνται διαγραμματα ροής, σε κάποιο βολικό σημείο κοντά στο διάγραμμα ροής που περιέχει το ορθογώνιο της συνάρτησης πρέπει να υπάρχει ξεχωριστά και το διάγραμμα ροής της συνάρτησης.

2.7.1. Παύση της εκτέλεσης μέχρι να πατηθεί το πλήκτρο Enter

Υπάρχουν αρκετές περιπτώσεις που το πρόγραμμα χρειάζεται να κάνει παύση, ώστε η/ο χρήστρια/ης να προλάβει, για παράδειγμα, να διαβάσει κάποια πληροφορία που εμφανίζει το πρόγραμμα στην οθόνη. Όταν έχει διαβάσει την πληροφορία, πατάει το πλήκτρο Enter και το πρόγραμμα συνεχίζει. Στην Python αυτό μπορεί να γίνει κάνοντας χρήση της συνάρτησης `input`,

ώστε το πρόγραμμα να κάνει παύση μέχρι το πάτημα του Enter. Μια πιθανή εντολή τέτοιου τύπου θα μπορούσε να είναι η ακόλουθη:

```
input('Πατήστε το Enter για να συνεχίσετε.')
```

Με αυτή την εντολή εμφανίζεται στην οθόνη το κείμενο «Πατήστε το Enter για να συνεχίσετε.» και το πρόγραμμα κάνει παύση μέχρι να πατηθεί το πλήκτρο.

2.7.2. Η λέξη-κλειδί `pass` και η χρήση της

Όταν γράφεται ένα πρόγραμμα, ιδίως όταν ακολουθείται τη σχεδίαση top-down, πολλές φορές υπάρχουν σημεία όπου πρέπει να υπάρχει κάποιος κώδικας, ώστε να είναι σωστό συντακτικά το πρόγραμμα σε εκείνο το σημείο, αλλά ο κώδικας αυτός δεν έχει γραφεί ακόμα. Σε αυτές τις περιπτώσεις είναι χρήσιμη η εντολή `pass`, η οποία είναι μια έγκυρη λέξη-κλειδί της Python που δεν κάνει τίποτα όταν εκτελείται. Ακολουθούν κάποιες ενδεικτικές περιπτώσεις.

Σε κάποιο σημείο του προγράμματος μπορεί να είναι γνωστό από πριν ότι θα χρησιμοποιηθεί μια συνάρτηση, η οποία όμως δεν έχει γραφεί ακόμα. Το μόνο που υπάρχει είναι το πιθανό όνομά της. Σε τέτοιες περιπτώσεις μπορεί να χρησιμοποιηθεί η λέξη-κλειδί `pass`, ώστε η συνάρτηση να έχει μεν σώμα, αλλά να μην κάνει τίποτα. Σε μεταγενέστερο στάδιο μπορεί να αντικατασταθεί το `pass` με τον κανονικό κώδικα της συνάρτησης. Να μια τέτοια συνάρτηση:

```
def an_empty_function():
    pass
```

Η λέξη-κλειδί `pass` αγνοείται από τον διερμηνευτή, έτσι η συνάρτηση όταν κληθεί δεν κάνει τίποτα, απλά επιστρέφει.

Μια άλλη χρήση της είναι σε εντολές `if-else`, όπου μπορεί να είναι γνωστό ότι στο μέλλον θα χρειαστούν και τα δύο σκέλη, αλλά δεν υπάρχουν ακόμα οι εντολές που θα πάνε σε αυτό το μπλοκ κώδικα. Να ένα τέτοιο `if`, όπου και οι δύο επιλογές δεν κάνουν τίποτα, με σκοπό να αντικατασταθούν με το σωστό κώδικα στο μέλλον:

```
if a < b:
    pass
else:
    pass
```

Τέλος, ένας βρόχος επανάληψης `while` που χρησιμοποιεί την `pass`:

```
while a < 123:
    pass
```

2.8. Τοπικές μεταβλητές

Οι τοπικές μεταβλητές είναι μεταβλητές που δημιουργούνται μέσα σε μια συνάρτηση και δεν μπορούν να προσπελαστούν από εντολές που βρίσκονται έξω από τη συνάρτηση. Διαφορετικές συναρτήσεις μπορούν να έχουν τοπικές μεταβλητές με το ίδιο όνομα, επειδή οι συναρτήσεις δεν μπορούν να δουν τις μεταβλητές η μια της άλλης.

Κάθε φορά που γίνεται ανάθεση τιμής σε μεταβλητή μέσα σε μια συνάρτηση, δημιουργείται μια τοπική μεταβλητή. Η τοπική μεταβλητή ανήκει στη συνάρτηση την οποία δημιουργείται και μπορεί να προσπελαστεί μόνο από εντολές της συνάρτησης. Αυτό τονίζει η λέξη τοπική: η μεταβλητή μπορεί να χρησιμοποιηθεί μόνο εντός της συνάρτησης στην οποία έχει δημιουργηθεί.

Η προσπάθεια να προσπελαστεί τοπική μεταβλητή μιας συνάρτησης από άλλη συνάρτηση προκαλεί λάθος.

2.8.1. Εμβέλεια και τοπικές μεταβλητές

Με απλά λόγια, η εμβέλεια μιας μεταβλητής είναι το τμήμα του προγράμματος από το οποίο μπορεί αυτή να προσπελαστεί. Η μεταβλητή είναι ορατή μόνο σε εντολές εντός της εμβέλειάς της. Για μια τοπική μεταβλητή, η εμβέλεια είναι η συνάρτηση στην οποία δημιουργήθηκε. Ακριβέστερα, μπορεί να προσπελαστεί μόνο από κώδικα της συνάρτησης που βρίσκεται μετά το σημείο που δημιουργείται. Δεν μπορεί να γίνει προσπέλαση σε αυτήν από κώδικα της συνάρτησης ο οποίος προηγείται του σημείου δημιουργίας της τοπικής μεταβλητής.

Η παρακάτω συνάρτηση θα προκαλέσει λάθος. Για να διορθωθεί πρέπει η εντολή ανάθεσης στη μεταβλητή να μεταφερθεί πριν τη συνάρτηση `print`.

```
def bad_local_var_access():
    print(f’The value of loc_var is {loc_var}.’) # this causes error
    loc_var = 1
```

Στο πρόγραμμα Κ. 15 υπάρχουν δύο συναρτήσεις που έχουν και οι δύο μια τοπική μεταβλητή με το όνομα age.

```
# Πρόγραμμα με δύο συναρτήσεις με τοπικές μεταβλητές με το ίδιο όνομα.

def main():
    # κλήση της συνάρτησης John_Cleese
    John_Cleese()
```

```
    # τώρα κλήση της συνάρτησης Eric_Idle
    Eric_Idle()

# Ορισμός της συνάρτησης Eric_Idle με τοπική μεταβλητή με όνομα age.
def Eric_Idle():
    age = 82
    print(f'O Eric Idle είναι {age} χρονών.')

# Ορισμός της συνάρτησης John_Cleese με τοπική μεταβλητή με όνομα age.
def John_Cleese():
    age = 85
    print(f'O John Cleese είναι {age} χρονών.')

# Κλήση της συνάρτησης main
main()
```
Κ. 15 - Δύο συναρτήσεις με τοπική μεταβλητή με το ίδιο όνομα.

Το πρόγραμμα παράγει την έξοδο **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**.

```
O John Cleese είναι 85 χρονών.
O Eric Idle είναι 82 χρονών.
```
Ε. 11 - Έξοδος του προγράμματος.

Όπως φαίνεται στην Ε. 11, στο πρόγραμμα υπάρχουν δύο μεταβλητές με το όνομα age, αλλά κάθε στιγμή μόνο μία (το πολύ) από αυτές είναι ορατή. Αυτό γίνεται κατά το χρόνο που εκτελείται η συνάρτηση που ανήκει η αντίστοιχη μεταβλητή.

Στον Κ. 16 το προηγούμενο πρόγραμμα έχει τροποποιηθεί ελαφρά.

```python
# Πρόγραμμα με δύο συναρτήσεις με τοπική μεταβλητή με το ίδιο όνομα.

def main():
    # Κλήση της συνάρτησης Eric_Idle
    Eric_Idle()
    # Κλήση της συνάρτησης John_Cleese
    John_Cleese()

# Ορισμός της συνάρτησης Eric_Idle με τοπική μεταβλητή με όνομα age.
def Eric_Idle():
    age = 82
    print(f'O Eric Idle είναι {age} χρονών.')

# Ορισμός της συνάρτησης John_Cleese με τοπική μεταβλητή με όνομα age.
def John_Cleese():
    age = 85
   print(f'O John Cleese είναι {age} χρονών.')
    Eric_Idle()

# Κλήση της συνάρτησης main
main()
```

Κ. 16 - Το προηγούμενο πρόγραμμα ελαφρά αλλαγμένο.

```
O Eric Idle είναι 82 χρονών.
O John Cleese είναι 85 χρονών.
O Eric Idle είναι 82 χρονών.
```

Ε. 12 - Έξοδος του τροποποιημένου προγράμματος.

Είναι φανερό από την Ε. 12 ότι ακόμα και όταν καλείται η συνάρτηση `Eric_Idle` μέσα από τη συνάρτηση `John_Cleese`, στην έξοδο εμφανίζεται η ηλικία του `Eric Idle`, αφού τη στιγμή αυτή είναι ορατή η μεταβλητή age της συνάρτησης `Eric_Idle`.

2.8.2. Καθολικές μεταβλητές

Οι μεταβλητές που δηλώνονται σε σημεία του κώδικα που δεν ανήκουν στο σώμα κάποιας συνάρτησης ονομάζονται *καθολικές* (global). Μπορούν να προσπελαστούν για ανάγνωση από οποιοδήποτε σημείο του προγράμματος, συμπεριλαμβανομένου του εσωτερικού των συναρτήσεων.

Για να μπορεί όμως μια συνάρτηση να *τροποποιήσει* την τιμή μιας καθολικής μεταβλητής πρέπει να δηλωθεί η μεταβλητή μέσα στη συνάρτηση σαν global.

Αυτά φαίνεται στο παρακάτω σύντομο παράδειγμα Κ. 17.

```python
# Μια καθολική μεταβλητή
fpa = 24
```

```python
def retail_price(value):
    """
    Υπολογίζει την τιμή λιανικής πώλησης.
    """
    return value * (100 + fpa) / 100

def change_fpa_perc(new_perc):
    """
    Αλλάζει την κλίμακα του ΦΠΑ
    """
    global fpa
    fpa = new_perc

print(f"Τιμή λιανικής: {retail_price(100)}.") # Τιμή λιανικής: 124.0.
change_fpa_perc(13)
print(f"Νέα τιμή λιανικής: {retail_price(100)}.") # Νέα τιμή λίανικής: 113.0.
```
Κ. 17 - Πρόγραμμα με καθολική (global) μεταβλητή.

Στην έξοδο Ε. 13 φαίνεται η αλλαγή της τιμής της μεταβλητής global fpa και η αλλαγή στην τομή λιανικής που επιφέρει.

```
Τιμή λιανικής: 124.0.
Νέα τιμή λιανικής: 113.0.
```
Ε. 13 - Έξοδος του προγράμματος με καθολική μεταβλητή.

2.8.3. Συναρτήσεις μέσα σε συναρτήσεις και μη τοπικές μεταβλητές

Η Python είναι από τις γλώσσες που επιτρέπουν τον ορισμό συναρτήσεων μέσα σε άλλες συναρτήσεις (άλλη τέτοια γλώσσα είναι η Pascal, ενώ η C δεν το επιτρέπει). Αυτές οι συναρτήσεις ονομάζονται *εμφωλιασμένες* (nested) ή *εσωτερικές συναρτήσεις* (inner functions).

Μια συνηθισμένη χρήση τους έχει να κάνει με κώδικα που χρησιμοποιείται μόνο από τη συνάρτηση που τις περικλείει και δεν χρειάζεται ή δεν πρέπει να είναι ορατός έξω από αυτήν.

Ο κώδικας στο Κ. 18 είναι αυτός του προηγούμενου παραδείγματος, γραμμένος με τρόπο που να κάνει χρήση εσωτερικής συνάρτησης. Η έξοδος του προγράμματος παραμένει η ίδια.

```python
# Μια καθολική μεταβλητή
fpa = 24


def retail_price(value, tax_percent):
    """
    Υπολογίζει την τιμή λιανικής πώλησης.
    """
```

```
    def tax(value):
        """
        Υπολογίζει τον καθαρό φόρο.
        """
        return value * tax_percent / 100

    return value + tax(value)


print(f"Τιμή λιανικής: {retail_price(100, fpa)}.") # Τιμή λιανικής: 124.0.
```
Κ. 18 - Πρόγραμμα με εμφωλιασμένη συνάρτηση.

Η εσωτερική συνάρτηση μπορεί να προσπελάσει μια μεταβλητή της συνάρτησης που την περικλείει, αλλά μόνο για ανάγνωση. Αν πρέπει να μπορεί να της αλλάξει τιμή, τότε αυτή η μεταβλητή πρέπει να δηλωθεί στην εσωτερική συνάρτηση σαν nonlocal.

Είναι αντίστοιχη δήλωση με την global, που αναφέρθηκε στην παράγραφο 2.8.2, αλλά για άλλη εμβέλεια.

Στο Κ. 19 φαίνεται ένα απλό παράδειγμα συνάρτησης με μη τοπική μεταβλητή και στο Ε. 14 η έξοδος που παράγει.

```
def compute_total_cost(item_prices):
    total_cost = 0

    def add_partial_cost(prices):
        nonlocal total_cost
        for p in prices:
            total_cost += p
            print(f"Νέο μερικό σύνολο: {total_cost:.2f}.")

    item_prices.append(23.22)
    add_partial_cost(item_prices)

    print(f"Τελικό σύνολο: {total_cost:.2f}.")


costs = [24.99, 19.10, 45.55, 9.45]
compute_total_cost(costs)
```
Κ. 19 - Συνάρτηση με μη τοπική μεταβλητή.

```
Νέο μερικό σύνολο: 24.99.
Νέο μερικό σύνολο: 44.09.
Νέο μερικό σύνολο: 89.64.
Νέο μερικό σύνολο: 99.09.
Νέο μερικό σύνολο: 122.31.
```

```
Τελικό σύνολο: 122.31.
```
Ε. 14 - Έξοδος προγράμματος με μη τοπική μεταβλητή.

2.9. assert: έλεγχος για μη επιτρεπτές τιμές κατά την εκτέλεση κώδικα

Η `assert` είναι μια δεσμευμένη λέξη-κλειδί της Python η οποία ελέγχει μια συνθήκη και, αν είναι ψευδής, εγείρεται λάθος.

Κάποιες ενδεικτικές περιπτώσεις χρήσης της είναι όταν πριν εκτελεστεί ένα τμήμα κώδικα χρειάζεται να επιβεβαιωθεί ότι κάποιες μεταβλητές είναι του σωστού τύπου και/ή μέσα σε ένα διάστημα τιμών.

Στον κώδικα Κ. 20 η `assert` ελέγχει έναν αριθμό, ώστε να υπολογιστεί η τετραγωνική του ρίζα μόνο αν είναι θετικός ή μηδέν και στο Ε. 15 αποτελέσματα για μια σωστή και μια λάθος τιμή. Όπως φαίνεται, η ενεργοποίηση της `assert` προκαλεί λάθος και διακοπή της εκτέλεσης του προγράμματος.

```
from math import sqrt

def sq_root(a):
    assert a >= 0, "Δεν μπορεί να ληφθεί τετραγωνική ρίζα αρνητικού αριθμού"
    return sqrt(a)

print(sq_root(2))
print(sq_root(-99))
```
Κ. 20 - Χρήση της assert.

```
1.4142135623730951
Traceback (most recent call last):
  File "c:\XXX\YYY\ZZZ\Python_A\assert-test.py", line 8, in <module>
    print(sq_root(-99))
          ~~~~~~~^^^^^
  File "c:\XXX\YYY\ZZZ\Python_A\assert-test.py", line 4, in sq_root
    assert a >= 0, "Δεν μπορεί να ληφθεί τετραγωνική ρίζα αρνητικού αριθμού"
           ^^^^^^
AssertionError: Δεν μπορεί να ληφθεί τετραγωνική ρίζα αρνητικού αριθμού
```
Ε. 15 - Έξοδος του προγράμματος με χρήση της assert.

Η χρήση της μπορεί να συνδυαστεί με το block `try-except` που παρουσιάζεται στο υποκεφάλαιο **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**.

2.10. Περνώντας ορίσματα σε συναρτήσεις

Όρισμα είναι ένα συγκεκριμένο δεδομένο που περνιέται σε μια συνάρτηση όταν καλείται. *Παράμετρος* (για την ακρίβεια, *μεταβλητή παράμετρος*) είναι μια μεταβλητή συνάρτησης η οποία παίρνει την τιμή της από ένα όρισμα που περνιέται στη συνάρτηση.

Στη συντριπτική πλειοψηφία της χρήσης συναρτήσεων, χρειάζεται να τους δοθούν κάποια δεδομένα, ώστε να τα επεξεργαστούν. Αυτά τα δεδομένα που δίνονται σε μια συνάρτηση ονομάζονται *ορίσματα*. Η συνάρτηση τα χρησιμοποιεί στους υπολογισμούς ή τις λειτουργίες που εκτελεί.

Για να μπορέσει μια συνάρτηση να δεχτεί ορίσματα όταν καλείται, πρέπει να έχουν οριστεί μια ή περισσότερες *μεταβλητές παράμετροι*. Μια μεταβλητή παράμετρος, για συντομία παράμετρος, είναι μια ειδική μεταβλητή στη οποία ανατίθεται η τιμή ενός ορίσματος όταν καλείται η συνάρτηση.

Η παρακάτω είναι μια απλή συνάρτηση που έχει μια μεταβλητή παράμετρο.

```
def print_circumference(radius):
    pi1 = 3.14159265
    circumference = radius * 2 * pi1
    print(circumference)
```

Η συνάρτηση ονομάζεται `print_circumference`. Δέχεται σαν όρισμα έναν αριθμό και τυπώνει την περίμετρο του κύκλου που έχει ακτίνα ίση με το όρισμα που δόθηκε στη συνάρτηση.

Στην κεφαλίδα της συνάρτησης, μέσα στην παρένθεση υπάρχει η λέξη `radius`. Αυτό είναι το όνομα μιας μεταβλητής παραμέτρου. Σε αυτή τη μεταβλητή θα ανατεθεί η τιμή ενός ορίσματος όταν κληθεί η συνάρτηση. Στον κώδικα **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** εμφανίζεται ένα μικρό πρόγραμμα που καλεί τη συνάρτηση. Η έξοδος του προγράμματος είναι στο σχόλιο της τελευταίας γραμμής, που καλείται η `main`.

```
# Παράδειγμα συνάρτησης με παράμετρο.
from math import pi

def print_circumference(radius):
    """
Η συνάρτηση print_circumference δέχεται ένα όρισμα που δίνει την ακτίνα
ενός κύκλου και τυπώνει την περιφέρεια του κύκλου.
Ορίζει μια τοπική μεταβλητή με την τιμή του π με ακρίβεια 8 δεκαδικών.
```

```
    """
    circumference = radius * 2 * pi
    print(f"Κύκλος με ακτίνα {radius} έχει περιφέρεια {circumference}.")

# Κλήση της συνάρτησης main
number = 10
print_circumference(number) # Κύκλος με ακτίνα 10 έχει περιφέρεια 62.83185307179586.
```

Κ. 21 - Παράδειγμα συνάρτησης με μεταβλητή παράμετρο.

Όταν τρέξει το πρόγραμμα, πρώτα εισάγεται από τη βιβλιοθήκη math η τιμή του **π** σαν pi. Κατόπιν δημιουργείται η τοπική μεταβλητή number και της ανατίθεται η τιμή 10. Στην επόμενη γραμμή καλείται η συνάρτηση print_circumference:

```
print_circumference(number)
```

Η number εμφανίζεται μέσα στις παρενθέσεις. Αυτό σημαίνει ότι περνιέται σαν όρισμα στη συνάρτηση print_circumference. Με την εκτέλεση αυτής της εντολής καλείται η συνάρτηση print_circumference και στην παράμετρο radius ανατίθεται η *τιμή* της μεταβλητής number, στην προκειμένη περίπτωση το 10.

Στη συνάρτηση print_circumference ανατίθεται στην τοπική μεταβλητή circumference η τιμή της παράστασης radius * 2 * pi. Η radius έχει τιμή 10 και το αποτέλεσμα (68.831853) το εμφανίζει η επόμενη γραμμή.

2.10.1. Εμβέλεια μεταβλητών παραμέτρων

Όπως έχει αναφερθεί, οι τοπικές μεταβλητές έχουν εμβέλεια μέσα στη συνάρτηση που δημιουργούνται από το σημείο που δηλώνονται και μετά. Με τον όρο *δήλωση* εννοείται η πρώτη φορά που γίνεται ανάθεση τιμής στη μεταβλητή. Η εμβέλεια των μεταβλητών παραμέτρων καλύπτει ολόκληρη τη συνάρτηση και μπορούν να προσπελαστούν από όλες τις εντολές της συνάρτησης, αλλά όχι έξω από αυτήν. Στην ουσία, η αναφορά τους στην κεφαλίδα της συνάρτησης τις δημιουργεί πριν την πρώτη εντολή της συνάρτησης.

Αυτό φαίνεται στο επόμενο παράδειγμα:

Έστω ότι ένα άτομο ζητάει βοήθεια και οδηγίες για να πάει κάπου. Το άτομο αυτό προέρχεται από κάποια από τις χώρες που δεν έχουν υιοθετήσει το μετρικό σύστημα επειδή το θεωρούν πολύ δυσνόητο και δύσχρηστο. Την αναφορά κάποιας απόστασης σε χιλιόμετρα δεν μπορεί να την αντιληφθεί. Με τη χρήση του προγράμματος Κ. 22 μπορεί να του δοθεί η απόσταση σε μίλια.

```python
# Πρόγραμμα με συνάρτηση μιας παραμέτρου που μετατρέπει χιλιόμετρα
# σε μίλια (ξηράς).

def main():
    # Τυπώνει πληροφορίες για το πρόγραμμα
    show_help()
    # Διάβασμα του αριθμού των χιλιομέτρων.
    kilometers = float(input('Πόσα είναι τα χιλιόμετρα; '))
    # Μετατροπή σε μίλια.
    convert_kms_to_miles(kilometers)

# Εμφάνισε κάποιες πληροφορίες για το πρόγραμμα
def show_help():
    print(
    """
    Το πρόγραμμα μετατρέπει χιλιόμετρα (km) σε μίλια. Τα μίλια είναι μίλια
    ξηράς, που χρησιμοποιούνται στις ΗΠΑ, τη Βρετανία και μερικές άλλες μικρές
    χώρες και περιοχές.
    Ένα μίλι αντιστοιχεί ακριβώς σε 1.609344 km.
    """
    )

def convert_kms_to_miles(km):
    """
Η συνάρτηση convert_kms_to miles δέχεται τον αριθμό των χιλιομέτρων,
τα μετατρέπει σε μίλια και τυπώνει το αποτέλεσμα.
    """
    km_to_miles = 1.609344   # Σταθερά που μετατρέπει μίλια σε km
    # Χρησιμοποιείται το αντίστροφό της για τη μετατροπή km σε μίλια

    print(f"Χιλιόμετρα: {km=}.")
    print(f"Μίλια: {miles=}.")

    miles = km / km_to_miles
    # Εκτύπωση με ακρίβεια δύο δεκαδικών
    print(f'Τα {km} km είναι {miles:.2f} μίλια.')

# Κλήση της main
main()
```
Κ. 22 - Πρόγραμμα μετατροπής χιλιομέτρων σε μίλια.

Αν εκτελεστεί ο κώδικας όπως είναι, φτάνει μέχρι και την εκτύπωση της παραμέτρου km στη συνάρτηση convert_kms_to_miles, όπως φαίνεται στην έξοδο Ε. 16. Μόλις επιχειρήσει να προσπελάσει τη μεταβλητή miles για να τυπώσει την τιμή της, εγείρεται λάθος επειδή η μεταβλητή δεν υπάρχει ακόμα, αφού δεν της έχει αποδοθεί κάποια τιμή μέχρι εκείνο το σημείο.

```
    Το πρόγραμμα μετατρέπει χιλιόμετρα (km) σε μίλια. Τα μίλια είναι μίλια
    ξηράς, που χρησιμοποιούνται στις ΗΠΑ, τη Βρετανία και μερικές άλλες μικρές
    χώρες και περιοχές.
    Ένα μίλι αντιστοιχεί ακριβώς σε 1.609344 km.

Πόσα είναι τα χιλιόμετρα; 12
Χιλιόμετρα: km=12.0.
Traceback (most recent call last):
  File "c:\XXX\YYY\ZZZ\Python_A\ch_functions-km_to_miles.py", line 39, in <module>
    main()
    ~~~~^^
  File "c:\XXX\YYY\ZZZ\Python_A\ch_functions-km_to_miles.py", line 10, in main
    convert_kms_to_miles(kilometers)
    ~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^
  File "c:\XXX\YYY\ZZZ\Python_A\ch_functions-km_to_miles.py", line 31, in
convert_kms_to_miles
    print(f"Μίλια: {miles=}.")
                    ^^^^^
UnboundLocalError: cannot access local variable 'miles' where it is not associated with a
value
```

Ε. 16 - Εκτέλεση με προσπέλαση μεταβλητής πριν οριστεί.

Αν αφαιρεθούν ή σχολιαστούν οι δύο `print` που τυπώνουν τη `miles` και την km, το πρόγραμμα εκτελείται κανονικά και παράγει τη έξοδο Ε. 17.

```
    Το πρόγραμμα μετατρέπει χιλιόμετρα (km) σε μίλια. Τα μίλια είναι μίλια
    ξηράς, που χρησιμοποιούνται στις ΗΠΑ, τη Βρετανία και μερικές άλλες μικρές
    χώρες και περιοχές.
    Ένα μίλι αντιστοιχεί ακριβώς σε 1.609344 km.

Πόσα είναι τα χιλιόμετρα; 8
Τα 8.0 km είναι 4.97 μίλια.
```

Ε. 17 - Έξοδος προγράμματος μετατροπής χιλιομέτρων σε μίλια.

Η εκτέλεση δείχνει ότι τα οκτώ χιλιόμετρα είναι πολύ κοντά στα πέντε μίλια, μια πληροφορία που θα είναι κατανοητή στο άτομο που ρώτησε.

2.10.2. Μια σύντομη παρένθεση: έλεγχος της εισόδου

Πολλές φορές όταν εκτελείται ένα πρόγραμμα δέχεται δεδομένα από πηγές που δεν είναι βέβαιο ότι τα δεδομένα που προσφέρουν είναι έγκυρα. Τέτοια πηγή μπορεί να είναι για παράδειγμα η

είσοδος από τη/ον χρήστρια/τη μέσω της `input`, ή η ανάγνωση ενός αρχείου δεδομένων (παρουσιάζεται στο **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**).

Σε τέτοιες περιπτώσεις είναι χρήσιμο, ή και απαραίτητο, να εξασφαλιστεί ότι τα δεδομένα είναι έγκυρα πριν περαστούν σε μια συνάρτηση για επεξεργασία. Η συνάρτηση μπορεί να περιμένει έγκυρα δεδομένα και το πέρασμα προβληματικών δεδομένων μπορεί να προκαλέσει σφάλμα ή εξαίρεση.

Για παράδειγμα, αν η είσοδος πρέπει να είναι ακέραιος, τότε θα πρέπει να περιέχει μόνο ψηφία, ίσως και ένα πρόσημο. Αν είναι κάποιο όνομα ή επώνυμο, θα πρέπει να περιέχει μόνο γράμματα και τουλάχιστον το πρώτο γράμμα να είναι κεφαλαίο. Αν πρέπει να είναι ένας δεκαδικός, εκτός από τα ψηφία και το πρόσημο, μπορεί να έχει και μια υποδιαστολή.

Τέτοιου είδους έλεγχοι, που μπορεί να είναι πιο εξειδικευμένοι αν είναι γνωστές κάποιες ιδιαιτερότητες των δεδομένων, βοηθούν στην αποφυγή πρόωρου τερματισμού του προγράμματος και επιτρέπουν να δοθεί η δυνατότητα στα άτομα που το χρησιμοποιούν να δώσουν μια σωστή είσοδο, π.χ. με χρήση ενός βρόχου επανάληψης.

2.10.3. Άσκηση 1

Το ίδιο άτομο ρωτάει και για τον καιρό των επομένων ημερών, ιδίως για τις θερμοκρασίες που θα επικρατήσουν, ώστε να ξέρει τι ρούχα θα χρειαστεί.

Στις χώρες που αναφέρθηκαν, δεν χρησιμοποιούν την κλίμακα θερμοκρασιών Κελσίου, αλλά την κλίμακα Φαρενάιτ. Για να μπορέσει να του δοθεί αυτή η πληροφορία, πρέπει να μετατραπούν οι βαθμοί Κελσίου σε βαθμούς Φαρενάιτ.

Για τη μετατροπή των βαθμών Φαρενάιτ σε Κελσίου, χρησιμοποιείται ο παρακάτω τύπος:

ºC = (ºF - 32) * 5 / 9

ή, ισοδύναμα:

ºC = (ºF - 32) / 1.8

Αντίστροφα, για να μετατραπούν οι βαθμοί Κελσίου σε βαθμούς Φαρενάιτ χρησιμοποιείται ο παρακάτω τύπος:

ºF = (ºC * 9 / 5) + 32

ή, ισοδύναμα:

ºF = ºC * 1.8 + 32

Γράψτε ένα πρόγραμμα σε Python που να ζητάει σαν είσοδο μια θερμοκρασία σε βαθμούς Κελσίου και να τυπώνει την αντίστοιχη θερμοκρασία σε βαθμούς Φαρενάιτ. Για τη μετατροπή να χρησιμοποιήσετε μια συνάρτηση που θα δέχεται ένα όρισμα, τη θερμοκρασία σε βαθμούς Κελσίου, θα τη μετατρέπει σε βαθμούς Φαρενάιτ και θα τυπώνει το αποτέλεσμα. Προσθέστε ένα docstring που να περιγράφει τη λειτουργία της συνάρτησης. Αν στην είσοδο δεν δοθεί κάποια τιμή, αλλά πατηθεί π.χ. απλά το πλήκτρο «Enter», να τυπώνεται ένα βοηθητικό κείμενο (θα μπορούσε να είναι το docstring της συνάρτησης).

Πριν περαστεί η είσοδος στη συνάρτηση για μετατροπή, να ελέγχετε αν είναι πράγματι δεκαδικός αριθμός της μορφής 3.14 ή 3,14 (μόνο ψηφία και μία υποδιαστολή, είτε τελεία είτε κόμμα) και προαιρετικά με ένα πρόσημο πριν τα ψηφία. Γράψτε και μια συνάρτηση που να επιστρέφει True αν η είσοδος είναι έγκυρη, αλλιώς False.

Η μετατροπή σε βαθμούς Φαρενάιτ να γίνεται μόνο για έγκυρες εισόδους. Αν δεν είναι έγκυρη η είσοδος, να ζητείται νέα.

Η λύση της άσκησης υπάρχει στο Παράρτημα Α’.

2.10.4. Περνώντας πολλά ορίσματα

Πολλές φορές οι συναρτήσεις χρειάζονται περισσότερες από μία πληροφορίες για να μπορέσουν να κάνουν τη λειτουργία τους.

Αν χρειάζεται μια συνάρτηση που να υπολογίζει την περίμετρο ενός παραλληλόγραμμου, χρειάζονται τα μήκη των δύο πλευρών του.

Αν χρειάζεται να βρεθεί αν περιέχεται ένα μικρό αλφαριθμητικό μέσα σε ένα μεγαλύτερο, πρέπει να περαστούν στη συνάρτηση αυτά τα δύο αλφαριθμητικά.

Το πρόγραμμα στο Κ. 23 υπολογίζει την περίμετρο ενός παραλληλόγραμμου. Η έξοδος παρουσιάζεται στο Ε. 18.

```python
# Πρόγραμμα που περιέχει συνάρτηση που υπολογίζει την περίμετρο
# παραλληλογράμμου.

# Η συνάρτηση print_perimeter υπολογίζει το συνολικό μήκος της περιμέτρου
# του παραλληλογράμμου.
def print_perimeter(width, height):
    perimeter = 2 * (width + height)
    print(f"Η περίμετρος είναι {perimeter}.")
```

```
width_input = int(input('Πλάτος παραλληλόγραμμου: '))
height_input = int(input('Ύψος παραλληλόγραμμου: '))
print_perimeter(width_input, height_input)
```
Κ. 23 - Συνάρτηση με δύο παραμέτρους, που υπολογίζει την περίμετρο ενός παραλληλόγραμμου.

```
Πλάτος παραλληλόγραμμου: 12
Ύψος παραλληλόγραμμου: 6
Η περίμετρος είναι 36.
```
Ε. 18 - Έξοδος υπολογισμού περιμέτρου παραλληλογράμμου.

Όπως φαίνεται, τα ονόματα των δύο μεταβλητών παραμέτρων βρίσκονται μέσα στις παρενθέσεις της κεφαλίδας της συνάρτησης και χωρίζονται μεταξύ τους με κόμμα. Αυτό ονομάζεται *λίστα παραμέτρων*.

Η τελευταία εντολή του προγράμματος καλεί τη συνάρτηση `print_perimeter` και της περνάει δύο ορίσματα: το `width_input` και το `height_input`. Τα ορίσματα περνιούνται με θέση στις αντίστοιχες μεταβλητές παραμέτρους στη συνάρτηση: το πρώτο όρισμα αντιστοιχεί και περνιέται στην πρώτη παράμετρο και το δεύτερο όρισμα στη δεύτερη. Οπότε, η τιμή της μεταβλητής `width_input` ανατίθεται στην παράμετρο `width` και η τιμή της μεταβλητής `height_input` ανατίθεται στην `height`.

Ας αντιστραφεί η σειρά των ορισμάτων στην κλήση της συνάρτησης:

```
print_perimeter(height_input, width_input)
```

Σε αυτή την περίπτωση η τιμή της μεταβλητής `height_input` ανατίθεται στην παράμετρο `width` και η τιμή της μεταβλητής `width_input` ανατίθεται στην `height`.

Στη συγκεκριμένη συνάρτηση αυτό δεν έχει επίπτωση στο αποτέλεσμα επειδή και οι δύο τιμές προστίθενται και μετά πολλαπλασιάζονται με το 2, αλλά στις περισσότερες περιπτώσεις η αλλαγή στη διάταξη επιφέρει και διαφορετικό, πιθανότατα λανθασμένο αποτέλεσμα, ακόμα και λάθος στην εκτέλεση, αν π.χ. διαφέρουν οι τύποι του ορίσματος και της παραμέτρου. Αν, για παράδειγμα, στη συνάρτηση αφαιρούνταν η μία παράμετρος από την άλλη, το αποτέλεσμα θα ήταν διαφορετικό.

Ο κώδικας στο Κ. 24 είναι ένα τέτοιο σύντομο παράδειγμα (τα αποτελέσματα της εκτέλεσης είναι στα σχόλια των δύο τελευταίων γραμμών):

```
# Πρόγραμμα που δείχνει ότι η σειρά των ορισμάτων έχει σημασία.

def subtraction(a, b):
    print(f"{a} - {b} = {a - b}.")
```

```
arg1 = 42
arg2 = 24
subtraction(arg1, arg2)  # 42 - 24 = 18.
subtraction(arg2, arg1)  # 24 - 42 = -18.
```
Κ. 24 - Κλήση συνάρτησης με τα ορίσματα σε διαφορετική σειρά.

Όπως φαίνεται, αντιστρέφοντας τη σειρά των ορισμάτων στην κλήση της συνάρτησης `subtraction`, αλλάζει και το αποτέλεσμα που τυπώνει η συνάρτηση. Για τη συνάρτηση δεν αλλάζει κάτι, πάντα αφαιρεί τη δεύτερη παράμετρο από την πρώτη.

2.10.5. Όταν οι συναρτήσεις επιστρέφουν τιμές

Μέχρι τώρα, για λόγους απλότητας οι συναρτήσεις που παρουσιάστηκαν δεν επιστρέφουν κάποια τιμή με το τέλος της εκτέλεσής τους. Η πιο συνηθισμένη περίπτωση είναι οι συναρτήσεις να επιστρέφουν κάτι που να μπορεί να χρησιμοποιήσει το τμήμα του προγράμματος που τις καλεί. Αυτό μπορεί να είναι μια απλή τιμή, όπως ένας αριθμός ή μια συμβολοσειρά, αλλά μπορεί να είναι, για παράδειγμα, μια λίστα, ένα λεξικό, ή ένα αντικείμενο (τα οποία αναλύονται σε επόμενα κεφάλαια: 3**Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**, 4 και 7).

Στον κώδικα Κ. 25 η συνάρτηση `discount()` υπολογίζει την τιμή ενός προϊόντος μετά την έκπτωση και την επιστρέφει. Το αποτέλεσμα φαίνεται στην Ε. 19.

```
# Πρόγραμμα με συνάρτηση που επιστρέφει την έκπτωση σε μια τιμή.

def discount(price, discount):
    """
    Η συνάρτηση υπολογίζει και επιστρέφει την τιμή μετά την έκπτωση.
    Η έκπτωση δίνεται σαν ποσοστό.
    """
    return price * (100 - discount) / 100

orig_price = 29
discount_perc = 25
disc_price = discount(orig_price, discount_perc)
print(f"Η τιμή {orig_price} με έκπτωση {discount_perc}% γίνεται {disc_price}.")
```
Κ. 25 - Συνάρτηση που επιστρέφει τιμή.

```
Η τιμή 29 με έκπτωση 25% γίνεται 21.75.
```
Ε. 19 - Η έξοδος του κώδικα με τη συνάρτηση που επιστρέφει την τιμή μετά την έκπτωση.

2.10.6. Επιφέροντας αλλαγές στις παραμέτρους

Όταν περνάει ένα όρισμα σε μια συνάρτηση στην Python, στην μεταβλητή παράμετρο της συνάρτησης περνάει μόνο η *τιμή* του ορίσματος και όχι το ίδιο το όρισμα. Μπορεί να ειπωθεί απλά ότι η τιμή του ορίσματος αντιγράφεται στην παράμετρο.

Αυτό σημαίνει ότι τυχόν αλλαγές που γίνονται στην τιμή της παραμέτρου μέσα στη συνάρτηση δεν επηρεάζουν το αρχικό όρισμα.

```
# Πρόγραμμα που δείχνει ότι η αλλαγή στις τιμές των παραμέτρων μέσα στις
# συναρτήσεις δεν επηρεάζουν το αντίστοιχο όρισμα.

def change_name(a_var):
    a_var = 'Paul'
    print(f"Μέσα στη συνάρτηση η τιμή αλλάζει σε '{a_var}'.")

name = 'John'
print(f"Η τιμή της name είναι '{name}'.")
change_name(name)
print(f"Η τιμή της name παραμένει '{name}' έξω από τη αυνάρτηση.")
```

Κ. 26 - Αλλαγές στην τιμή της παραμέτρου μέσα στη συνάρτηση δεν επηρεάζει το αρχικό όρισμα.

Η εκτέλεση του κώδικα Κ. 26 δείχνει ότι το αρχικό όρισμα δεν επηρεάζεται από τις αλλαγές στην τιμή της παραμέτρου μέσα στη συνάρτηση, όπως φαίνεται στην Ε. 20.

```
Η τιμή της name είναι 'John'.
Μέσα στη συνάρτηση η τιμή αλλάζει σε 'Paul'.
Η τιμή της name παραμένει 'John' έξω από τη συνάρτηση.
```

Ε. 20 - Το αρχικό όρισμα δεν επηρεάζεται.

Στην εκτύπωση της εξόδου φαίνεται ότι η αλλαγή στην τιμή της μεταβλητής παραμέτρου a_var μέσα στη συνάρτηση change_name δεν επηρεάζει την τιμή του αρχικού ορίσματος name, που παραμένει να είναι John.

Το πέρασμα παραμέτρων έτσι ώστε να μην αλλάζει η τιμή ενός ορίσματος μέσα από τη συνάρτηση στην οποία περάστηκε ονομάζεται πέρασμα με τιμή (pass by value). Αυτός ο τρόπος προστατεύει την τιμή του ορίσματος. Αν χρειάζεται η συνάρτηση να αλλάξει την τιμή ενός ορίσματος, αυτό μπορεί να γίνει με άλλους τρόπους.

Εδώ πρέπει να επισημανθεί ότι η παραπάνω συζήτηση αφορά τους βασικούς τύπους μεταβλητών, όπως είναι οι αριθμοί και οι συμβολοσειρές. Αν το όρισμα που περνιέται σε μια συνάρτηση αφορά για παράδειγμα, λίστα, λεξικό ή αντικείμενο, τότε η συνάρτηση μπορεί να του αλλάξει τιμή, επειδή

στη συνάρτηση δεν περνιέται αντίγραφο του ορίσματος, αλλά μια *αναφορά* στο ίδιο το όρισμα. Σε αυτή την περίπτωση η συνάρτηση μπορεί να αλλάξει τα περιεχόμενα του ορίσματος.

2.10.7. Ορίσματα με λέξη-κλειδί

Στα μέχρι στιγμής προγράμματα τα ορίσματα περνάνε στις συναρτήσεις μέσω της θέσης των μεταβλητών παραμέτρων. Αυτός ο τρόπος χρησιμοποιείται στις περισσότερες γλώσσες προγραμματισμού. Η Python παρέχει και έναν άλλο τρόπο για να καθοριστεί σε ποια μεταβλητή παράμετρο θα περαστεί ένα όρισμα. Μέσα στην παρένθεση στην εντολή κλήσης μιας συνάρτησης μπορούμε να γραφεί το εξής:

`όνομα_παραμέτρου=τιμή`

Σε αυτή τη μορφή περάσματος τιμής, το όνομα_παραμέτρου είναι το όνομα της μεταβλητής παραμέτρου και η τιμή είναι η τιμή που περνάει σε αυτή την παράμετρο. Ένα όρισμα που γράφεται με αυτό τον τρόπο ονομάζεται *όρισμα με λέξη-κλειδί*.

Αυτός ο τρόπος κλήσης επιτρέπει την κλήση της συνάρτησης με τη σειρά των ορισμάτων με λέξη-κλειδί να μην παίζει ρόλο, ενώ γίνεται εμφανές ποια τιμή αντιστοιχεί σε ποια παράμετρο.

Στο παρακάτω πρόγραμμα, Κ. 27, η συνάρτηση που υπολογίζει την περίμετρο ενός παραλληλόγραμμου καλείται με ορίσματα με λέξη-κλειδί. Στο Ε. 21 εμφανίζεται η έξοδος που παράγει η εκτέλεση του προγράμματος.

```python
# Πρόγραμμα με συνάρτηση που υπολογίζει την περίμετρο παραλληλόγραμμου.

def main():
    """
    Παίρνει τα μήκη των πλευρών από το χρήστη, καλεί τη συνάρτηση υπολογισμού
    της περιμέτρου και τυπώνει το αποτέλεσμα.
    """
    side1_input = int(input('Πρώτη πλευρά παραλληλόγραμμου: '))
    side2_input = int(input('Δεύτερη πλευρά παραλληλόγραμμου: '))
    perimeter = calc_perimeter(side2=side2_input, side1=side1_input)
    print(f"Η περίμετρος είναι {perimeter}.")

def calc_perimeter(side1, side2):
    """
    Υπολογίζει και επιστρέφει την περίμετρο ενός παραλληλόγραμμου
    """
    perimeter = 2 * (side1 + side2)
    return perimeter
```

```
# Κλήση της main
main()
```
Κ. 27 - Κλήση συνάρτησης με όρισμα με λέξη-κλειδί.

```
Πρώτη πλευρά παραλληλόγραμμου: 8
Δεύτερη πλευρά παραλληλόγραμμου: 2
Η περίμετρος είναι 20 (κλήση με ορίσματα θέσης).
Η περίμετρος είναι 20 (κλήση με ορίσματα με λέξη-κλειδί).
```
Ε. 21 - Έξοδος του προγράμματος με κλήση συνάρτησης με λέξη-κλειδί.

Πρέπει να επισημανθεί ότι αν σε μια κλήση συνάρτησης περαστεί ένα όρισμα με τη μορφή όνομα_παραμέτρου=τιμή, τότε *όλα* τα υπόλοιπα ορίσματα θέσης πρέπει να περαστούν και αυτά με τη μορφή αυτή, αλλιώς προκαλείται συντακτικό λάθος.

Αν η συνάρτηση με ορισμό

```
def afunc(a, b, c, d):
    print(a, b, c,d)
```

κληθεί σαν

```
afunc(1, b=2, 3, 4)
```

το αποτέλεσμα είναι:

```
    afunc(1, b=2, 3, 4)
                     ^
SyntaxError: positional argument follows keyword argument
```

Η σωστή κλήση είναι η:

```
afunc(1, b=2, c=3, d=4)
```

2.10.8. Προαιρετικές παράμετροι συναρτήσεων

Υπάρχουν περιπτώσεις συναρτήσεων όπου μία παράμετρος τις περισσότερες φορές παίρνει μια συγκεκριμένη τιμή και σπάνια χρειάζεται να της δοθεί διαφορετική τιμή. Για τέτοιες περιπτώσεις μπορούν να χρησιμοποιηθούν οι *προαιρετικές παράμετροι* ή *παράμετροι με προκαθορισμένη τιμή*. Σε αυτές τις παραμέτρους δίνεται μια προκαθορισμένη τιμή στη δήλωση της συνάρτησης.

Αν στην κλήση της συνάρτησης περαστεί όρισμα για μια τέτοια παράμετρο, χρησιμοποιείται η τιμή του ορίσματος, αλλιώς η παράμετρος παίρνει την προκαθορισμένη τιμή. Αυτό επιτρέπει την κλήση συναρτήσεων με λιγότερα ορίσματα, κάνοντας πιο εύκολη την ανάγνωση του κώδικα και βοηθώντας στη μείωση πιθανών λαθών.

Οι προαιρετικές παράμετροι μπορεί να έχουν να κάνουν τόσο με ορίσματα θέσης, όσο και με ορίσματα με λέξη-κλειδί. Στην περίπτωση ορισμάτων θέσης, οι προαιρετικές παράμετροι τοποθετούνται μετά από όλες τις υποχρεωτικές παραμέτρους στη δήλωση αλλά και στην κλήση της συνάρτησης.

Στο πρόγραμμα Κ. 28 και οι δύο συναρτήσεις έχουν προαιρετική παράμετρο. Η `hello` έχει το πρόσωπο του χαιρετισμού, με προκαθορισμένη τιμή «σας» και η `value_calc` το ποσοστό του ΦΠΑ, με προκαθορισμένη τιμή το 24. Στην Ε. 22 εμφανίζεται η έξοδος του προγράμματος.

```
def hello(name="σας"):
    """
    Επιστρέφει έναν χαιρετισμό. Αν δεν δοθεί όνομα, χρησιμοποιεί το "σας".
    """
    return f"Καλημέρα {name}."

def value_calc(price, vat=24):
    """
    Υπολογίζει την τελική τιμή σαν την αρχική συν τον ΦΠΑ.
    Έχει προκαθορισμένη τιμή το 24%.
    """
    return price *  (1 + vat / 100)

aspirin_value = value_calc(4.60, vat=6)
milk_value = value_calc(0.90, 13)
gadget_value = value_calc(250)

print(f"{hello()} Η ασπιρίνη σας κοστίζει {aspirin_value:.2f}€.")
print(f"{hello("Γιώργο")} Το γάλα σας κοστίζει {milk_value:.02f}€.")
print(f"{hello("Μαρία")} Το gadget σας κοστίζει {gadget_value:.02f}€.")
```

Κ. 28 - Χαιρετισμός και τιμολόγηση με προαιρετικές παραμέτρους.

```
Καλημέρα σας. Η ασπιρίνη σας κοστίζει 4.88€.
Καλημέρα Γιώργο. Το γάλα σας κοστίζει 1.02€.
Καλημέρα Μαρία. Το gadget σας κοστίζει 310.00€.
```

Ε. 22 - Έξοδος με διάφορους χαιρετισμούς και τιμολογήσεις.

2.10.9. Συναρτήσεις με μεταβλητό πλήθος ορισμάτων

Η Python δίνει τη δυνατότητα να οριστούν συναρτήσεις που δέχονται μεταβλητό αριθμό ορισμάτων. Αυτό είναι χρήσιμο για περιπτώσεις που είτε δεν είναι γνωστό από την αρχή το πλήθος των ορισμάτων, είτε μεταβάλλεται από κλήση σε κλήση.

Υπάρχουν δύο είδη μεταβλητού πλήθους ορισμάτων, τα μεταβλητά ορίσματα θέσης (positional) και τα ονοματισμένα μεταβλητά ορίσματα (keyword arguments).

2.10.9.1 Μεταβλητά ορίσματα θέσης

Για να δηλωθεί ότι μια συνάρτηση χρησιμοποιεί μεταβλητά ορίσματα θέσης, χρησιμοποιείται ο τελεστής «*» στη δήλωση της συνάρτησης, ακολουθούμενος αμέσως από ένα όνομα παραμέτρου. Από σύμβαση, χρησιμοποιείται συνήθως το όνομα `args` (κάνοντας τη συνολική δήλωση «`*args`»), αλλά αυτό δεν είναι υποχρεωτικό και μπορεί να χρησιμοποιηθεί οποιοδήποτε όνομα.

Στον κώδικα Κ. 29 χρησιμοποιείται ένα μεταβλητό όρισμα θέσης για να υπολογιστεί το άθροισμα όλων των ορισμάτων.

```
def summa(s, *args):
    """
    Προσθέτει όλα τα μεταβλητά ορίσματα θέσης στο πρώτο όρισμα.
    """
    for i in args:
        s += i
    return s

print(f"Αθροισμα 1: {summa(1, 2, 3, 4, 5)}.")
print(f"Αθροισμα 2: {summa(3.14159, 2.718, 1.618)}.")
print(f"Αθροισμα 3: {summa("αβγά", " ", "και", " ", "spam")}.")
```

Κ. 29 - Συνάρτηση με μεταβλητό όρισμα θέσης

.

Η εκτέλεση του κώδικα παράγει τα ακόλουθα αποτελέσματα:

```
Αθροισμα 1: 15.
Αθροισμα 2: 7.47759.
Αθροισμα 3: αβγά και spam.
```

Ε. 23 - Έξοδος προγράμματος με συνάρτηση με μεταβλητό όρισμα θέσης.

Αξίζει να σημειωθεί ότι με λίγη προσοχή στον τρόπο υλοποίησης της συνάρτησης, αυτή μπορεί να χρησιμοποιηθεί για πολλούς τύπους ορισμάτων. Στο παράδειγμα χρησιμοποιείται για ακέραιους, πραγματικούς, αλλά και συμβολοσειρές.

Η χρήση μιας παραμέτρου θέσης πριν τα μεταβλητά ορίσματα επιτρέπει τη χρήση της συνάρτησης για όσους τύπους μεταβλητών υποστηρίζουν τον τελεστή «+» και όχι μόνο αριθμούς. Αυτός ο τρόπος συγγραφής κώδικα, που είναι γενικός και μπορεί να χρησιμοποιηθεί από πολλούς τύπους μεταβλητών, προτιμάται στην Python και συναντιέται συχνά.

2.10.9.2 Ονοματισμένα μεταβλητά ορίσματα

Τα ονοματισμένα μεταβλητά ορίσματα επιτρέπουν να περαστούν σε μια συνάρτηση ονοματισμένα ορίσματα που δεν εμφανίζονται στον ορισμό της. Δηλώνονται με τον τελεστή «**» και αποτελούν ένα λεξικό.

Τα λεξικά αναπτύσσονται στο κεφάλαιο 4, εδώ αρκεί η γνώση ότι αποτελούνται από ζεύγη που έχουν τη μορφή `κλειδί: τιμή` και ότι αν d είναι ένα λεξικό, για να προσπελαστεί η `τιμή` χρησιμοποιείται η έκφραση `d[κλειδί]`.

Ένας απλός κώδικας για τη χρήση τους είναι ο Κ. 30, που παράγει την έξοδο Ε. 24.

```
def order(price, **kwargs):
    """
    Τυπώνει όλα τα μεταβλητά ονοματισμένα ορίσματα.
    """
    print(f"Τιμή προϊόντος: {price}.")
    # Η μέθοδος items() επιστρέφει τα στοιχεία του λεξικού σε μορφή
    # κατάλληλη για το for (λίστα με μέλη ζεύγη - tuples)
    for key, value in kwargs.items():
        print(f"{key}: {value}.")

order(24.99, fpa=24, discount=15, shipping=4.99)
order(99.01, fpa=13, shipping=2.49)
order(9.99, discount=15)
```
Κ. 30 - Συνάρτηση με ονοματισμένα μεταβλητά ορίσματα.

```
Τιμή προϊόντος: 24.99.
fpa: 24.
discount: 15.
shipping: 4.99.
Τιμή προϊόντος: 99.01.
fpa: 13.
shipping: 2.49.
Τιμή προϊόντος: 9.99.
discount: 15.
```
Ε. 24 - Έξοδος του προγράμματος με ονοματισμένα μεταβλητά ορίσματα.

2.10.10. Άσκηση 2 – μεταβλητά ορίσματα θέσης

Γράψτε μια συνάρτηση που να υπολογίζει και να επιστρέφει την περίμετρο τυχαίου πολυγώνου. Σαν ορίσματα να δέχεται τα μήκη των πλευρών του πολυγώνου.

Όπως είναι λογικό, πρέπει να δίνονται τα μήκη τουλάχιστον τριών πλευρών σαν ορίσματα.

Η λύση της άσκησης υπάρχει στο Παράρτημα Α'.

2.10.11. Παράμετροι μόνο με λέξη-κλειδί

Από την έκδοση 3 τηςPython, μπορούν σε μια συνάρτηση να υπάρχουν παράμετροι στις οποίες όταν καλείται η συνάρτηση, περνιέται τιμή *μόνο* με τη χρήση λέξης-κλειδιού (με τη μορφή όνομα_παραμέτρου=τιμή).

Αυτές οι παράμετροι είναι *υποχρεωτικές* αν στη δήλωση της συνάρτησης δεν ορίζεται προκαθορισμένη τιμή, αλλά δηλώνεται μόνο το όνομα της παραμέτρου.

Αν στη δήλωση της συνάρτησης δίνεται και προκαθορισμένη τιμή, τότε η παράμετρος είναι προαιρετική.

Αυτού του του τύπου οι παράμετροι, αν υπάρχουν, πρέπει να δηλώνονται μετά τα μεταβλητά ορίσματα θέσης (*args) και πριν τα ονοματισμένα μεταβλητά ορίσματα (**kwargs):

```
def afunc(p1, p2, *args, ka1="a", ka2, **kwargs):
```

Στην παραπάνω δήλωση συνάρτησης, τα ka1 και ka2 είναι παράμετροι μόνο με λέξη-κλειδί, όπου το ka1 είναι προαιρετικό, ενώ το ka2 υποχρεωτικό, αφού δεν έχει προκαθορισμένη τιμή.

Αν δεν υπάρχουν μεταβλητά ορίσματα θέσης, τότε αμέσως πριν τις παραμέτρους μόνο με λέξη-κλειδί πρέπει να υπάρχει μια κενή (null) παράμετρος που αποτελείται μόνο από ένα «*», το οποίο δεν αντιστοιχεί σε κανένα όρισμα:

```
def afunc(p1, p2, *, ka1="a", ka2):
```

Στη συνάρτηση που εμφανίζεται στον κώδικα Κ. 31 υπάρχουν τέτοιου τύπου παράμετροι, οι giftwrap και discount.

2.10.12. Κανόνες για το συνδυασμό των διαφόρων τύπων ορισμάτων

Η δυνατότητα χρήσης διαφορετικών τύπων ορισμάτων δημιουργεί και την ανάγκη ύπαρξης κάποιων κανόνων για το πώς θα χρησιμοποιούνται.

Στον ορισμό των συναρτήσεων, ακολουθείται η εξής σειρά:

- Πρώτες δηλώνονται όλες οι παράμετροι θέσης.
- Κατόπιν οι ονοματισμένες (ή προαιρετικές) παράμετροι της μορφής `όνομα_παραμέτρου=τιμή`.
- Ακολουθεί η δήλωση για τα μεταβλητά ορίσματα θέσης.
- Μετά από αυτήν οι παράμετροι μόνο με λέξη-κλειδί.
- Τελευταία είναι πάντα η δήλωση για τα ονοματισμένα μεταβλητά ορίσματα.

Η σειρά αυτή έχει σκοπό να είναι σαφές στον διερμηνευτή τι είδους (και ποιο) είναι κάθε όρισμα.

Αντίστοιχα, στις κλήσεις των συναρτήσεων ακολουθείται η παρακάτω σειρά:

- Πρώτα περνιούνται όλα τα ορίσματα θέσης
- Κατόπιν, αν υπάρχουν, τα ονοματισμένα ορίσματα (η σειρά μεταξύ τους δεν είναι αυστηρή). Αν δεν υπάρχουν ονοματισμένα μεταβλητά ορίσματα (`**kwargs`), τότε το αναγνωριστικό (όνομα) κάθε ορίσματος πρέπει να ταιριάζει με το όνομα κάποιας από τις παραμέτρους στον ορισμό της συνάρτησης.
- Ακολουθούν τα μεταβλητά ορίσματα θέσης.
- Μετά είναι τα ορίσματα μόνο με λέξη-κλειδί.
- Τέλος, τα μεταβλητά ονοματισμένα ορίσματα είναι πάντα τελευταία, εφόσον υπάρχουν.

Τα παραπάνω θα συζητηθούν με βάση τη συνάρτηση που εμφανίζεται στον Κ. 31.

```
def bill(price, count, *extras, giftwrap=True, discount, **payment):
    giftwrap_price = 3.80

    total_cost = price * count

    for bill in extras:
        total_cost += bill

    if giftwrap:
        total_cost += giftwrap_price

    if discount > 0:
        total_cost = total_cost * (100 - discount) / 100

    print(f"{payment=}")
    return total_cost
```
Κ. 31- Συνάρτηση που παίρνει διαφόρων ειδών ορίσματα.

Η συνάρτηση `bill()` δέχεται όλων των ειδών τα ορίσματα: θέσης, μεταβλητά ορίσματα θέσης, ορίσματα με λέξη-κλειδί και ονοματισμένα μεταβλητά ορίσματα.

Η κλήση της `bill()` που φαίνεται στο Κ. 32 είναι σωστή και παράγει την έξοδο Ε. 25.

```
tot_cost = bill(100, 2, 30, 20, giftwrap=False, discount=10, afm="046774356",
                card_no="9999 8888 3333 1111"
                )
print(f"Συνολικό κόστος: {tot_cost:.2f}.")
```
Κ. 32 - Μια σωστή κλήση της bill().

```
payment={'afm': '046774356', 'card_no': '9999 8888 3333 1111'}
Συνολικό κόστος: 225.00.
```
Ε. 25 - Έξοδος της σωστής κλήσης.

Η κλήση που φαίνεται στο Κ. 33 προκαλεί συντακτικό λάθος, όπως φαίνεται στην έξοδο που προκαλεί η εκτέλεσή της, στο Ε. 26. Το γεγονός ότι το λάθος είναι συντακτικό σημαίνει ότι ο κώδικας (εδώ, η κλήση της `bill()`) δεν έφτασε ποτέ στο σημείο να εκτελεστεί.

```
# Κλήση που προκαλεί συντακτικό λάθος:

tot_cost = bill(100, 2, 30, giftwrap=False, 20, discount=10, afm="046774356",
                card_no="9999 8888 3333 1111"
                )
print(f"Συνολικό κόστος: {tot_cost:.2f}.")
```
Κ. 33 - Κλήση που προκαλεί συντακτικό λάθος.

```
  File "c:\XXX\YYY\ZZZ\Python_A\arg-rules.py", line 34
    )
    ^
SyntaxError: positional argument follows keyword argument
```
Ε. 26 - Εκτέλεση κώδικα με συντακτικό λάθος.

Και η κλήση στο Κ. 34 προκαλεί συντακτικό λάθος, όπως δείχνει η έξοδος Ε. 27.

```
# Και αυτή η κλήση προκαλεί συντακτικό λάθος:

tot_cost = bill(100, count=2, 30, 20, giftwrap=False, discount=10, afm="046774356",
 card_no="9999 8888 3333 1111"
               )
print(f"Συνολικό κόστος: {tot_cost:.2f}.")
```
Κ. 34 – Και αυτή η κλήση προκαλεί συντακτικό λάθος.

```
  File "c:\XXX\YYY\ZZZ\Python_A\arg-rules.py", line 44
    )
    ^
```

```
SyntaxError: positional argument follows keyword argument
```
Ε. 27 - Άλλο ένα συντακτικό λάθος.

Η κλήση στην Κ. 35 προκαλεί λάθος κατά την εκτέλεση του κώδικα (της συνάρτησης `bill()`), όπως φαίνεται και από την έξοδο Ε. 28. Σε αυτή την περίπτωση ο κώδικας είναι συντακτικά σωστός, αλλά όταν ο διερμηνευτής πάει να εκτελέσει τη συνάρτηση, βρίσκει ότι λείπει ένα υποχρεωτικό όρισμα, το `discount`.

```
# Κλήση που προκαλεί λάθος κατά την εκτέλεση του κώδικα:

tot_cost = bill(100, 2, 30, 20, giftwrap=False, afm="046774356",
                card_no="9999 8888 3333 1111"
                )
print(f"Συνολικό κόστος: {tot_cost:.2f}.")
```
Κ. 35 - Κλήση που προκαλεί λάθος κατά την εκτέλεση του κώδικα.

```
Traceback (most recent call last):
  File "c:\XXX\YYY\ZZZ\Python_A\arg-rules.py", line 52, in <module>
    tot_cost = bill(100, 2, 30, 20, giftwrap=False, afm="046774356",
                    card_no="9999 8888 3333 1111"
                    )
TypeError: bill() missing 1 required keyword-only argument: 'discount'
```
Ε. 28 – Έξοδος με λάθος κατά την εκτέλεση του κώδικα.

2.11. Αναδρομή

Η αναδρομή (recursion) είναι ένας τρόπος υπολογισμού συναρτήσεων όπου η τιμή που ζητείται να υπολογιστεί βασίζεται σε μια ή περισσότερες προηγούμενες τιμές της συνάρτησης. Αυτό σημαίνει ότι για να υπολογιστεί η τρέχουσα τιμή της συνάρτησης πρέπει η συνάρτηση να καλέσει τον εαυτό της για τον υπολογισμό των προηγούμενων τιμών από τις οποίες εξαρτάται η τρέχουσα και μετά για κάποιες προηγούμενες, μέχρι να φτάσει σε κάποιες τιμές που είναι αρχικές.

Μια γνωστή μαθηματική συνάρτηση που είναι αναδρομική είναι το παραγοντικό (n!):

$n! = n * (n-1)!$ (που οδηγεί και στην εναλλακτική μορφή $n! = n * (n-1) * (n-2) * \ldots * 1$)

όπου:

$1! = 1$

$0! = 1$

Έστω ότι πρέπει να υπολογιστεί η τιμή 5!. Από τον τύπο της είναι σαφές ότι 5! = 5 * 4!, αλλά 4! = 4 * 3!, κατόπιν 3! = 3 * 2!, και 2! = 2 * 1!. Τέλος, 1! = 1, οπότε μπορεί να υπολογιστεί πλέον το 2!, κατόπιν το 3!, μετά το 4! και τέλος το ζητούμενο 5!

Παρακάτω ο κώδικας Κ. 36 υλοποιεί τη συνάρτηση του παραγοντικού. Η εκτέλεσή του παράγει την έξοδο Ε. 29.

```
def factorial(n):
    """
    Υπολογίζει το παραγοντικό ενός θετικού ακεραίου.
    """
    assert n > 0 and isinstance(n, int), "Δεν είναι θετικός ακέραιος."
    if n == 1:
        return 1
    else:
        return n * factorial(n -1)

for i in range(5, 25, 5):
    print(f"{i}! = {factorial(i)}.")
```

Κ. 36 - Υπολογισμός παραγοντικού με αναδρομική συνάρτηση.

Η εκτέλεσή του παράγει την έξοδο **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**:

```
5! = 120.
10! = 3628800.
15! = 1307674368000.
20! = 2432902008176640000.
```

Ε. 29 - Τα παραγοντικά τεσσάρων θετικών ακεραίων.

Οι αναδρομικές συναρτήσεις δίνουν έναν απλό και κομψό τρόπο γραφής συναρτήσεων στις περιπτώσεις που μπορούν να εφαρμοστούν, ενώ σε πολλές περιπτώσεις είναι και ο μοναδικός τρόπος υλοποίησης μιας λύσης.

Η αναδρομή όμως έχει και αρκετά προβλήματα: καταναλώνεται πολλή μνήμη, (όσο περισσότερες οι διαδοχικές αναδρομικές κλήσεις, τόσο περισσότερη μνήμη), είναι αργή σαν μέθοδος (κάθε κλήση σημαίνει δημιουργία του περιβάλλοντος εκτέλεσης μιας συνάρτησης, συν το χρόνο κλήσης και επιστροφής) και όπως με τη μνήμη, όσο περισσότερες οι αναδρομικές κλήσεις, τόσο πιο πολύ αργεί το αποτέλεσμα. Οι διαδοχικές κλήσεις της συνάρτησης από τον εαυτό της μένουν ανοιχτές μέχρι να φτάσουν στην αρχική τιμή. Μόνο τότε αρχίζουν να επιστρέφουν, με την αντίστροφη σειρά κλήσης.

2.11.1. Άσκηση 3 - αναδρομή

Μια άλλη γνωστή αναδρομική σχέση είναι η ακολουθία Fibonacci:

$F(n) = F(n-1) + F(n-2)$

Όπου:

$F(0) = 0$

$F(1) = 1$.

Σε αυτήν, για να υπολογιστεί, για παράδειγμα, η τιμή του F(5), χρειάζεται να βρεθούν πρώτα οι τιμές του F(4) και του F(3), κ.ο.κ..

Γράψτε μια συνάρτηση που να υπολογίζει την ακολουθία Fibonacci. Υπολογίστε την τιμή της για τους αριθμούς 5, 10, 20 και 40.

Προσθέστε σαν πρώτη γραμμή της συνάρτησης να τυπώνει τον τρέχοντα όρο που πρόκειται να υπολογιστεί. Μήπως παρατηρείτε ότι υπάρχουν πολλοί όροι που υπολογίζονται πολλές φορές;

2.12. Προγραμματισμός: δομημένος, τμηματικός, διαδικασιακός ή αντικειμενοστραφής;

Οι πρώτες γλώσσες προγραμματισμού (σχετικά) υψηλού επιπέδου είχαν λίγες και όχι ιδιαίτερα ευέλικτες δομές ελέγχου της ροής ενός προγράμματος. Η χρήση της εντολής goto ήταν (αναγκαστικά) συχνή, δυσκολεύοντας πολύ την παρακολούθηση της ροής ενός προγράμματος. Ήταν αρκετά κοντά στις γλώσσες μηχανής από αυτή την άποψη. Ο κώδικας ήταν συνήθως δυσνόητος και επιρρεπής στα προγραμματιστικά λάθη.

Αυτό οδήγησε στην εξέλιξη των γλωσσών και τη δημιουργία προγραμματιστικών/αλγοριθμικών δομών υψηλότερου επιπέδου, που είναι πολύ πιο ευέλικτες και πολύ πιο εύκολα αναγνώσιμες από τον άνθρωπο. Δομές όπως το `if/else`, οι επαναλήψεις τύπου `for` και `while` τότε άρχισαν να χρησιμοποιούνται.

Η χρήση τέτοιων δομών στον προγραμματισμό βοηθάει στη δημιουργία καλύτερης ποιότητας κώδικα, με πολύ μεγαλύτερη σαφήνεια και πολύ ευκολότερα κατανοητό από τον άνθρωπο. Αυτός ο τρόπος προγραμματισμού, με τη χρήση των διαφόρων δομών ελέγχου ροής και επαναλήψεων λέγεται *δομημένος προγραμματισμός*.

Ο *τμηματικός προγραμματισμός* είναι ένας τρόπος *οργάνωσης* ομάδων συναρτήσεων που έχουν κοινό αντικείμενο σε τμήματα, ώστε να διευκολύνεται η κατανόηση της λειτουργίας τους και η συντήρηση του κώδικα.

Ο *διαδικασιακός προγραμματισμός* είναι ένας τρόπος προγραμματισμού όπου τα προγράμματα δομούνται σε ένα σύνολο *διαδικασιών,* με την έννοια μιας αυτόνομης μονάδας που επιτελεί μια λειτουργία (υπάρχει και σχετική σημείωση στο υποκεφάλαιο 2.3). Στην Python τέτοιες μονάδες είναι οι συναρτήσεις. Οι διαδικασίες επενεργούν σε *δομές δεδομένων*. Η ομαδοποίηση αυτών των διαδικασιών σε αυτόνομες μονάδες που υλοποιούν μια συγκεκριμένη λειτουργικότητα συνδέει τον διαδικασιακό προγραμματισμό με τον τμηματικό.

Ο *αντικειμενοστραφής προγραμματισμός* (object-oriented programming, OOP) χειρίζεται *αντικείμενα* (objects) και χρησιμοποιεί την τεχνική της ενθυλάκωσης, όπου δεδομένα και μηχανισμοί που επενεργούν σε αυτά (όπως οι συναρτήσεις, που στον αντικειμενοστραφή προγραμματισμό ονομάζονται *μέθοδοι*) δένονται μαζί σε ένα ενιαίο δέμα. Ο πιο συνηθισμένος τρόπος για να γίνει αυτό είναι μέσω των *κλάσεων*. Οι κλάσεις είναι ένα είδος τμηματικού προγραμματισμού. Κάθε κλάση είναι ένα τμήμα κώδικα που αποτελείται από δεδομένα και από τις μεθόδους που επενεργούν σε αυτά, άρα είναι ένα ανεξάρτητο *τμήμα*.

Αναλυτικότερη παρουσίαση του αντικειμενοστραφούς προγραμματισμού γίνεται σε επόμενο κεφάλαιο.

2.13. Προχωρημένες δυνατότητες της Python για συναρτήσεις

Η Python προσφέρει πολλές δυνατότητες που σχετίζονται με τις συναρτήσεις. Αρκετές από αυτές είναι αρκετά προχωρημένες.

Αναφέρονται ενδεικτικά τα παρακάτω:

- Συναρτήσεις λάμδα (lambda functions) – ονομάζονται και ανώνυμες (anonymous) συναρτήσεις επειδή δεν έχουν όνομα
- Γεννήτριες (generators)
- μερικές συναρτήσεις (partial functions)
- συναρτήσεις υψηλότερης τάξης (higher order functions)
- διακοσμητές (decorators)
- συρρουτίνες (coroutines)
- κλειστότητες (closures)

Η χρήση τους απαιτεί την απόκτηση κάποιας εμπειρίας στον προγραμματισμό η/και εξοικείωση γύρω από προχωρημένες τεχνικές προγραμματισμού και δεν καλύπτονται από την ύλη αυτού του συγγράμματος.

2.14. Ολοκληρωμένα περιβάλλοντα ανάπτυξης (Integrated Development Environments, IDEs)

Για το έργο της ανάπτυξης λογισμικού έχουν δημιουργηθεί πολλά βοηθητικά εργαλεία. Κλασικό τέτοιο εργαλείο είναι ο διορθωτής κειμένου (editor), που κάνει πιο εύκολη τη συγγραφή κώδικα. Μπορεί να προσφέρει βασικές λειτουργίες όπως η αναζήτηση και αντικατάσταση κειμένου, αλλά και αναγνώριση της σύνταξης μιας γλώσσας προγραμματισμού και ανάδειξής της στον κώδικα με χρήση διαφορετικών χρωμάτων για λέξεις-κλειδιά, μεταβλητές, αντικείμενα κλπ.

Ξεκινώντας από τους διορθωτές κειμένου αναπτύχθηκαν εργαλεία που προσφέρουν πλήρη υποστήριξη της διαδικασίας της ανάπτυξης λογισμικού, τα ολοκληρωμένα περιβάλλοντα ανάπτυξης (Integrated Development Environments, IDEs).

Μπορούν να αναγνωρίσουν τη γλώσσα προγραμματισμού και να την αναδεικνύουν στον κώδικα, να συμπληρώνουν αυτόματα τη σύνταξη εντολών, π.χ. με τη συμπλήρωση λέξεων-κλειδιών, παρενθέσεων, εισαγωγικών κλπ.

Επίσης, ιδίως για γλώσσες που χρησιμοποιούν μεταγλωττιστή, μπορούν να αναλάβουν όλο τον κύκλο παραγωγής του τελικού εκτελέσιμου κώδικα, να βοηθήσουν στην εκσφαλμάτωση (debugging), να εφαρμόσουν σουίτες δοκιμών για τον κώδικα (testing), τη διαχείριση του ελέγχου των εκδόσεων του κώδικα (version control), κλπ..

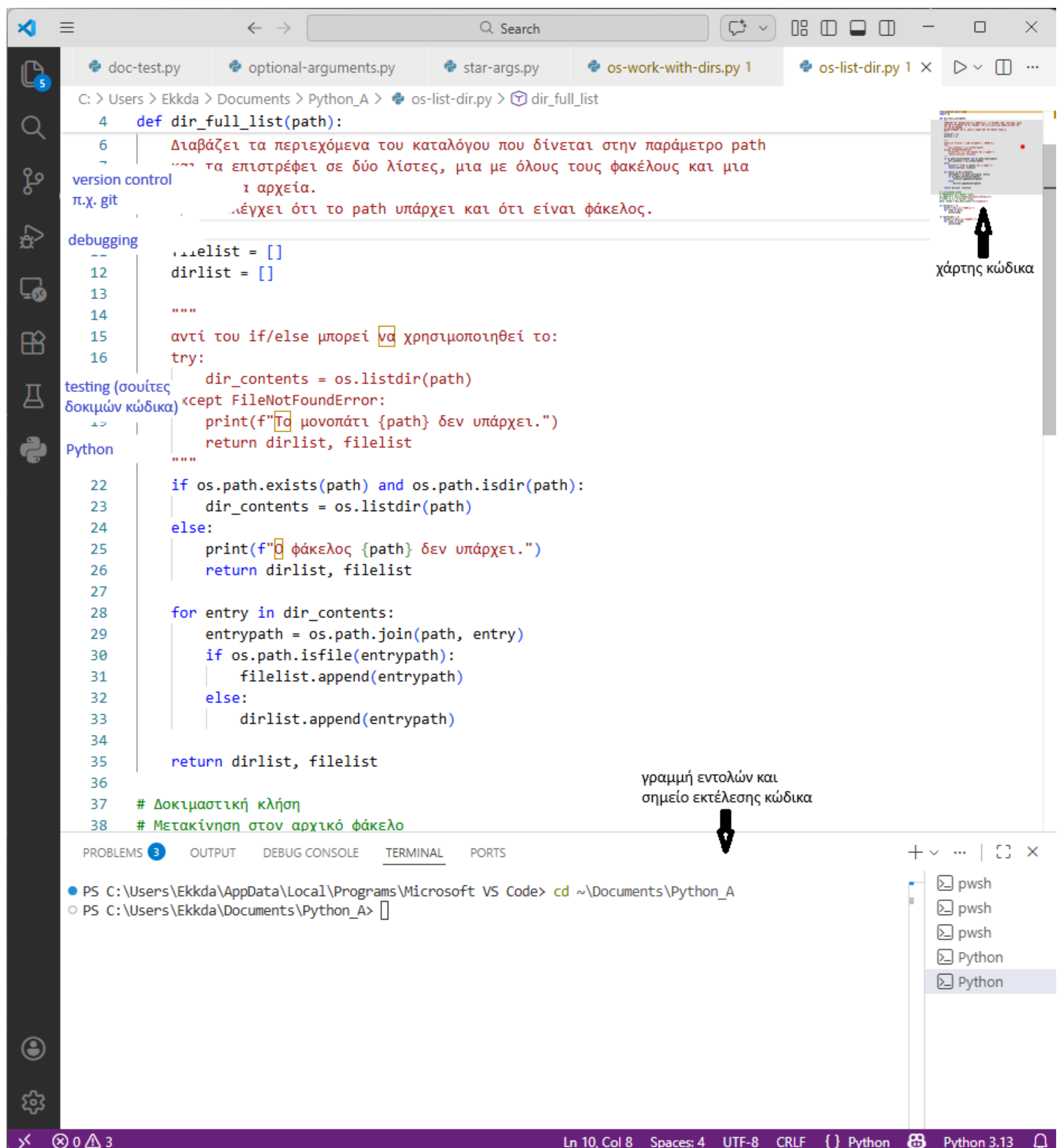
Επίσης συνήθως παρέχουν και περιβάλλον γραμμής εντολών, όπου εκτελείται ο κώδικας και εμφανίζεται η έξοδος και τυχόν μηνύματα λάθους, αλλά μπορούν και να δοθούν εντολές του λειτουργικού συστήματος, π.χ. να γίνει μετάβαση σε συγκεκριμένο φάκελο για την εκτέλεση του προγράμματος.

Υπάρχουν αρκετά IDEs που υποστηρίζουν την Python. Άλλα είναι εμπορικά προϊόντα, άλλα είναι ελεύθερο ή δωρεάν λογισμικό, άλλα υποστηρίζουν ένα λειτουργικό σύστημα και άλλα υποστηρίζουν πολλά λειτουργικά συστήματα.

Μερικά από τα πιο δημοφιλή είναι τα PyCharm, Anaconda Spyder, Eclipse και VS Code.

Το VS Code είναι ένα IDE που διατίθεται από την Microsoft σαν ελεύθερο λογισμικό. Υποστηρίζει πολλές γλώσσες προγραμματισμού, είναι επεκτάσιμο μέσω extensions και προσφέρει πολλές δυνατότητες. Εκτός από τις δυνατότητες των IDEs που έχουν ήδη αναφερθεί, το VS Code έχει πολλά επιπλέον χαρακτηριστικά. Μερικά από αυτά, που αφορούν και την Python, είναι:

- Υποστήριξη για ιδεατά περιβάλλοντα (virtual environments), για εγκατάσταση τόσο της Python, όσο και βιβλιοθηκών ανεξάρτητα για κάθε εφαρμογή που αναπτύσσεται.
- Υποστήριξη για Jupyter notebooks.
- Δυνατότητα για εγκατάσταση επεκτάσεων για την Python.
- Έλεγχος των εκδόσεων του κώδικα και του ιστορικού των αλλαγών στον κώδικα με υποστήριξη για συλλογική ανάπτυξη εφαρμογών από ομάδες.
- Δυνατότητα συνεργασίας σε πραγματικό χρόνο ομάδας προγραμματιστών μέσω του live share.



Εικόνα 10 - Το κύριο παράθυρο του VS Code.

Στην Εικόνα 10 φαίνεται το κύριο παράθυρο του VS Code, με επισημάνσεις σε μερικά ενδιαφέροντα σημεία.

Στην πάνω δεξιά γωνία φαίνεται ένας χάρτης του κώδικα του οποίου γίνεται επεξεργασία τη συγκεκριμένη στιγμή και το γκριζαρισμένο κομμάτι είναι αυτό που φαίνεται στο κύριο παράθυρο.

Στο κάτω μέρος του παράθυρου φαίνεται ένα παράθυρο-παιδί που έχει γραμμή εντολών. Σε αυτή μπορούν να δοθούν εντολές και επίσης εκεί εκτελείται ο κώδικας.

Στην αριστερή κάθετη μαύρη στήλη φαίνονται τα εικονίδια από διάφορα εργαλεία.

Το πρώτο από επάνω, που έχει και τον γαλάζιο κύκλο, ανοίγει μια στήλη περιήγησης με τα αρχεία που είναι ανοιχτά στο VS Code (ο αριθμός στον γαλάζιο κύκλο δείχνει πόσα δεν έχουν σωθεί ενώ έχουν αλλαγές), τον τρέχοντα φάκελο από τον οποίο φορτώνει αρχεία το VS Code, ένα περίγραμμα του κώδικα, και μια χρονοσειρά αλλαγών.

Το εικονίδιο με τα τρία κυκλάκια και τις γραμμές που τα ενώνουν είναι για τη διαχείριση των εκδόσεων του κώδικα μέσω του git. Απαιτεί εγκατάσταση του git για Windows.

Το ακριβώς από κάτω εικονίδιο είναι για την εκσφαλμάτωση του κώδικα. Μπορεί να εκτελεί τον κώδικα γραμμή-γραμμή, να σταματάει σε σημεία ελέγχου που θέτει ο χρήστης/ρια (breakpoints) και πολλά άλλα.

Με το εικονίδιο με τον πεπλατυσμένο δοκιμαστικό σωλήνα μπορούν να εφαρμοστούν σουίτες δοκιμών στον κώδικα. Απευθύνεται περισσότερο σε έργα με μεγάλα μεγέθη κώδικα.

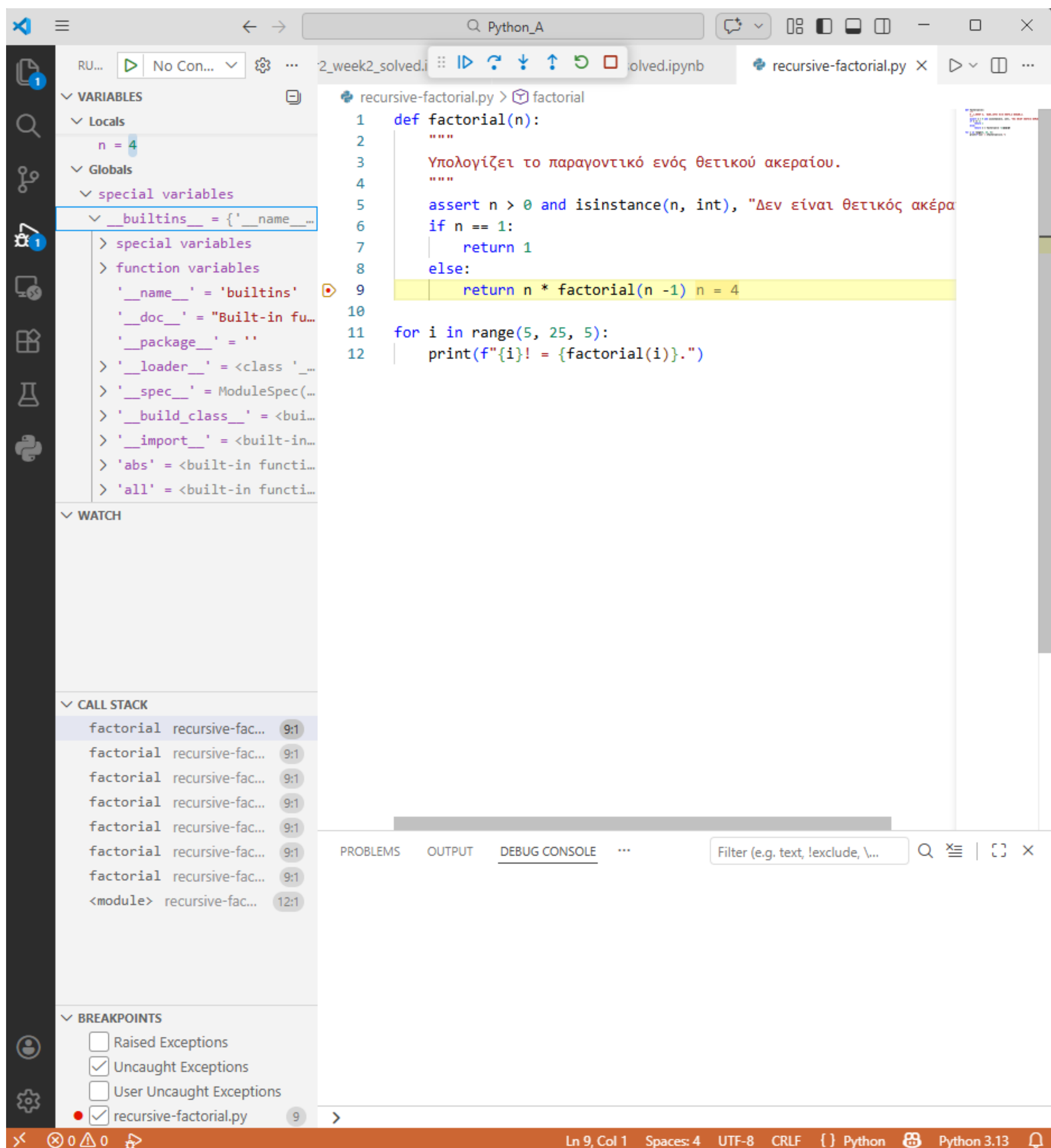
Το τελευταίο εικονίδιο, με το σήμα της Python, είναι για τη διαχείριση έργων σε Python και για τη δημιουργία εικονικών περιβαλλόντων για έργα Python, κλπ.

Στην Εικόνα 11 φαίνεται το περιβάλλον εκσφαλμάτωσης κώδικα του VS Code. Το πρόγραμμα έχει σταματήσει στη γραμμή 9, που έχει τεθεί ένα σημείο σταματήματος (breakpoint). Αυτό φαίνεται από το βελάκι με την κόκκινη τελεία αριστερά από τον αριθμό 9. Δεξιά στη γραμμή το VS Code δείχνει την τρέχουσα τιμή της n (`n = 4`). Η αρχική κλήση έγινε για το 10!.

Αριστερά πάνω, κάτω από το «VARIABLES» φαίνονται οι τοπικές και καθολικές μεταβλητές και οι τιμές τους, ενώ στο κάτω μέρος, με την επικεφαλίδα «CALL STACK» φαίνεται σε βάθος πόσων επιπέδων κλήσης βρίσκεται αυτή τη στιγμή το πρόγραμμα. Η πάνω-πάνω γραμμή δείχνει την τρέχουσα κλήση της `factorial`. Συνολικά το πρόγραμμα βρίσκεται σε βάθος 7 κλήσεων της `factorial`. Γνωρίζοντας ότι η αρχική κλήση έγινε με όρισμα 10, το βάθος των επτά κλήσεων συμφωνεί με την τρέχουσα τιμή (4). Η τελευταία γραμμή («<module>») αναφέρεται στην εκτέλεση του κώδικα του προγράμματος, με το βρόχο `for` στις γραμμές 11 και 12.

Στο παραθυράκι στο πάνω μέρος του παράθυρου, πάνω από τον κώδικα, είναι τα χειριστήρια για την εκσφαλμάτωση: τρέξιμο και σταμάτημα του κώδικα, εκτέλεση γραμμή-γραμμή, κλπ.

Αν πατηθεί το κύριο μενού του VS Code (τρεις οριζόντιες γραμμές στην πάνω αριστερή γωνία), επιλέγοντας την τελευταία επιλογή (Help), εμφανίζεται ένα πλούσιο μενού επιλογών βοήθειας, που αξίζει να εξερευνηθεί αν σκοπεύει κανείς να χρησιμοποιεί το VS Code συστηματικά.



Εικόνα 11 - Το περιβάλλον εκσφαλμάτωσης του VS Code.

2.15. Ερωτήσεις κλειστού τύπου

1. Ο διεθνής όρος για την τεχνική της διάσπασης ενός προβλήματος σε μικρότερα υποπροβλήματα είναι:
    A. Search and replace
    A. Divide and conquer
    B. Divide and multiply
    C. Conquer the castle
2. Στην Python, όταν μια συνάρτηση δεν επιστρέφει ρητά μια τιμή, στην πραγματικότητα επιστρέφει:
    A. 0
    B. False
    C. None
    D. True
3. Έχουμε τον παρακάτω κώδικα:

```
glob0 = 5

def f1(a):
  glob0 = 100 * a
  return glob0

def f2(a):
  global glob0
  glob0 = glob0 * a

res1 = f1(5)
print(f"glob0: {glob0}, res1: {res1}")
res2 = f2(100)
print(f"glob0: {glob0}, res2: {res2}")
```

Τι θα τυπώσει όταν τον τρέξουμε;

A. `glob0: 5, res1: 500`
   `glob0: 500, res2: None`
B. `glob0: 500, res1: 500`
   `glob0: 5, res2: 0`
C. `glob0: 5, res1: 5`
   `glob0: 5, res2: None`
D. `glob0: 50, res1: 50`
   `glob0: 5, res2: 500`

4. Ποιες από τις παρακάτω συναρτήσεις είναι έγκυρες στην Python;

A.
```
def func1():
    pass
```
B.
```
def func1():
    break
```
C.
```
def func1():
    continue
```
D.
```
def func1():
    ....
```

5. Έχουμε την παρακάτω συνάρτηση Python, που υπολογίζει την περίμετρο ενός πολυγώνου, ανεξάρτητα του αριθμού των πλευρών του:

```
def perimeter(a, b, c, *d):
    length = a + b + c
    for side in d:
        length = length + side
    return length
```

Ποια από τις παρακάτω κλήσεις της είναι σωστή;

```
A. perimeter(4, 3, 2, 6, 5, 10, c=3)
B. perimeter(4, 3, 2, 6, 5, c=7, 3)
C. perimeter(4, 3)
D. perimeter(4, 3, 2, 6, 5, 10, 3)
```

2.16. Ασκήσεις προς επίλυση

1. Γράψτε ένα πρόγραμμα που να δέχεται σαν είσοδο μια συμβολοσειρά που να αποτελείται από έναν δεκαδικό αριθμό και ένα γράμμα, π.χ. 31.5C. Υποθέστε τα παρακάτω:

- Αν το γράμμα είναι F, τότε ο αριθμός είναι θερμοκρασία σε βαθμούς Φαρενάιτ και πρέπει να τον μετατρέψετε σε βαθμούς Κελσίου.
- Αν το γράμμα είναι C, τότε πρόκειται για βαθμούς Κελσίου και πρέπει να μετατραπεί σε βαθμούς Φαρενάιτ.
- Αν το γράμμα είναι k, τότε πρόκειται για χιλιόμετρα και πρέπει να τα μετατρέψετε σε μίλια.
- Αν το γράμμα είναι m, τότε είναι μίλια (ξηράς) και πρέπει να μετατραπούν σε χιλιόμετρα.
- Αν η είσοδος είναι οτιδήποτε διαφορετικό, να τυπώνει ένα κείμενο βοήθειας για τον τρόπο χρήσης του προγράμματος.

Το πρόγραμμα πρέπει να διαβάζει την είσοδο, να καταλαβαίνει τι είναι, να καλεί μια αντίστοιχη συνάρτηση μετατροπής και να τυπώνει το αποτέλεσμα. Στις συναρτήσεις να υπάρχουν τα αντίστοιχα docstrings.

2. Η εξίσωση που περιγράφει μια ευθεία στο επίπεδο είναι η:

$y = ax + b$ (1)

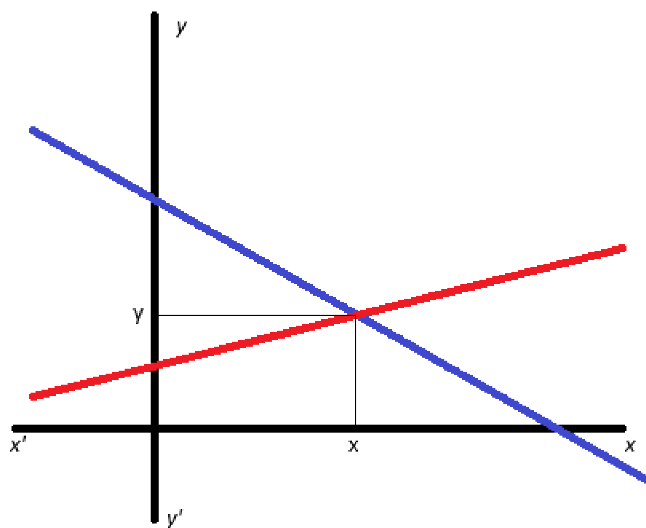
όπου a και b είναι αριθμητικές σταθερές, x η μεταβλητή της ευθείας (άξονας X) και y η τιμή της (άξονας Y).

Έχουμε δύο ευθείες:

$y_1 = a_1x + b_1$ (2)

$y_2 = a_2x + b_2$ (3)

Για να τέμνονται οι δύο ευθείες, πρέπει να υπάρχει ένα σημείο x, όπου οι τιμές y_1 και y_2 είναι ίσες. Αυτό φαίνεται και στην Εικόνα 12.



Εικόνα 12 - Ευθείες που τέμνονται.

Στο σημείο που τέμνονται η κόκκινη και η μπλε ευθεία, οι τιμές του x και του y είναι ίσες και για τις δύο. Αυτό σημαίνει ότι:

$y_1 = y_2, \Rightarrow \ a_1x + b_1 = a_2x + b_2$

Επιλύοντας ως προς x, παίρνουμε:

$(a_1 - a_2)\ x = (b_2 - b_1) \ \Rightarrow$ (4)

$x = (b_2 - b_1) / (a_1 - a_2)$ (5)

Γράψτε ένα πρόγραμμα που θα διαβάζει τις τέσσερεις αριθμητικές σταθερές (a_1, a_2, b_1, b_2) και θα τις περνάει σε μια συνάρτηση.

Η συνάρτηση να υπολογίζει την τιμή του x στο οποίο τέμνονται οι δύο ευθείες (χρησιμοποιώντας την εξίσωση 5) και μετά, χρησιμοποιώντας τον τύπο της μιας από τις δύο, να υπολογίζει και το y και να τα τυπώνει.

Δώστε προσοχή στην περίπτωση που $a_1 = -a_2$, που σημαίνει ότι οι δύο ευθείες είναι παράλληλες.

3. Γράψτε ένα πρόγραμμα που θα διαβάζει τις συντεταγμένες x και y του κέντρου και την ακτίνα δύο κύκλων και θα καλεί μια συνάρτηση που θα υπολογίζει αν οι κύκλοι τέμνονται, αν δεν τέμνονται λόγω απόστασης ή αν δεν τέμνονται επειδή ο ένας περιέχεται μέσα στον άλλο.

Η απόσταση των δύο κέντρων των κύκλων είναι η υποτείνουσα ενός ορθογωνίου τριγώνου και μπορεί εύκολα να υπολογιστεί.

Χρειάζεται να βρείτε τρεις συνθήκες:

- Τη συνθήκη για να περιέχεται πλήρως ο ένας κύκλος μέσα στον άλλο.
- Τη συνθήκη για να τέμνονται.
- Τη συνθήκη ώστε ούτε να τέμνονται, ούτε να περιέχεται ο ένας κύκλος μέσα στον άλλο.

4. Εργάζεστε σε ένα αστεροσκοπείο και ένα από τα καθήκοντά σας είναι να μετράτε την απόσταση Γης – Σελήνης. Αυτό γίνεται με ένα όργανο που στέλνει μια δέσμη laser στη Σελήνη, πετυχαίνοντας τα σημεία που έχουν τοποθετηθεί ανακλαστήρες. Το όργανο μετράει το χρόνο από την εκπομπή της δέσμης laser μέχρι τη λήψη της ανάκλασης.

Επειδή η απόσταση που μετριέται με αυτόν τον τρόπο είναι η απόσταση ανάμεσα στις επιφάνειες της Γης και της Σελήνης, πρέπει να λάβετε υπόψη σας και τις ακτίνες των δύο σωμάτων:

Ισημερινή ακτίνα Γης: 6378.137km

Ισημερινή ακτίνα Σελήνης: 1738.1km

Η ταχύτητα του φωτός στο κενό είναι ακριβώς 299792.458km.

Για να μπορέσετε να απορρίψετε τυχόν λάθος σήματα από θόρυβο, η μέγιστη απόσταση Γης – Σελήνης είναι 406719.97km και η ελάχιστη 356352.93km.

Γράψτε ένα πρόγραμμα που θα ζητάει τη μέτρηση του οργάνου και θα καλεί μια συνάρτηση η οποία θα υπολογίζει τη στιγμιαία απόσταση Γης – Σελήνης και θα την επιστρέφει.

5. Η εξίσωση δεύτερου βαθμού:

$ax^2 + bx + c = 0$

μπορεί να έχει από καμία έως δύο λύσεις.

Αυτό καθορίζεται από την *διακρίνουσα*:

$\Delta = b^2 - 4ac$.

Αν $\Delta < 0$, τότε η εξίσωση δεν έχει λύση.

Αν $\Delta = 0$, τότε έχει μία λύση.

Αν $\Delta > 0$, τότε έχει δύο λύσεις.

Οι λύσεις δίνονται από τον τύπο:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Η ποσότητα κάτω από την τετραγωνική ρίζα ονομάζεται διακρίνουσα.

Γράψτε ένα πρόγραμμα που θα ζητάει τις τρεις αριθμητικές σταθερές (a, b και c) και θα καλεί μια συνάρτηση που θα υπολογίζει και θα τυπώνει τις λύσεις.

ΚΕΦΑΛΑΙΟ 3: ΛΙΣΤΕΣ, ΒΙΒΛΙΟΘΗΚΕΣ ΚΑΙ ΤΜΗΜΑΤΑ

Στην Python, εκτός από τους βασικούς τύπους δεδομένων (int, float, str, bool), υπάρχουν και οι σύνθετοι τύποι δεδομένων (composite data types) που αντιστοιχούν σε δομές και επιτρέπουν την αποθήκευση και οργάνωση πολλαπλών τιμών σε ένα ενιαίο αντικείμενο. Οι σύνθετοι τύποι δεδομένων ομαδοποιούν δεδομένα διαφορετικών τύπων (π.χ. αριθμούς, συμβολοσειρές, ακόμα και άλλα σύνθετα αντικείμενα), διευκολύνοντας τη διαχείριση και επεξεργασία πολύπλοκων πληροφοριών. Οι κύριοι σύνθετοι τύποι είναι:

- Λίστες (Lists): Μεταβλητές συλλογές στοιχείων, που τροποποιούνται δυναμικά
- Πλειάδες (Tuples): Αμετάβλητες συλλογές στοιχείων προσφέροντας ασφάλεια δεδομένων.
- Σύνολα (Sets): Μοναδικά στοιχεία χωρίς σειρά
- Λεξικά (Dictionaries): Ζεύγη κλειδιού-τιμής, επιτρέποντας την αντιστοίχιση κλειδιών με τιμές για γρήγορη αναζήτηση.

Παράδειγμα Σύνθετων Τύπων:

Τύπος	Σύνταξη	Χαρακτηριστικό
Λίστες (Lists)	[1, 2, 3]	Διατεταγμένη, αλλάζει (mutable).
Πλειάδες (Tuples)	(1, 2, 3)	Διατεταγμένη, δεν αλλάζει (immutable).
Λεξικά (Dictionaries)	{"key": "value"}	Διατηρεί σειρά εισαγωγής (Python 3.7+), ζεύγη κλειδιών-τιμών.
Σύνολα (Sets)	{1, 2, 3}	Μη διατεταγμένη, μόνο μοναδικά στοιχεία.

Στο κεφάλαιο αυτό εξετάζονται οι λίστες (lists), ο πιο ευέλικτος και ευρέως χρησιμοποιούμενος σύνθετος τύπος, καλύπτοντας σε βάθος τη δημιουργία, τη δεικτοδότηση, τη διάσχιση, τα τμήματα (slices), την αντιγραφή, καθώς και τους πολυδιάστατους πίνακες. Οι υπόλοιπες δομές δεδομένων πλειάδες (tuples), σύνολα (sets) και λεξικά (dictionaries), θα εξεταστούν με λεπτομέρεια στο επόμενο Κεφάλαιο.

Πέρα από τις δομές δεδομένων, ένα βασικό χαρακτηριστικό της Python είναι το πλούσιο οικοσύστημα βιβλιοθηκών (libraries) που επεκτείνει τις δυνατότητές της. Στο κεφάλαιο αυτό παρουσιάζεται ο τρόπος εγκατάστασης εξωτερικών βιβλιοθηκών μέσω του διαχειριστή πακέτων pip

και του αποθετηρίου PyPI (Python Package Index), παρέχοντας στον προγραμματιστή πρόσβαση σε χιλιάδες έτοιμα εργαλεία για κάθε είδους εφαρμογή.

Τέλος, αναλύεται η οργάνωση του κώδικα σε τμήματα (modules) και πακέτα (packages), μια πρακτική απαραίτητη καθώς τα προγράμματα μεγαλώνουν σε μέγεθος και πολυπλοκότητα. Η σωστή δόμηση του κώδικα σε ξεχωριστά αρχεία βελτιώνει την αναγνωσιμότητα, τη συντηρησιμότητα και την επαναχρησιμοποίηση. Στο πλαίσιο αυτό, εξηγείται ο ρόλος της μεταβλητής __name__ η οποία επιτρέπει τον έλεγχο αν ένα αρχείο εκτελείται ως αυτόνομο πρόγραμμα ή εισάγεται ως module σε άλλο αρχείο.

3.1. Λίστες

Οι λίστες είναι ο πιο ευέλικτος και συχνά χρησιμοποιούμενος σύνθετος τύπος δεδομένων στην Python. Η δομή της λίστας ομαδοποιεί μια ακολουθία στοιχείων, είτε αυτά είναι αριθμοί, είτε συμβολοσειρές, είτε και οποιουδήποτε άλλου τύπου, είτε συνδυασμός τύπων δίνοντας την δυνατότητα τα στοιχεία αυτά να τροποποιηθούν χωρίς περιορισμούς.

Σε αντίθεση με άλλες γλώσσες προγραμματισμού (όπως η C++ ή η Java), όπου ένας πίνακας δεσμεύει συνεχόμενο χώρο για συγκεκριμένο τύπο δεδομένων, οι λίστες στην Python αποθηκεύουν **αναφορές (references)** σε αντικείμενα.

Αυτό σημαίνει ότι:

1. Τα στοιχεία της λίστας μπορεί να είναι διασκορπισμένα στη μνήμη.
2. Η ίδια η λίστα περιέχει μια σειρά από δείκτες (pointers) που οδηγούν στις διευθύνσεις αυτών των αντικειμένων.
3. Αυτή η δομή επιτρέπει τη συνύπαρξη διαφορετικών τύπων (π.χ. [1, "hello", 3.14]) στην ίδια συλλογή.

Τα βασικά χαρακτηριστικά της λίστας ως δομή δεδομένων είναι τα εξής:

- **Συλλογή Στοιχείων (Collection)**: χρησιμοποιείται για την αποθήκευση πολλαπλών δεδομένων ταυτόχρονα
- **Διάταξη/Σειρά (Ordering**): είναι διατεταγμένη, δηλαδή τα στοιχεία έχουν μια καθορισμένη σειρά (ακολουθία). Τα στοιχεία διατηρούν τη σειρά με την οποία προστέθηκαν. Τα νέα στοιχεία που προστίθενται, τοποθετούνται στο τέλος της λίστας.
- **Μεταβλητότητα (Mutability):** είναι μεταβλητή (mutable), δηλαδή τα στοιχεία της μπορούν να τροποποιηθούν μετά τη δημιουργία της λίστας.

- **Ετερογένεια (Heterogeneity)**: τα δεδομένα της μπορούν να είναι ετερογενή, δηλαδή πολλών διαφορετικών τύπων χωρίς περιορισμό.
- **Δυναμικό Μέγεθος:** Οι λίστες μπορούν να αυξομειώνονται δυναμικά. Δεν χρειάζεται να ορίσουμε το μέγεθος κατά τη δημιουργία.
- **Επιτρέπονται Διπλότυπα:** Μια λίστα μπορεί να έχει το ίδιο στοιχείο πολλές φορές.

3.1.1. Δημιουργία Λίστας

Στην Python υπάρχουν διάφοροι τρόποι δημιουργίας λίστας, ανάλογα με τις ανάγκες μας. Ο καθένας εξυπηρετεί διαφορετικό σκοπό και έχει διαφορετική υπολογιστική επιβάρυνση.

3.1.1.1 Δημιουργία με αγκύλες [ ] - Literal Notation

Ο πιο απλός και συνηθισμένος τρόπος είναι η δημιουργία λίστας με αγκύλες.

```
new_list = [ element1, element2, …,element ]
```

Παραδείγματα δημιουργίας λίστας με αγκύλες

```
# Κενή λίστα
empty_list = []

# Λίστα με αρχικά στοιχεία
numbers = [1, 2, 3, 4, 5]
fruits = ["μήλο", "πορτοκάλι", "μπανάνα"]

# Μικτά δεδομένα
mixed = [1, "hello", 3.14, True, None]

# Εμφωλευμένες λίστες
nested = [[1, 2], [3, 4], [5, 6]]

# Με υπολογισμούς
calculated = [2+3, 10*5, 100//3]
print(calculated)  # [5, 50, 33]
```

Κ. 37: Δημιουργία Λίστας με αγκύλες

3.1.1.2 Δημιουργία με τον κατασκευαστή (constructor) list()

Ο κατασκευαστής list() είναι μια ενσωματωμένη συνάρτηση που δημιουργεί ένα νέο αντικείμενο λίστας. Ο κατασκευαστής list() μπορεί να δημιουργήσει λίστα από άλλες δομές δεδομένων.

```
new_list = list(iterable)
```

Ως **iterable** (επαναλήψιμο αντικείμενο) ορίζεται οποιαδήποτε δομή δεδομένων επιτρέπει τη διάσχιση των στοιχείων της, όπως συμβολοσειρές, πλειάδες, σύνολα, λεξικά.

Μετατροπή από Διαφορετικές Δομές:

- **Από Συμβολοσειρές (Strings):** Όταν μια συμβολοσειρά εισάγεται στον κατασκευαστή `list()`, η Python την αντιμετωπίζει ως μια ακολουθία χαρακτήρων. Το αποτέλεσμα είναι μια λίστα όπου κάθε στοιχείο είναι ένας μεμονωμένος χαρακτήρας (συμπεριλαμβανομένων των κενών διαστημάτων και των σημείων στίξης).

  Θεωρητικό Παράδειγμα: Η λέξη `"PYTHON"` μετατρέπεται στην ακολουθία `['P', 'Y', 'T', 'H','O', 'N']`.

- **Από Πλειάδες (Tuples):** Οι πλειάδες είναι αμετάβλητες (immutable). Η μετατροπή τους σε λίστα μέσω της `list()` είναι απαραίτητη όταν οι προγραμματιστές επιθυμούν να αποκτήσουν τη δυνατότητα τροποποίησης των δεδομένων που αρχικά ήταν "κλειδωμένα".

  Λειτουργία: Δημιουργείται ένα νέο, μεταλλαξιμό αντίγραφο των δεδομένων της πλειάδας.

  Θεωρητικό Παράδειγμα:

```
# Η πλειάδα περιέχει σταθερές συντεταγμένες
point_tuple = (10, 20, 30)

# Μετατροπή σε λίστα
point_list = list(point_tuple)
print(point_list) # [10, 20, 30]
```

Κ 38: Δημιουργίας Λίστας από Πλειάδα με τον κατασκευαστή list()

- **Από Σύνολα (Sets):** Τα σύνολα είναι μη διατεταγμένες συλλογές (unordered) μοναδικών στοιχείων. Με τη μετατροπή ενός συνόλου σε λίστα, επιβάλλεται μια συγκεκριμένη σειρά στα στοιχεία (βάσει της τρέχουσας διάταξης στη μνήμη), επιτρέποντας πλέον την πρόσβαση στα στοιχεία μέσω δεικτών (`index`). Όπως θα αναφέρεται στο επόμενο κεφάλαιο, στα σύνολα δεν είναι εφικτή η πρόσβαση των στοιχείων μέσω δεικτών.

  Θεωρητικό Παράδειγμα:

```
# Ένα σύνολο με μοναδικούς κωδικούς
codes_set = {101, 102, 103, 104}

# Μετατροπή σε λίστα για πρόσβαση με δείκτη
codes_list = list(codes_set)
```

```
print(codes_list) # [101, 102, 103, 104]
```

Κ 39: Δημιουργίας Λίστας από Σύνολο με τον κατασκευαστή list()

- **Από Λεξικά (Dictionaries):** Η μετατροπή ενός λεξικού παρουσιάζει ιδιαίτερο ενδιαφέρον:
    - Προεπιλογή: Η list(my_dict) επιστρέφει μια λίστα που περιέχει μόνο τα κλειδιά (keys) του λεξικού.
    - Τιμές: Για τη λήψη των τιμών απαιτείται η μέθοδος .values() (π.χ. list(my_dict.values())).
    - Ζεύγη: Για τη λήψη ζευγών (κλειδί-τιμή) χρησιμοποιείται η μέθοδος dict.items() (π.χ. list(my_dict.items())).

  Θεωρητικό Παράδειγμα:

```
inventory = {'Μήλα': 50, 'Μπανάνες': 30, 'Κεράσια': 20}

# 1. Μόνο τα Κλειδιά (Προεπιλεγμένη λειτουργία)
keys_list = list(inventory)
print(keys_list) # ['Μήλα', 'Μπανάνες', 'Κεράσια']

# 2. Μόνο οι Τιμές
values_list = list(inventory.values())
print(values_list) # [50, 30, 20]

# 3. Ζεύγη Κλειδιού-Τιμής (ως λίστα από πλειάδες)
items_list = list(inventory.items())
print(items_list) # [('Μήλα', 50), ('Μπανάνες', 30), ('Κεράσια', 20)]
```

Κ 40: Δημιουργίας Λίστας από Λεξικό με τον κατασκευαστή list()

3.1.1.3 Δημιουργία λίστας με την range()

Η συνάρτηση `range()` δημιουργεί ακολουθίες αριθμών. Δέχεται κατά την κλήση της **έως** τρεις παραμέτρους:

`range( start, stop, step )`

- **start (Αρχή):** Η τιμή από την οποία ξεκινά η ακολουθία (συμπεριλαμβάνεται). Προεπιλογή: **0**.
- **stop (Τέλος):** Η τιμή στην οποία σταματά η ακολουθία (**δεν συμπεριλαμβάνεται**). Η ακολουθία φτάνει μέχρι το stop - 1.
- **step (Βήμα):** Η απόσταση μεταξύ των αριθμών. Μπορεί να είναι θετική (αύξουσα) ή αρνητική (φθίνουσα). Προεπιλογή: **1**.

Περιπτώσεις Χρήσης

- **Μονή Παράμετρος range(stop):** Δημιουργεί ακολουθία από το 0 έως το stop - 1.

  Παράδειγμα: list(range(5)) -> [0, 1, 2, 3, 4]

- **Διπλή Παράμετρος range(start, stop:** Ορίζει ρητά το σημείο εκκίνησης.

  Παράδειγμα: list(range(10, 15)) -> [10, 11, 12, 13, 14]

- **Τριπλή Παράμετρος range(start, stop, step):** Επιτρέπει τη δημιουργία ειδικών ακολουθιών (π.χ. μόνο ζυγοί αριθμοί).

  Παράδειγμα: list(range(0, 11, 2)) -> [0, 2, 4, 6, 8, 10]

- **Αντίστροφη Ακολουθία (Negative Step):** Απαιτεί το start να είναι μεγαλύτερο από το stop.

  Παράδειγμα: list(range(5, 0, -1)) -> [5, 4, 3, 2, 1]

Παραδείγματα:

```
# range(stop) - από 0 έως stop-1
numbers = list(range(10))
print(numbers)  # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# range(start, stop) - από start έως stop-1
numbers = list(range(1, 11))
print(numbers)  # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# range(start, stop, step) - με βήμα
evens = list(range(0, 20, 2))
print(evens)  # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

odds = list(range(1, 20, 2))
print(odds)  # [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

# Αντίστροφη σειρά
countdown = list(range(10, 0, -1))
print(countdown)  # [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

# Αρνητικοί αριθμοί
negatives = list(range(-5, 5))
print(negatives)  # [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
```

Κ 41: Δημιουργίας λίστας με την range()

3.1.1.4 Δημιουργία λίστας με επανάληψη (πολλαπλασιασμός λίστας)

Η δημιουργία λίστας με επανάληψη (πολλαπλασιασμός λίστας) χρησιμοποιείται για τη δημιουργία λίστας με επαναλαμβανόμενες τιμές.

```
new_list = [element1,.., elementN] * K
```

Δημιουργείται νέα λίστα new_list με στοιχεία K φορές τα element1,..,element

Παραδείγματα

```
# Βασική επανάληψη
zeros = [0] * 5
print(zeros)  # [0, 0, 0, 0, 0]

# Με συμβολοσειρές - strings
greetings = ["Hello"] * 3
print(greetings)  # ['Hello', 'Hello', 'Hello']

# Με πολλαπλά στοιχεία
pattern = [1, 2, 3] * 3
print(pattern)  # [1, 2, 3, 1, 2, 3, 1, 2, 3]

# Αρχικοποίηση πίνακα
row = [0] * 10
print(row)  # [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Κ 42: Δημιουργία Λίστας με επαναλαμβανόμενες τιμές

Οι διαφορετικοί τρόποι δημιουργίας λίστας προσφέρουν ευελιξία και καθαρότερο κώδικα. Κάθε μία από αυτές τις μεθόδους εξυπηρετεί διαφορετικές ανάγκες.

3.1.1.5 Μήκος Λίστας len()

Για τον υπολογισμό των στοιχείων μιας λίστας, χρησιμοποιείται η συνάρτηση len().

```
len(list)
```

Παράδειγμα

```
thislist = ["μήλο", "μπανάνα", "κεράσι"]
print(len(thislist)) # 3
```

Κ 43 Παράδειγμα Εύρεσης του Μήκους μιας Λίστας

3.1.1.6 Τύπος Δεδομένων Λίστας type()

Οι λίστες στην Python ορίζονται ως αντικείμενα με τον τύπο δεδομένων:

```
<class 'list'>
```

Η συνάρτηση `type()` χρησιμοποιείται για εύρεση του τύπου μιας μεταβλητής:

```
type(variable)
```

Παράδειγμα

```
mylist = ["μήλο", "μπανάνα", "κεράσι"]
print(type(mylist)) # <class 'list'>
```

Κ 44 Παράδειγμα type() με τύπο δεδομένων <class 'list'>

3.1.1.7 Ασκήσεις Δημιουργίας Λίστας

Άσκηση 1: Βασικές Μέθοδοι Κατασκευής

- Να δημιουργηθεί μια λίστα με το όνομα `days` η οποία θα περιέχει τις ημέρες της εβδομάδας ως συμβολοσειρές, χρησιμοποιώντας τη μέθοδο των αγκυλών `[]`.
- Να δημιουργηθεί μια κενή λίστα με το όνομα `log_data` χρησιμοποιώντας τον constructor `list()`.
- Να μετατραπεί η λέξη "ALGORITHM" σε λίστα χαρακτήρων (π.χ. `[..'A', 'L',.]`) χρησιμοποιώντας τη συνάρτηση `list()`.

Άσκηση 2: Αρχικοποίηση με τον Τελεστή Πολλαπλασιασμού (*)

1. Να δημιουργηθεί μια λίστα `buffer` η οποία θα περιέχει **50 μηδενικά** (0).
2. Να δημιουργηθεί μια λίστα `status_codes` η οποία θα περιέχει 10 φορές την τιμή **"Pending"**.

Άσκηση 3: Χρήση της `range()`

Να δημιουργηθεί μια λίστα `multiples_of_five` που θα περιέχει όλους τους αριθμούς από το **0 έως το 100** (συμπεριλαμβανομένου) με βήμα 5, χρησιμοποιώντας τη συνάρτηση `range()`.

Άσκηση 4: Δυναμική Δημιουργία από Άλλες Δομές

Δίνεται η πλειάδα (tuple): `coords = (10.5, 20.8, 30.1)`. Να μετατραπεί σε λίστα ώστε να είναι δυνατή η μετέπειτα τροποποίηση των τιμών της, χρησιμοποιώντας τον constructor `list()`.

3.1.2. Πρόσβαση σε στοιχεία της Λίστας - Δεικτοδότηση Λιστών

Τα στοιχεία της λίστας είναι ευρετηριασμένα (indexed) σε συγκεκριμένη θέση και μπορούν να προσπελαστούν αναφερόμενοι στον αριθμό ευρετηρίου τους. Κάθε στοιχείο έχει μια μοναδική θέση που ονομάζεται **δείκτης** (index).

Το στοιχείο στη θέση k της λίστας (list) προσεγγίζεται ως `list[k]`.

Παράδειγμα:

```
mylist = ["μήλο", "μπανάνα", "κεράσι"]
print(mylist[1]) # μπανάνα
```

Κ 45 Παράδειγμα πρόσβασης στοιχείου λίστας στην θέση 1

Η δεικτοδότηση (indexing) είναι ο μηχανισμός με τον οποίο πραγματοποιείται η πρόσβαση σε μεμονωμένα στοιχεία μιας λίστας.

Σημαντικά Σημεία:

- Οι δείκτες ξεκινούν από το 0 (zero-based indexing)
- Το πρώτο στοιχείο είναι στη θέση 0
- Το τελευταίο στοιχείο είναι στη θέση `len(list) - 1`
- Υποστηρίζεται και αρνητική δεικτοδότηση

3.1.2.1 Θετική Δεικτοδότηση

Ξεκινά από το 0 για το πρώτο στοιχείο και φτάνει έως το n-1, όπου n είναι το μήκος της λίστας.

Παραδείγματα Θετικής Δεικτοδότησης

```
# Απεικόνιση δεικτών:
# Στοιχεία:  ["μήλο", "πορτοκάλι", "μπανάνα", "κεράσι", "ακτινίδιο"]
# Δείκτες:      0          1            2           3           4

# Πρόσβαση σε στοιχεία
print(fruits[0])  # "μήλο" - Πρώτο στοιχείο
print(fruits[1])  # "πορτοκάλι" - Δεύτερο στοιχείο
print(fruits[4])  # "ακτινίδιο" - Πέμπτο (τελευταίο) στοιχείο

# Τροποποίηση στοιχείου
fruits[1] = "λεμόνι"
print(fruits)  # ["μήλο", "λεμόνι", "μπανάνα", "κεράσι", "ακτινίδιο"]
```

Κ 46: Παραδείγματα Θετικής Δεικτοδότησης

3.1.2.2 Αρνητική Δεικτοδότηση

Ξεκινά από το -1 για το τελευταίο στοιχείο και φτάνει έως το -n για το πρώτο, όπου n είναι το μήκος της λίστας. Είναι εξαιρετικά χρήσιμη για την πρόσβαση σε στοιχεία που βρίσκονται στο τέλος της λίστας χωρίς να είναι γνωστό το ακριβές μήκος της.

Οπτική Αναπαράσταση

```
Στοιχεία:    [ 'P',  'y',  't',  'h',  'o',  'n' ]
Θετικός:       0     1     2     3     4     5
Αρνητικός:    -6    -5    -4    -3    -2    -1
```

Παραδείγματα Αρνητικής Δεικτοδότησης

```
# Απεικόνιση δεικτών:
# Θετικοί:       0          1            2           3           4
# Στοιχεία:  ["μήλο", "πορτοκάλι", "μπανάνα", "κεράσι", "ακτινίδιο"]
# Αρνητικοί:    -5         -4           -3          -2          -1

# Πρόσβαση με αρνητικούς δείκτες
print(fruits[-1])  # "ακτινίδιο" - Τελευταίο στοιχείο
print(fruits[-2])  # "κεράσι" - Προτελευταίο
print(fruits[-5])  # "μήλο" - Πέμπτο από το τέλος (πρώτο)

# Σχέση θετικών-αρνητικών:
# fruits[0] == fruits[-5]
# fruits[4] == fruits[-1]
# Γενικά: fruits[i] == fruits[i - len(fruits)]
```

Κ 47: Παραδείγματα Αρνητικής Δεικτοδότησης

Η Αρνητική Δεικτοδότηση χρησιμοποιείται για την πρόσβαση σε στοιχεία που βρίσκονται στο τέλος της λίστας χωρίς να είναι γνωστό το ακριβές μήκος της.

```
data = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Τελευταίο
last = data[-1]

# Προτελευταίο
second_last = data[-2]

# Τα 3 τελευταία στοιχεία
last_three = data[-3:]  # [80, 90, 100]
```

Κ 48: Παραδείγματα Αρνητικής Δεικτοδότησης για την πρόσβαση σε στοιχεία που βρίσκονται στο τέλος της λίστας χωρίς να είναι γνωστό το ακριβές μήκος της.

3.1.2.3 Τμήματα Λιστών (Slicing)

Ο τεμαχισμός (Slicing) επιτρέπει την εξαγωγή ενός τμήματος (sub-list) από την αρχική λίστα. Το αποτέλεσμα θα είναι μια νέα λίστα με τα καθορισμένα στοιχεία.

Η σύνταξη είναι:

```
list[start : stop : step]
```

όπου:

- start: Ο δείκτης έναρξης (συμπεριλαμβάνεται). Αν δεν υπάρχει, χρησιμοποιείται ως προεπιλογή το 0.
- stop: Ο δείκτης λήξης (δεν συμπεριλαμβάνεται). Η εξαγωγή σταματά στο stop-1.
- step: Το βήμα της διάσχισης. Μπορεί να είναι θετικό ή αρνητικό. Αν δεν υπάρχει, χρησιμοποιείται ως προεπιλογή το 1.

Παραδείγματα

```
nums = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
#       0   1   2   3   4   5   6   7   8   9

# 1. Βασικό Slice (από θέση 2 έως 5-1=4)
print(nums[2:5])    # [20, 30, 40]

# 2. Παράλειψη start (από την αρχή)
print(nums[:4])     # [0, 10, 20, 30]

# 3. Παράλειψη stop (μέχρι το τέλος)
print(nums[7:])     # [70, 80, 90]

# 4. Χρήση βήματος (Step)
print(nums[1:8:2]) # [10, 30, 50, 70] (κάθε δεύτερο στοιχείο)
```

Κ 49: Παραδείγματα Τεμαχισμού (Slicing)

Στον παρακάτω πίνακα αναφέρονται παραδείγματα διαφορετικής σύνταξης:

Σύνταξη	Περιγραφή	Παράδειγμα	Αποτέλεσμα

[start:stop]	Τμήμα από start έως stop-1	lst[2:5]	[lst[2], lst[3], lst[4]]
[:stop]	Από αρχή έως stop-1	lst[:3]	[lst[0], lst[1], lst[2]]
[start:]	Από start έως τέλος	lst[5:]	Από θέση 5 έως τέλος
[:]	Αντιγραφή λίστας	lst[:]	Όλη η λίστα
[::step]	Όλη με βήμα	lst[::2]	Κάθε 2ο στοιχείο
[::-1]	Αντιστροφή	lst[::-1]	Αντίστροφα όλα τα στοιχεία
[-n:]	Τα n τελευταία	lst[-3:]	Τελευταία 3
[:-n]	Όλα εκτός n τελευταίων	lst[:-2]	Χωρίς τα τελευταία 2
[start:stop:step]	Πλήρης έλεγχος	lst[1:8:2]	[lst[1], lst[3], lst[5], lst[7]]

3.1.2.4 Σφάλματα και Περιορισμοί

3.1.2.4.1 IndexError

Στην προσπέλαση δείκτη που δεν υπάρχει (εκτός ορίων), η Python εγείρει ένα `IndexError`.

```
L = [1, 2, 3]
print(L[5])
```

Κ 50: Παράδειγμα προσπέλασης σε δείκτη λίστας που δεν υπάρχει: IndexError

```
Exception has occurred: IndexError ×
list index out of range
  File "C:\Users\maria\Downloads\test.py", line 44, in <module>
    print(L[5])
          ~^^^
IndexError: list index out of range
```

Κ 51: IndexError Exception στη προσπέλαση δείκτη που δεν υπάρχει (εκτός ορίων)

3.1.2.5 Τεμαχισμός (Slicing) και εκτός ορίων (Out of Range)

Ο τεμαχισμός (slicing) δεν προκαλεί σφάλμα αν οι δείκτες είναι εκτός ορίων. Η Python επιστρέφει ό,τι μπορεί να βρει.

```
L = [1, 2, 3]
print(L[1:100]) # Έξοδος: [2, 3] (Δεν χτυπάει σφάλμα)
```

Κ 52: Παράδειγμα Τεμαχισμού (Slicing) εκτός ορίων (Out of Range)

3.1.2.6 Ασκήσεις Δεικτοδότησης

Άσκηση 5: Αρνητική Δεικτοδότηση (Negative Indexing)

Έστω η λίστα με το ιστορικό περιήγησης ενός χρήστη (από το παλαιότερο στο νεότερο): `web_history = ["google.com", "wikipedia.org", "github.com", "python.org", "stackoverflow.com"]`

Ζητούνται τα εξής:

1. Να ανακτηθεί η **τελευταία** σελίδα που επισκέφθηκε ο χρήστης χρησιμοποιώντας αρνητικό δείκτη.
2. Να ανακτηθεί η **τρίτη σελίδα από το τέλος** (δηλ. το github.com).
3. Να δημιουργηθεί ένα τμήμα λίστας που θα περιέχει τις **δύο τελευταίες** σελίδες του ιστορικού, χρησιμοποιώντας αποκλειστικά αρνητικούς δείκτες.
4. Να εξηγηθεί τι θα επιστρέψει η εντολή `web_history[-10: ]` και γιατί δεν προκαλείται σφάλμα `(IndexError)`.

Άσκηση 6: Τμήματα Λιστών (Slicing)

Έστω η λίστα: numbers = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Ζητούνται τα εξής:

1. Να δημιουργηθεί μια νέα λίστα mid_section που θα περιέχει τους αριθμούς από το **30 έως και το 70**.
2. Να εξαχθούν τα **πρώτα τέσσερα** στοιχεία της λίστας με μία μόνο εντολή slicing.
3. Να δημιουργηθεί μια λίστα steps που θα περιέχει **κάθε δεύτερο στοιχείο** ολόκληρης της αρχικής λίστας (δηλ. 0, 20, 40...).
4. Να αντιστραφεί ολόκληρη η λίστα numbers με τη χρήση της παραμέτρου step.

3.1.3. Τροποποίηση στοιχείων λίστας

Η τροποποίηση ή αλλαγή τιμής ενός συγκεκριμένου στοιχείου της λίστας στην θέση K, πραγματοποιείται με την ανάθεση νέας τιμής στο στοιχείο της λίστας:

```
list[K] = new_value
```

Παράδειγμα

```
thislist = ["μήλο", "μπανάνα", "κεράσι"]
thislist[1] = "αχλάδι"
print(thislist) # ['μήλο', 'αχλάδι', 'κεράσι']
```

Κ 53: Τροποποίηση τιμής σε στοιχείο της λίστας

3.1.3.1 Τροποποίηση μέσω Δεικτοδότησης (Mutability)

Είναι δυνατή η τροποποίηση των στοιχείων της λίστας χρησιμοποιώντας τους δείκτες. Η ανάθεση σε τμήμα της λίστας, για παράδειγμα `L[a:b] = new_list`, μπορεί να αλλάξει το **μήκος** της λίστας, καθώς η Python αντικαθιστά ολόκληρο το τμήμα με τα νέα στοιχεία, ακόμη και αν αυτά είναι περισσότερα ή λιγότερα από τα αρχικά.

Παραδείγματα

```
data = [10, 20, 30, 40]

# Αλλαγή μεμονωμένου στοιχείου
data[1] = 99

# Μαζική αλλαγή μέσω Slicing
data[2:] = [77, 88]

print(data) # [10, 99, 77, 88]

# Αλλαγή μέσω Slicing
data[1:2] = [12, 13, 14]

print(data) # [10, 12, 13, 14, 77, 88]
```

Κ 54: Παραδείγματα Τροποποίησης Λίστας μέσω Δεικτοδότησης

3.1.3.2 Ασκήσεις Τροποποίησης Στοιχείων Λίστας

Άσκηση 7: Ενημέρωση Μεμονωμένων Στοιχείων (Indexing)

Έστω η λίστα τιμών: `prices = [10.50, 25.00, 5.75, 40.00, 15.20]`

Ζητούνται τα εξής:

1. Λόγω αύξησης του κόστους, η τιμή του δεύτερου προϊόντος (δείκτης 1) πρέπει να αλλάξει σε 30.00.

2. Η τελευταία τιμή της λίστας (χρήση αρνητικού δείκτη) πρέπει να μειωθεί κατά 2.00 μονάδες (δηλαδή η νέα τιμή να προκύψει από την αφαίρεση του 2 από την παλιά).
3. Να εκτυπωθεί η λίστα μετά από κάθε αλλαγή για την επιβεβαίωση των αποτελεσμάτων.

Άσκηση 8: Μαζική Τροποποίηση μέσω Τμημάτων (Slicing Assignment)

Έστω μια λίστα που καταγράφει τις ημέρες εργασίας ενός υπαλλήλου σε ένα δεκαήμερο: `schedule = ["Εργασία", "Εργασία", "Ρεπό", "Εργασία", "Εργασία", "Εργασία", "Εργασία", "Ρεπό", "Εργασία", "Εργασία"]`

Ζητούνται τα εξής:

1. Ο υπάλληλος αποφάσισε να πάρει άδεια για τρεις συνεχόμενες ημέρες, από την 4η έως και την 6η ημέρα (δείκτες 3, 4 και 5). Να αντικατασταθούν αυτά τα στοιχεία με τη λέξη **"Άδεια"** χρησιμοποιώντας μία μόνο εντολή slicing.
2. Να αντικατασταθούν τα δύο πρώτα στοιχεία της λίστας με τις τιμές **"Τηλεργασία"** και **"Τηλεργασία"**.
3. Να εξηγηθεί τι θα συμβεί αν στο τμήμα `schedule[0:2]` ανατεθεί μια λίστα με τρία στοιχεία αντί για δύο (π.χ. `["Α", "Β", "Γ"]`).

3.1.4. Λειτουργίες και Μέθοδοι Λιστών

Οι λίστες υποστηρίζουν μια πληθώρα ενσωματωμένων λειτουργιών και μεθόδων που διευκολύνουν τη δυναμική διαχείριση των δεδομένων.

3.1.4.1 Βασικές Λειτουργίες (Built-in Functions & Operators)

Οι λειτουργίες αυτές εκτελούνται απευθείας πάνω στη δομή της λίστας:

- **len(list)**: Επιστρέφει το πλήθος των στοιχείων που περιέχει η λίστα (παράγραφος 3.1.1.5).
- **del list[index] ή del list[start:stop], ή del list:** οριστική διαγραφή στοιχείων λίστας:
    - **del list[index]:** Οριστική διαγραφή του στοιχείου στη συγκεκριμένη θέση index

```
# Διαγραφή συγκεκριμένης θέσης
colors = ["κόκκινο", "μπλε", "πράσινο", "κίτρινο", "μαύρο"]
del colors[1]  # Διαγράφει το "μπλε"
print(colors)  # ['κόκκινο', 'πράσινο', 'κίτρινο', 'μαύρο']

# Διαγραφή με αρνητικό δείκτη
del colors[-1]  # Διαγράφει το τελευταίο
```

```
print(colors)  # ['κόκκινο', 'πράσινο', 'κίτρινο']
```

Κ. 55: Λειτουργία Διαγραφής του στοιχείου στη θέση index με την del list[index]

- **del list[start:stop]**: Οριστική διαγραφή των στοιχείων σε ολόκληρα τμήματα της λίστας από start σε stop.

```
# Διαγραφή τμήματος (slice)
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
del numbers[3:6]  # Διαγράφει θέσεις 3, 4, 5
print(numbers)  # [0, 1, 2, 6, 7, 8, 9]
```

Κ 56: Λειτουργία Διαγραφής τμήματος λίστας με την del list[start:stop]

- **del list**: Οριστική διαγραφή **ολόκληρης της μεταβλητής** — η μεταβλητή παύει να υπάρχει και κάθε προσπάθεια πρόσβασης σε αυτή θα δώσει `NameError`..

```
my_list = [1, 2, 3]
del my_list
print(my_list)  # NameError: name 'my_list' is not defined
```

Κ 57: Λειτουργία Διαγραφής της μεταβλητής list με την del list

- **in / not in**: Ελέγχεται η ύπαρξη (ή μη) ενός στοιχείου στη λίστα, επιστρέφοντας λογική τιμή (True/False).

```
# Βασική χρήση του in και not in
fruits = ["μήλο", "πορτοκάλι", "μπανάνα", "κεράσι"]

if "μήλο" in fruits:
    print("Το μήλο υπάρχει στη λίστα")  # Θα εκτυπωθεί

if "αχλάδι" not in fruits:
    print("Το αχλάδι δεν υπάρχει")  # Θα εκτυπωθεί
```

Κ 58: έλεγχος ύπαρξης στοιχείου σε λίστα με τις in/not in -

- **max(list), min(list), sum(list)** - Στατιστικές Λειτουργίες: Αυτές οι συναρτήσεις εκτελούν βασικούς υπολογισμούς σε λίστες.
    - **max(list)**: επιστρέφει το μέγιστο στοιχείο της λίστας
    - **min(list)**: επιστρέφει το ελάχιστο στοιχείο της λίστας
    - **sum(list)**: επιστρέφει το άθροισμα των στοιχείων της λίστας

```
numbers = [45, 12, 78, 23, 67, 34, 89, 56]

# Μέγιστο στοιχείο
maximum = max(numbers)
```

```python
print(f"Μέγιστο: {maximum}")  # 89

# Ελάχιστο στοιχείο
minimum = min(numbers)
print(f"Ελάχιστο: {minimum}")  # 12

# Άθροισμα
total = sum(numbers)
print(f"Άθροισμα: {total}")  # 404

# Μέσος όρος
average = sum(numbers) / len(numbers)
print(f"Μέσος όρος: {average:.2f}")  # 50.50
```

Κ 59 : Χρήση συναρτήσεων max(), min(), sum()

- **sorted(list) – Ταξινόμηση στοιχείων λίστας (Επιστρέφει Νέα Λίστα)**: Η συνάρτηση sorted() δημιουργεί και επιστρέφει μια νέα ταξινομημένη λίστα χωρίς να τροποποιεί την αρχική.

```python
numbers = [45, 12, 78, 23, 67]

# Αύξουσα ταξινόμηση
sorted_numbers = sorted(numbers)
print(sorted_numbers)  # [12, 23, 45, 67, 78]
print(numbers)         # [45, 12, 78, 23, 67] - Η αρχική δεν άλλαξε!

# Φθίνουσα ταξινόμηση
reverse_sorted = sorted(numbers, reverse=True)
print(reverse_sorted)  # [78, 67, 45, 23, 12]

# Με strings - αλφαβητική
names = ["Ζωή", "Άννα", "Μαρία", "Βασίλης"]
sorted_names = sorted(names)
print(sorted_names)  # ['Άννα', 'Βασίλης', 'Ζωή', 'Μαρία']
```

Κ 60: sorted() - Ταξινόμηση Λίστας

3.1.4.2 Μέθοδοι Λιστών

Οι μέθοδοι είναι συναρτήσεις που ανήκουν στα αντικείμενα λίστας και καλούνται με τη σύνταξη:

```
list.method()
```

3.1.4.2.1 Προσθήκη Στοιχείων

Μέθοδος append(element) - Προσθήκη στοιχείου στο τέλος: προσθέτει το όρισμα element στο τέλος της λίστας.

```python
# Βασική χρήση
fruits = ["μήλο", "πορτοκάλι"]
fruits.append("μπανάνα")
print(fruits)  # ['μήλο', 'πορτοκάλι', 'μπανάνα']

# Προσθήκη διαφόρων τύπων
data = [1, 2, 3]
data.append(4)
data.append("hello")
data.append([5, 6])  # Προσθέτει τη λίστα ως ΕΝΑ στοιχείο
print(data)  # [1, 2, 3, 4, 'hello', [5, 6]]
```

Κ 61: append: προσθήκη στοιχείου στο τέλος της λίστας

Μέθοδος extend(seq) - Προσθήκη Πολλαπλών Στοιχείων: Η μέθοδος extend() προσθέτει τα στοιχεία seq στο τέλος της λίστας. Το όρισμα seq μπορεί να είναι λίστα-list, πλειάδα-tuple, αλφαριθμητικά-strings. Στην περίπτωση των αλφαριθμητικών αν είχαμε την λίστα thelist με στοιχεία ['1', '2', '3'] η προσθήκη του αλφαριθμητικού '45', θα είχε ως αποτέλεσμα: ['1', '2', '3', '4', '5'].

```python
thelist = ['1', '2', '3']
thelist.extend('45')
print(thelist)  # ['1', '2', '3', '4', '5']
```

Παραδείγματα:

```python
# Βασική Χρήση
list2 = [1, 2, 3]
list2.extend([4, 5])
print(list2)  # [1, 2, 3, 4, 5] - Επίπεδη λίστα

# Διαφορά από append()
list1 = [1, 2, 3]
list1.append([4, 5])
print(list1)  # [1, 2, 3, [4, 5]] - Εμφωλευμένη λίστα

# Με άλλα iterables
# προσθήκη αλφαριθμητικού-string
letters = ['a', 'b']
letters.extend('cd')  # String
print(letters)  # ['a', 'b', 'c', 'd']

# προσθήκη πλειάδας
```

```
nums = [1, 2, 3, 4]
nums.extend((5, 6, 7))
print(nums) # [1, 2, 3, 4, 5, 6, 7]

# Range
numbers = [1, 2]
numbers.extend(range(3, 6))
print(numbers)  # [1, 2, 3, 4, 5]
```

Κ 62: extend: προσθήκη πολλαπλών στοιχείων στο τέλος της λίστας

Μέθοδος insert(index, element) - Προσθήκη σε Συγκεκριμένη Θέση: εισάγει το στοιχείο `element` στη συγκεκριμένη θέση `index` της λίστας.

```
# Βασική χρήση - insert(index, element)
fruits = ["μήλο", "μπανάνα", "κεράσι"]
fruits.insert(1, "πορτοκάλι")  # Εισαγωγή στη θέση 1
print(fruits)  # ['μήλο', 'πορτοκάλι', 'μπανάνα', 'κεράσι']

# Εισαγωγή στην αρχή
numbers = [2, 3, 4]
numbers.insert(0, 1)
print(numbers)  # [1, 2, 3, 4]
```

Κ 63: insert: προσθήκη στοιχείου σε συγκεκριμένη θέση της λίστας

Σημείωση: Προσθήκη στοιχείων σε λίστα: Για την προσθήκη στοιχείων σε λίστα χρησιμοποιούνται:

- append(element): προσθήκη στοιχείου-element στο τέλος της λίστας
- insert(index, element): προσθήκη στοιχείου-element σε συγκεκριμένη θέση-index της λίστας
- extend(seq): προσθήκη στοιχείων του ορίσματος seq στο τέλος της λίστας

Κρίσιμες Διαφορές

Μέθοδος	Λειτουργία	Αποτέλεσμα
append(x)	Προσθέτει το `x` ως ένα στοιχείο.	Η λίστα μεγαλώνει κατά 1.
extend(L)	Προσθέτει όλα τα στοιχεία της `L` ένα-ένα.	Η λίστα μεγαλώνει κατά `len(L)`.

3.1.4.2.2 Αφαίρεση Στοιχείων

Μέθοδος remove(element) - Διαγραφή Συγκεκριμένου Στοιχείου: Η μέθοδος remove() διαγράφει την πρώτη εμφάνιση του ορίσματος element από τη λίστα.

```
# Βασική χρήση
```

```python
fruits = ["μήλο", "πορτοκάλι", "μπανάνα", "πορτοκάλι"]
fruits.remove("πορτοκάλι")  # Διαγράφει το πρώτο "πορτοκάλι"
print(fruits)  # ['μήλο', 'μπανάνα', 'πορτοκάλι']

# ValueError αν το στοιχείο δεν υπάρχει
numbers = [1, 2, 3, 4, 5]
numbers.remove(10)  # ValueError: list.remove(x): x not in list

# Ασφαλής διαγραφή
item = 10
if item in numbers:
    numbers.remove(item)
else:
    print(f"{item} δεν υπάρχει στη λίστα")

# Διαγραφή όλων των εμφανίσεων
numbers = [1, 2, 3, 2, 4, 2, 5]
while 2 in numbers:
    numbers.remove(2)
print(numbers)  # [1, 3, 4, 5]
```

Κ 64: remove: Διαγραφή στοιχείου από την λίστα

Μέθοδος pop() ή pop(index) - Διαγραφή και Επιστροφή Στοιχείου: Η μέθοδος pop() διαγράφει από τη λίστα και επιστρέφει το στοιχείο της λίστας που διέγραψε.

- **pop(), χωρίς όρισμα**: διαγράφει το τελευταίο στοιχείο της λίστας και επιστρέφει το στοιχείο της λίστας που διέγραψε,
- **pop(index), με όρισμα index**: διαγράφει το στοιχείο στη συγκεκριμένη θέση-index της λίστας και επιστρέφει το στοιχείο της λίστας που διέγραψε.

Παραδείγματα

```python
# Χωρίς όρισμα - διαγράφει και επιστρέφει το τελευταίο
numbers = [10, 20, 30, 40, 50]
last = numbers.pop()
print(last)     # 50
print(numbers)  # [10, 20, 30, 40]

# Με όρισμα - διαγράφει από συγκεκριμένη θέση
numbers = [10, 20, 30, 40, 50]
second = numbers.pop(1)
print(second)   # 20
print(numbers)  # [10, 30, 40, 50]

# Stack (LIFO - Last In First Out)
```

```
stack = []
stack.append(1)  # Push
stack.append(2)
stack.append(3)
print(stack)      # [1, 2, 3]
print(stack.pop())  # 3 - Pop
print(stack.pop())  # 2
print(stack)      # [1]
```

Κ 65: pop: Διαγράφει και επιστρέφει το στοιχείο

Μέθοδος clear() - Διαγραφή Όλων: Η μέθοδος clear() αφαιρεί όλα τα στοιχεία από τη λίστα. Η λίστα παραμένει, αλλά χωρίς περιεχόμενο (κενή: [])

Παραδείγματα

```
# Βασική χρήση
numbers = [1, 2, 3, 4, 5]
numbers.clear()
print(numbers)  # []
print(len(numbers))  # 0

# Ισοδύναμοι τρόποι
# χρήση del
numbers = [1, 2, 3, 4, 5]
del numbers[:]
print(numbers)  # []
# ανάθεση κενής λίστας
numbers = [1, 2, 3, 4, 5]
numbers[:] = []
print(numbers)  # []

# Διαφορά από νέα ανάθεση
list1 = [1, 2, 3]
list2 = list1  # Αναφορά στο ίδιο αντικείμενο

# Με clear()
list1.clear()
print(list1)  # []
print(list2)  # [] - Επηρεάστηκε!

# Με νέα ανάθεση
list1 = [1, 2, 3]
list2 = list1
list1 = []  # Νέα λίστα
print(list1)  # []
```

```
print(list2)  # [1, 2, 3] - Δεν επηρεάστηκε
```

Κ 66: clear: Διαγραφή στοιχείων της λίστας

Σημείωση: Αφαίρεση στοιχείων από λίστα: Για την αφαίρεση στοιχείων από λίστα χρησιμοποιούνται:

- Μέθοδος remove(element): αφαίρεση στοιχείου-element της λίστας. Στην περίπτωση στοιχείων με τη ίδια τιμή, αφαιρείται η πρώτη εμφάνιση.
- Μέθοδος pop(index), pop():
  - pop(index), αφαίρεση του στοιχείου σε συγκεκριμένη θέση-index της λίστας.
  - pop(): αφαίρεση του τελευταίου στοιχείου της λίστας.
- Λειτουργία del list[index], del list:
  - del list[index] αφαίρεση στοιχείου της λίστας list[index] στην θέση index.
  - del list: διαγραφή ολόκληρης της μεταβλητής — η μεταβλητή παύει να υπάρχει και κάθε προσπάθεια πρόσβασης σε αυτή θα δώσει NameError.
- Μέθοδος clear(): αδειάζει τη λίστα. Η λίστα παραμένει, αλλά χωρίς περιεχόμενο (κενή: [])

Κρίσιμες Διαφορές

Μέθοδος	Λειτουργία	Αποτέλεσμα
remove(x)	Αφαιρεί βάσει τιμής.	Σφάλμα αν το x δεν υπάρχει.
pop(i)	Αφαιρεί βάσει θέσης και την επιστρέφει.	Χρήσιμο για επεξεργασία του στοιχείου.

3.1.4.2.3 Μέθοδος index(element) - Εύρεση Θέσης

Η μέθοδος index() επιστρέφει τον δείκτη της πρώτης εμφάνισης του ορίσματος element.

Παραδείγματα

```
# Βασική χρήση
fruits = ["μήλο", "πορτοκάλι", "μπανάνα", "κεράσι"]
position = fruits.index("μπανάνα")
print(position)  # 2

# ValueError αν δεν υπάρχει
# position = fruits.index("αχλάδι")  # ValueError: 'αχλάδι' is not in list

# Ασφαλής εύρεση
fruit = "αχλάδι"
```

```python
if fruit in fruits:
    position = fruits.index(fruit)
    print(f"{fruit} βρίσκεται στη θέση {position}")
else:
    print(f"{fruit} δεν βρέθηκε")
```

Κ 67: index: εύρεση θέσης ενός στοιχείου της λίστας

3.1.4.2.4 Μέθοδος count(element) - Μέτρηση Εμφανίσεων

Η μέθοδος count() μετράει πόσες φορές εμφανίζεται το όρισμα element στη λίστα.

Παραδείγματα

```python
# Βασική χρήση
numbers = [1, 2, 3, 2, 4, 2, 5, 2]
count = numbers.count(2)
print(f"Το 2 εμφανίζεται {count} φορές")  # 4

# Αν δεν υπάρχει, επιστρέφει 0
count = numbers.count(99)
print(count)  # 0

# Με strings
text = list("hello world")
print(text.count('l'))  # 3
print(text.count('o'))  # 2
```

Κ 68: count: Μέτρηση εμφανίσεων ενός στοιχείου στην λίστα

3.1.4.2.5 Μέθοδος sort() – Ταξινόμηση στοιχείων

Η μέθοδος sort() ταξινομεί τη λίστα στη θέση της (δεν επιστρέφει νέα λίστα).

Παραδείγματα

```python
# Βασική χρήση - Αύξουσα
numbers = [45, 12, 78, 23, 67]
numbers.sort()
print(numbers)  # [12, 23, 45, 67, 78]

# Βασική χρήση - Αύξουσα
fruits = ["μήλο", "αχλάδι", "κεράσι", "πορτοκάλι"]
fruits.sort()
print(fruits)  # ['αχλάδι', 'κεράσι', 'μήλο', 'πορτοκάλι']

# Φθίνουσα ταξινόμηση
numbers = [45, 12, 78, 23, 67]
```

```
numbers.sort(reverse=True)
print(numbers)  # [78, 67, 45, 23, 12]
```

K 69: sort: Ταξινόμηση στοιχείων της λίστας

3.1.4.2.6 Μέθοδος reverse() – Αντιστροφή στοιχείων

Η μέθοδος reverse() αντιστρέφει τη σειρά των στοιχείων στη λίστα.

Παραδείγματα

```
# Βασική χρήση
numbers = [1, 2, 3, 4, 5]
numbers.reverse()
print(numbers)  # [5, 4, 3, 2, 1]

# Διαφορά από reversed()
numbers = [1, 2, 3, 4, 5]

# reversed() επιστρέφει iterator
rev_iter = reversed(numbers)
rev_list = list(rev_iter)
print(numbers)   # [1, 2, 3, 4, 5] - Αμετάβλητη
print(rev_list)  # [5, 4, 3, 2, 1]
```

K 70: reverse: Αντιστροφή σειράς στοιχείων της λίστας

3.1.4.2.7 Μέθοδος copy() - Δημιουργία Αντιγράφου

Η μέθοδος copy() δημιουργεί και επιστρέφει ένα αντίγραφο της λίστας.

Παραδείγματα

```
# Βασική χρήση
original = [1, 2, 3, 4, 5]
duplicate = original.copy()

# Τροποποίηση αντιγράφου
duplicate.append(6)
print(original)  # [1, 2, 3, 4, 5] - Δεν επηρεάστηκε
print(duplicate)  # [1, 2, 3, 4, 5, 6]
```

K 71: copy: Δημιουργία αντιγράφου της λίστας

3.1.4.3 Ασκήσεις: Λειτουργίες και Μέθοδοι Λιστών

Άσκηση 9: Διαχείριση Καταλόγου Προϊόντων

Δίνεται η αρχική λίστα προϊόντων μιας αποθήκης: `products = ['Γάλα', 'Ψωμί', 'Αυγά', 'Τυρί']` **Ζητούνται τα εξής:**

1. Να προστεθεί το προϊόν **"Γιαούρτι"** στο τέλος της λίστας με τη χρήση της κατάλληλης μεθόδου.
2. Να εισαχθεί το προϊόν **"Μέλι"** στη δεύτερη θέση της λίστας (δείκτης 1).
3. Να αφαιρεθεί το προϊόν **"Αυγά"** από τη λίστα αναζητώντας το με το όνομά του.
4. Να εκτυπωθεί το τελικό πλήθος των προϊόντων χρησιμοποιώντας την ενσωματωμένη λειτουργία len().

Άσκηση 10: Ανάλυση και Ταξινόμηση Δεδομένων

Δίνεται η λίστα με τις θερμοκρασίες που καταγράφηκαν σε μια πόλη κατά τη διάρκεια μιας εβδομάδας: temps = [12, 15, 10, 18, 15, 22, 15]

Ζητούνται τα εξής:

1. Να υπολογιστεί πόσες φορές εμφανίζεται η θερμοκρασία **15** στη λίστα με τη χρήση της μεθόδου .count().
2. Να βρεθεί η θέση (index) της πρώτης εμφάνισης της θερμοκρασίας **22**.
3. Να ταξινομηθούν οι θερμοκρασίες σε **φθίνουσα σειρά** (από τη μεγαλύτερη στη μικρότερη) με τη μέθοδο .sort().
4. Να αντιστραφεί η σειρά των στοιχείων της λίστας με τη μέθοδο .reverse().

Άσκηση 11: Έλεγχος Μέλους και Διαγραφή

Δίνεται η λίστα με τα ονόματα των εγκεκριμένων χρηστών ενός συστήματος: users = ['Νίκος', 'Ελένη', 'Ανδρέας', 'Μαρία', 'Κώστας']

Ζητούνται τα εξής:

1. Να πραγματοποιηθεί έλεγχος αν ο χρήστης **"Γιώργος"** περιλαμβάνεται στη λίστα χρησιμοποιώντας τον τελεστή in. Αν δεν περιλαμβάνεται, να εκτυπώνεται το μήνυμα "Ο χρήστης δεν βρέθηκε".
2. Να αφαιρεθεί ο τελευταίος χρήστης της λίστας με τη μέθοδο .pop() και να αποθηκευτεί το όνομά του σε μια μεταβλητή removed_user.

3. Να διαγραφεί ο χρήστης που βρίσκεται στη θέση 0 με τη χρήση της εντολής del.

Άσκηση 12: Συνδυασμός Λιστών και Επέκταση

Δίνονται δύο λίστες με ονόματα φρούτων: basket_a = ['Μήλο', 'Αχλάδι'] και basket_b = ['Φράουλα', 'Ακτινίδιο']

Ζητούνται τα εξής:

1. Να επεκταθεί η λίστα basket_a με τα στοιχεία της basket_b χρησιμοποιώντας τη μέθοδο .extend().
2. Να ελεγχθεί αν η λέξη **"Μπανάνα"** δεν περιλαμβάνεται στη νέα λίστα basket_a με τη χρήση του τελεστή not in.
3. Να καθαριστούν όλα τα στοιχεία της λίστας basket_a με τη μέθοδο .clear().

3.1.5. Διάσχιση στοιχείων Λίστας (Iteration)

Η διάσχιση (iteration) στοιχείων μιας λίστας αποτελεί τη διαδικασία προσπέλασης κάθε στοιχείου της λίστας με τη σειρά. Οι προγραμματιστές επιλέγουν τον τρόπο διάσχισης ανάλογα με το αν χρειάζονται μόνο την τιμή του στοιχείου ή και τη θέση (δείκτη) του.

3.1.5.1 Διάσχιση βάσει Τιμής (Value-based Iteration)

Η Διάσχιση βάσει Τιμής αποτελεί τον πιο κοινό τρόπο διάσχισης. Ο βρόχος for λαμβάνει κάθε στοιχείο της συλλογής διαδοχικά. Η πολυπλοκότητα της πλήρους διάσχισης είναι γραμμική, δηλαδή O(n), όπου n είναι το πλήθος των στοιχείων.

Παράδειγμα διάσχιση τιμής με τον βρόχο for:

```
grades = [10, 18, 15, 20]
for grade in grades:
    # Εδώ εκτελείται η επεξεργασία της τιμής
    print(f"Βαθμός: {grade}")
```

Κ 72: Διάσχιση βάσει Τιμής (Value-based Iteration) με τον βρόχο for

3.1.5.2 Διάσχιση βάσει Δείκτη (Index-based Iteration) με τη χρήση βρόχου for

Η Διάσχιση βάσει Δείκτη με τη χρήση βρόχου for χρησιμοποιείται όταν απαιτείται η τροποποίηση του στοιχείου στη θέση του (in-place) ή όταν η λογική του προγράμματος εξαρτάται από τη θέση. Στην περίπτωση αυτή της διάσχισης χρησιμοποιούνται οι συναρτήσεις `range()` και `len()`.

Παράδειγμα διάσχισης βάσει Δείκτη με τον βρόχο for:

```
# Διάσχιση βάσει δείκτη (Index-based Iteration)
fruits = ["μήλο", "αχλάδι", "κεράσι", "πορτοκάλι"]

# H range(len(fruits)) δημιουργεί τους δείκτες: 0, 1, 2, 3
for i in range(len(fruits)):
    # Λαμβάνεται το στοιχείο στη θέση i και αντικαθίσταται
    # από μια νέα συμβολοσειρά που περιλαμβάνει το πρόθεμα
    fruits[i] = "Φρέσκο " + fruits[i]

# Εκτύπωση της ενημερωμένης λίστας
print(fruits) #['Φρέσκο μήλο', 'Φρέσκο αχλάδι', 'Φρέσκο κεράσι', 'Φρέσκο πορτοκάλι']
```

Κ 73: Διάσχιση βάσει Δείκτη (Index-based iteration) με τη χρήση βρόχου for

3.1.5.3 Διάσχιση βάσει Τιμής και Δείκτη : enumerate()

Όταν απαιτείται ταυτόχρονα η τιμή και ο δείκτης, προτιμάται η Διάσχιση βάσει Τιμής και Δείκτη με την συνάρτηση enumerate(). Θεωρείται η βέλτιστη πρακτική καθώς καθιστά τον κώδικα πιο ευανάγνωστο και αποδοτικό. Η enumerate() χρησιμοποιείται συχνά για τη δημιουργία αναφορών ή καταλόγων όπου η σειρά εμφάνισης έχει σημασία (π.χ. κατατάξεις, λίστες επιτυχόντων).

```
students = ['Γιώργος Ανδρέου', 'Μαρία Κωνσταντίνου', 'Νίκος Αντωνίου', 'Ελένη Γεωργίου']

# Χρήση της enumerate για ταυτόχρονη λήψη δείκτη (i) και τιμής (name)
# H παράμετρος start=1 ορίζει ότι η αρίθμηση θα ξεκινήσει από το 1 αντί για το 0
for i, name in enumerate(students, start=1):
    print(f"{i}. {name}")

# Έξοδος:
# 1. Γιώργος Ανδρέου
# 2. Μαρία Κωνσταντίνου
# 3. Νίκος Αντωνίου
# 4. Ελένη Γεωργίου
```

Κ 74: Enumerate: Διάσχιση βάσει Τιμής και Δείκτη με τον βρόχο for

3.1.5.4 Διάσχιση με τη χρήση βρόχου While

Η διάσχιση με τη χρήση βρόχου while είναι στην πραγματικότητα μια διάσχιση βάσει δείκτη. Παρόλο που στην Python προτιμάται ο βρόχος for, η while χρησιμοποιείται όταν ο ρυθμός της διάσχισης δεν είναι σταθερός ή όταν η συνθήκη εξόδου δεν εξαρτάται μόνο από το τέλος της λίστας.

Χαρακτηριστικά:

- Απαιτεί την αρχικοποίηση ενός εξωτερικού μετρητή (συνήθως i = 0).
- Απαιτεί την αύξηση του εξωτερικού μετρητή (i += 1) σε κάθε επανάληψη.
- Προσφέρει τον πλήρη έλεγχο του δείκτη.

Παράδειγμα

```
fruits = ["μήλο", "μπανάνα", "πορτοκάλι"]
i = 0  # Αρχικοποίηση δείκτη

while i < len(fruits):
    print(f"Θέση {i}: {fruits[i]}")
    i += 1  # Προσοχή: Αν παραλειφθεί, δημιουργείται ατέρμονος βρόχος
```

Κ 75: Διάσχιση λίστας με τη χρήση βρόχου while και εξωτερικό μετρητή

3.1.5.5 Ασκήσεις Διάσχισης Λιστών (Iteration)

Άσκηση 13: Αναζήτηση Μέγιστου

Δίνεται η λίστα prices = [10.5, 45.0, 12.8, 33.4, 60.1, 22.0]. Να γραφεί κώδικας που διασχίζει τη λίστα και βρίσκει τη μέγιστη τιμή χωρίς τη χρήση της έτοιμης συνάρτησης max().

Άσκηση 14: "Το Καλάθι με τα Φρούτα"

Δίνεται η λίστα: `basket = ['Μήλο', 'Μπανάνα', 'Κεράσι', 'Μήλο', 'Πορτοκάλι', 'Μήλο']`

Ζητούμενο: Να γραφεί κώδικας ο οποίος θα διασχίζει τη λίστα βάσει δείκτη. Εάν το στοιχείο είναι "Μήλο", να αντικαθίσταται από τη φράση "Εξαντλήθηκε". Σε κάθε άλλη περίπτωση, το στοιχείο να παραμένει ως έχει.

Άσκηση 15: "Πίνακας Αποτελεσμάτων"

Δίνεται η λίστα με τους τερματίσαντες ενός αγώνα δρόμου: `runners = ["Δημήτρης", "Κατερίνα", "Κώστας", "Σοφία"]`

Ζητούμενο: Να χρησιμοποιηθεί η `enumerate()` ώστε να εμφανιστούν τα ονόματα των τριών πρώτων αθλητών με τη μορφή:

- Μετάλλιο 1: Δημήτρης
- Μετάλλιο 2: Κατερίνα
- Μετάλλιο 3: Κώστας

3.1.6. Πολυδιάστατοι Πίνακες (Lists of Lists)

Οι πολυδιάστατοι πίνακες είναι λίστες που περιέχουν άλλες λίστες, δημιουργώντας δομές πινάκων (matrices). Οι εμφωλευμένες λίστες (nested lists), χρησιμοποιούν διαδοχικές αγκύλες. Κάθε αγκύλη "εισχωρεί" σε ένα επίπεδο βάθους.

Ένας δισδιάστατος πίνακας αποτελείται από **γραμμές (rows)** και **στήλες (columns)**:

```
# Ένας πίνακας 3x3 (3 γραμμές, 3 στήλες)
matrix = [
    [1, 2, 3], # Γραμμή 0
    [4, 5, 6], # Γραμμή 1
    [7, 8, 9]  # Γραμμή 2
]
```

Κ 76: Ένας δισδιάστατος πίνακας αποτελείται από γραμμές (rows) και στήλες (columns)

Πρόσβαση σε Στοιχεία

- Η πρόσβαση γίνεται με δύο δείκτες: matrix[row][column].
- Ο πρώτος δείκτης ορίζει τη **γραμμή**.
- Ο δεύτερος δείκτης ορίζει τη **στήλη**.
- `matrix[0][0]` -> Το στοιχείο `1` (πρώτη γραμμή, πρώτη στήλη).
- `matrix[1][2]` -> Το στοιχείο `6` (δεύτερη γραμμή, τρίτη στήλη).

Διάσχιση Πολυδιάστατων Πινάκων

Για να προσπελάσουμε όλα τα στοιχεία ενός 2D πίνακα, απαιτούνται **εμφωλευμένοι βρόχοι (nested loops)**. Ο εξωτερικός βρόχος διατρέχει τις γραμμές και ο εσωτερικός τις στήλες της κάθε γραμμής.

Παράδειγμα προσπέλασης με τιμές:

```
matrix = [
    [1, 2, 3], # Γραμμή 0
    [4, 5, 6], # Γραμμή 1
    [7, 8, 9]  # Γραμμή 2
]
for row in matrix:
    for element in row:
        print(element, end=" ")
    print() # Αλλαγή γραμμής μετά από κάθε σειρά
```

Κ 77: Προσπέλαση στοιχείων βάσει τιμών

Παράδειγμα προσπέλασης με range:

```
# Ένας πίνακας 3x3 (3 γραμμές, 3 στήλες)
matrix = [
    [1, 2, 3], # Γραμμή 0
    [4, 5, 6], # Γραμμή 1
    [7, 8, 9]  # Γραμμή 2
]
for i in range(len(matrix)):
    for j in range(len(matrix[i])):
        print(f"matrix[{i}][{j}] = {matrix[i][j]}")
```

Κ 78: Προσπέλαση στοιχείων πίνακα βάσει δείκτη

Τρισδιάστατοι Πίνακες (3D Arrays)

Ένας 3D πίνακας μπορεί να οπτικοποιηθεί ως ένας κύβος ή ως μια στοίβα από σελίδες, όπου κάθε σελίδα είναι ένας 2D πίνακας.

Παράδειγμα 3D πίνακα που αναπαριστά 3 τάξεις x 4 μαθητές x 3 μαθήματα:

```
classes = [
    # Τάξη Α
    [
        [85, 90, 78],  # Μαθητής 1
        [88, 76, 95],  # Μαθητής 2
        [92, 88, 91],  # Μαθητής 3
        [79, 84, 87]   # Μαθητής 4
    ],
    # Τάξη Β
    [
        [90, 92, 88],
        [85, 89, 91],
        [78, 82, 80],
        [94, 96, 93]
    ],
    # Τάξη Γ
    [
        [87, 85, 89],
        [92, 90, 94],
        [81, 83, 86],
        [88, 87, 90]
    ]
]

# Πρόσβαση: classes[τάξη][μαθητής][μάθημα]
print(f"Τάξη Α, Μαθητής 2, Μάθημα 3: {classes[0][1][2]}")  # 95
```

```python
# Μέσος όρος κάθε τάξης
for i, classroom in enumerate(classes, 1):
    total = sum(sum(student) for student in classroom)
    count = len(classroom) * len(classroom[0])
    average = total / count
    print(f"Μέσος όρος Τάξης {chr(64+i)}: {average:.2f}")
```

Κ 79: Παράδειγμα 3D πίνακα που αναπαριστά 3 τάξεις x 4 μαθητές x 3 μαθήματα

Στο παραπάνω παράδειγμα για να προσπελαστεί ο βαθμός του 2^{ου} Μαθητή της Α τάξης στο 3^{ο} Μάθημα γράφουμε: `classes[0][1][2]  # 95`

3.1.6.1 Άσκηση 16: Πολυδιάστατοι Πίνακες

Να υλοποιηθεί σύστημα βαθμολόγησης σε πίνακα 5 μαθητών x 4 μαθημάτων

```python
grades = [

        [87, 85, 89, 72],
        [92, 90, 94, 76],
        [81, 23, 83, 86],
        [88, 45, 87, 90],
        [94, 100,96, 93]
    ]
```

που να:

1. Υπολογίζει τον μέσο όρο κάθε μαθητή
2. Βρίσκει το καλύτερο μάθημα κάθε μαθητή
3. Εμφανίζει ποιοι μαθητές πέρασαν (μέσος όρος ≥ 50)
4. Ταξινομεί μαθητές κατά επίδοση

3.1.7. Αντιγραφή Λιστών

Η Αντιγραφή Λιστών δημιουργεί ένα νέο, αυτόνομο αντικείμενο Λίστας στη μνήμη με τα ίδια περιεχόμενα. Ακολουθούν οι δύο βασικοί τρόποι για τη δημιουργία ενός ανεξάρτητου αντίγραφου μιας μονοδιάστατης λίστας:

- **Η μέθοδος copy():** Είναι η πιο διαδεδομένη μέθοδος που εφαρμόζεται απευθείας πάνω στο αντικείμενο της λίστας.

```python
original_list = [10, 20, 30]
# Δημιουργία αντιγράφου
new_list = original_list.copy()
```

```python
# Τροποποίηση του αντιγράφου
new_list[0] = 99

print(original_list) # [10, 20, 30] -> Παραμένει αμετάβλητη
print(new_list)      # [99, 20, 30] -> Άλλαξε μόνο το αντίγραφο
```

Κ 80: Αντιγραφή λίστας με την μέθοδο copy

- **Ο κατασκευαστής `list()`:** Χρησιμοποιώντας τον κατασκευαστή (constructor) της λίστας δεσμεύεται νέος χώρος στην μνήμη.

```python
vathmoi_mathiti = [15, 18, 14]
asfales_antigrafo = list(vathmoi_mathiti)
```

Κ 81: Αντιγραφή λίστας με τον κατασκευαστή list()

- **Αντιγραφή με Slicing ([:]):** Με αυτό τον τρόπο στην Python λαμβάνεται ένα κομμάτι της λίστας από το πρώτο έως το τελευταίο στοιχείο και τοποθετείται σε μια νέα λίστα. Όπως η copy(), η [:] κάνει **ρηχό αντίγραφο** (shallow copy).

```python
original_list = [10, 20, 30]
# Το [:] σημαίνει "από την αρχή μέχρι το τέλος"
new_list = original_list[:]

new_list[0] = 100
print(original_list) # [10, 20, 30] - Δεν επηρεάστηκε
```

Κ 82: Αντιγραφή λίστας με slicing [:]

- **Η Βαθιά Αντιγραφή (Deep Copy):** Όταν η λίστα είναι **πολυδιάστατη** (λίστα που περιέχει άλλες λίστες), οι παραπάνω μέθοδοι δεν αντιγράψουν τις εσωτερικές – εμφωλευμένες λίστες. Σε αυτή την περίπτωση, απαιτείται η χρήση της `deepcopy`.

  Παράδειγμα Προβλήματος:

```python
matrix = [[1, 2], [3, 4]]
shallow_copy = matrix.copy()

shallow_copy[0][0] = 99
print(matrix[0][0]) # Θα τυπώσει 99! Η αρχική λίστα αλλοιώθηκε.
```

Κ 83: Πρόβλημα στην Αντιγραφή λίστας με εμφωλευμένες λίστες

Η λύση της deepcopy

```python
import copy

matrix = [[1, 2], [3, 4]]
deep_copy = copy.deepcopy(matrix)

deep_copy[0][0] = 99
print(matrix[0][0]) # Θα τυπώσει 1. Η αρχική λίστα είναι ασφαλής!
```

Κ 84: Αντιγραφή λίστας με εμφωλευμένες λίστες με την χρήση της copy.deepcopy

3.1.7.1 Ασκήσεις Αντιγραφής Λιστών

Άσκηση 17: Έχετε μια λίστα με τα ονόματα των 5 μαθητών που συμμετέχουν σε ένα τμήμα. Θέλετε να δημιουργήσετε ένα αντίγραφο της λίστας για να προσθέσετε έναν επιπλέον μαθητή που ήρθε με μετεγγραφή, αλλά η αρχική λίστα του τμήματος πρέπει να παραμείνει αμετάβλητη.

Άσκηση 18: Έχετε τον πίνακα με τις βαθμολογίες (5 μαθητές x 2 μαθήματα). Αποφασίζετε να δώσετε μια "δεύτερη ευκαιρία" στον πρώτο μαθητή, αλλάζοντας τον πρώτο του βαθμό σε 100 στο αντίγραφο του πίνακα. Πρέπει να αποδείξετε ότι ο αρχικός πίνακας βαθμολογιών δεν επηρεάστηκε.

3.2. Εγκατάσταση Βιβλιοθηκών με τη χρήση του PIP από το PyPI

Στο οικοσύστημα της Python, η δυνατότητα επέκτασης των βασικών λειτουργιών της γλώσσας μέσω εξωτερικών βιβλιοθηκών αποτελεί ένα από τα ισχυρότερα πλεονεκτήματά της. Για την ομαλή ενσωμάτωση αυτών των βιβλιοθηκών, χρησιμοποιούνται δύο βασικά εργαλεία: το **PyPI** (Python Package Index) και το **pip** (Pip Installs Packages).

3.2.1. Τι είναι το PyPI (Python Package Index)

Το **PyPI** είναι το επίσημο κεντρικό αποθετήριο λογισμικού τρίτων για την Python. Φανταστείτε το ως μια τεράστια "βιβλιοθήκη" στο διαδίκτυο, η οποία φιλοξενεί εκατοντάδες χιλιάδες πακέτα κώδικα (libraries) που έχουν δημιουργηθεί από την κοινότητα των προγραμματιστών.

Χαρακτηριστικά:

- Φιλοξενεί πάνω από 500,000 πακέτα (libraries).

- Είναι Δωρεάν και ανοιχτό για όλους.

- Είναι Διαθέσιμο στο: https://pypi.org

- Περιέχει βιβλιοθήκες για κάθε χρήση (web, data science, AI, κ.λπ.)

3.2.2. Τι είναι το PIP (Pip Installs Packages);

Το **pip** (Pip Installs Packages) είναι το επίσημο προεπιλεγμένο εργαλείο διαχείρισης πακέτων (package manager) της Python. Πρόκειται για ένα εργαλείο γραμμής εντολών που επιτρέπει:

- την σύνδεση στο PyPI,
- την αυτόματη λήψη βιβλιοθηκών,
- την εγκατάσταση, ενημέρωση και διαγραφή πακέτων στο σύστημα του κάθε χρήστη,
- την διαχείριση εξαρτήσεων (dependencies).

3.2.2.1 Έλεγχος εγκατάστασης του PIP

Μεταβείτε στη γραμμή εντολών στη θέση του καταλόγου scripts της Python και πληκτρολογήστε τα εξής:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip --version
```

3.2.2.2 Εγκατάσταση PIP

Για την εγκατάσταση του PIP, μεταβείτε στον σύνδεσμο https://pypi.org/project/pip/ για την λήψη και εγκατάστασή του. Το PIP εγκαθίσταται αυτόματα κατά την εγκατάσταση της python (https://www.python.org/).

3.2.3. Πρακτική Εφαρμογή: Βασικές Εντολές

Η χρήση του pip πραγματοποιείται μέσω του **Terminal** (macOS/Linux) ή της **Γραμμής Εντολών / PowerShell** (Windows).

3.2.3.1 Εγκατάσταση Βιβλιοθήκης

Η βασική σύνταξη για την εγκατάσταση ενός πακέτου είναι:

```
pip install [όνομα_βιβλιοθήκης]
```

Παράδειγμα: Για την εγκατάσταση της δημοφιλής βιβλιοθήκης **requests** για τη διαχείριση αιτημάτων στο διαδίκτυο γράφετε την εντολή:

```
pip install requests
```

3.2.4. Προβολή Εγκατεστημένων Βιβλιοθηκών

Για την προβολή των βιβλιοθηκών που είναι ήδη εγκατεστημένες στο σύστημα ενός χρήστη καθώς και των εκδόσεων τους, γράφετε την εντολή:

```
pip list
```

3.2.5. Απεγκατάσταση Βιβλιοθήκης

Για την απεγκατάσταση μιας βιβλιοθήκης που δεν είναι πλέον απαραίτητη, γράφετε την εντολή:

```
pip uninstall [όνομα_βιβλιοθήκης]
```

Παράδειγμα: Για την απεγκατάσταση της βιβλιοθήκης **requests**

```
pip uninstall requests
```

3.2.6. Διαδικασία Επαλήθευσης

Μετά την εγκατάσταση μιας βιβλιοθήκης μέσω του pip, χρειάζεται να επιβεβαιωθεί ότι η Python μπορεί να την αναγνωρίσει. Αυτό επιτυγχάνεται με την εντολή `import` μέσα στον κώδικα.

Παράδειγμα:

```
import requests

# Αν δεν εμφανιστεί σφάλμα (ModuleNotFoundError), η εγκατάσταση ήταν επιτυχής.
print("Η βιβλιοθήκη εγκαταστάθηκε σωστά!")
```

Κ. 85: Χρήση βιβλιοθήκης με την import

3.2.7. Άσκηση Διαχείρισης Βιβλιοθηκών

Άσκηση 19: Εξοικείωση με την εύρεση και εγκατάσταση βιβλιοθηκών

1. Ανοίξτε τη γραμμή εντολών του υπολογιστή σας.
2. Χρησιμοποιήστε το pip για να εγκαταστήσετε τη βιβλιοθήκη emoji (μια βιβλιοθήκη που επιτρέπει τη χρήση εικονιδίων στον κώδικα).
3. Δημιουργήστε ένα νέο αρχείο Python (π.χ. test_emoji.py).
4. Γράψτε τον παρακάτω κώδικα για να ελέγξετε τη λειτουργία της:

```
import emoji
print(emoji.emojize('Η Python είναι :thumbs_up:'))
```

5. Εκτελέστε το πρόγραμμα και επιβεβαιώστε ότι το μήνυμα εμφανίζεται με το αντίστοιχο εικονίδιο.

3.3. Modules και Packages

Καθώς τα προγράμματα αυξάνονται σε μέγεθος και πολυπλοκότητα, η διατήρηση όλου του κώδικα σε ένα μόνο αρχείο καθίσταται πρακτικά αδύνατη. Η Python παρέχει μηχανισμούς για τον τεμαχισμό του κώδικα σε μικρότερα, επαναχρησιμοποιήσιμα και οργανωμένα τμήματα: τα **Modules** και τα **Packages**.

3.3.1. Τι είναι το Module;

Ένα **Module** (Ενότητα) είναι ένα απλό αρχείο με κατάληξη .py, το οποίο περιέχει ορισμούς συναρτήσεων, μεταβλητών ή κλάσεων. Σκοπός του είναι να ομαδοποιεί σχετικό κώδικα ώστε να μπορεί να χρησιμοποιηθεί σε άλλα προγράμματα μέσω της εντολής `import`.

3.3.2. Οργάνωση κώδικα σε Modules

Η οργάνωση του κώδικα σε modules ακολουθεί την αρχή του **Modular Programming** (Αρθρωτός Προγραμματισμός). Αυτό διασφαλίζει:

1. **Επαναχρησιμοποίηση:** Γράφετε τον κώδικα μία φορά και τον καλείτε από παντού.
2. **Ευκολία Συντήρησης:** Διορθώνετε ένα σφάλμα σε ένα αρχείο και η αλλαγή εφαρμόζεται παντού.
3. **Αναγνωσιμότητα:** Το κύριο πρόγραμμα παραμένει "καθαρό" και σύντομο.

Πρακτικό Παράδειγμα

Φανταστείτε ότι θέλουμε να οργανώσουμε τις μαθηματικές πράξεις ενός συστήματος βαθμολόγησης.

Βήμα 1: Δημιουργία του Module (statistics_tools.py)

```
# Αρχείο: statistics_tools.py

def calculate_average(grades_list):
    return sum(grades_list) / len(grades_list)

def get_status(average):
    return "Πέρασε" if average >= 50 else "Απέτυχε"
```

Κ 86: Δημιουργία Module

Βήμα 2: Χρήση του Module στο Κύριο Πρόγραμμα (main.py)

```
# Αρχείο: main.py
import statistics_tools

student_grades = [70, 85, 90, 60]

# Κλήση συναρτήσεων από το module
avg = statistics_tools.calculate_average(student_grades)
result = statistics_tools.get_status(avg)

print(f"Μέσος Όρος: {avg}, Κατάσταση: {result}")
```

Κ 87: Χρήση Module

3.3.3. Τρόποι Εισαγωγής (Importing)

Υπάρχουν οι ακόλουθοι τρόποι για την εισαγωγή (importing κώδικα), ανάλογα με τις ανάγκες:

`import module_name`	Εισάγει όλο το module **Παράδειγμα**: `import numpy`
`from module_name import function_name`	Εισάγει συγκεκριμένη συνάρτηση **Παράδειγμα:** `from statistics_tools import calculate_average`
`import module_name as alias`	Δίνει ένα σύντομο ψευδώνυμο **Παράδειγμα**: `import numpy as np`

3.3.4. Άσκηση 20 - Δημιουργία και χρήση Module

1. Δημιουργήστε ένα αρχείο με όνομα **converter.py**. Μέσα σε αυτό, ορίστε μια συνάρτηση to_percentage(score, total) που δέχεται έναν βαθμό και ένα σύνολο, και επιστρέφει το ποσοστό επί τοις εκατό (%).
2. Δημιουργήστε ένα δεύτερο αρχείο με όνομα **app.py**.
3. Στο αρχείο app.py, κάντε εισαγωγή (import) τη συνάρτηση από το converter.py.
4. Ζητήστε από τον χρήστη έναν βαθμό (π.χ. 15) και ένα άριστα (π.χ. 20) και εμφανίστε το ποσοστό (π.χ. 75.0%).

3.3.5. Τι είναι το Package;

Ένα **Package** (Πακέτο) είναι μια συλλογή από πολλά Modules οργανωμένα σε έναν φάκελο. Για να αναγνωρίσει η Python έναν φάκελο ως Package, παραδοσιακά χρησιμοποιείται ένα αρχείο με το όνομα __init__.py (ακόμα και αν αυτό είναι κενό). Τα Packages επιτρέπουν μια ιεραρχική δομή στον κώδικα `package.subpackage.module`. Το package δύναται να περιλαμβάνει άλλα packages. Το τελευταίο επίπεδο της ιεραρχικής δομής είναι τα modules.

3.3.5.1 Τα Επίπεδα της Ιεραρχίας

Η ιεραρχία, από το ανώτερο προς το κατώτερο επίπεδο, διαμορφώνεται ως εξής:

1. **Library (Βιβλιοθήκη):** Μια συλλογή από πολλά Packages που επιλύουν ένα ευρύ φάσμα προβλημάτων (π.χ. η Standard Library της Python).
2. **Package (Πακέτο):** Ένας φυσικός φάκελος στο σύστημα αρχείων που περιέχει Modules ή άλλα Sub-packages. Για να θεωρηθεί ένας φάκελος ως package, περιέχει συνήθως το αρχείο `__init__.py`.
3. **Sub-package (Υπο-πακέτο):** Ένας φάκελος μέσα σε έναν άλλο φάκελο πακέτου. Χρησιμοποιείται για περαιτέρω κατηγοριοποίηση.
4. **Module (Ενότητα):** Ένα αρχείο με κατάληξη `.py`. **Αυτό είναι το τελευταίο επίπεδο της ιεραρχίας όσον αφορά τη δομή των αρχείων.**
5. **Attributes (Στοιχεία):** Μέσα στο Module υπάρχουν οι συναρτήσεις, οι μεταβλητές και οι κλάσεις που περιέχουν τον πραγματικό κώδικα.

3.3.5.1.1 Πρακτικό Παράδειγμα Δομής

Ας υποθέσουμε ότι δημιουργούμε ένα πακέτο για μια πανεπιστημιακή εφαρμογή με το όνομα university.

Εικόνα 13: Παράδειγμα Ιεραρχικής δομής package, subpackages, modules

Ανάλυση της Δομής:

- university: Είναι το κεντρικό package.
- students & grades: Είναι υπο-πακέτα (subpackages) που ομαδοποιούν διαφορετικές λειτουργίες.
- enrollment.py, profiles.py, calculator.py: Είναι τα Modules. Εδώ σταματά η ιεραρχία των αρχείων. Μέσα σε αυτά τα αρχεία γράφετε ο κώδικας.

3.3.5.2 Πρόσβαση στα επίπεδα της ιεραρχίας

Η πρόσβαση στα επίπεδα της ιεραρχίας γίνεται με τη χρήση της τελείας (.):

```python
# Πρόσβαση στο module 'calculator' που βρίσκεται στο sub-package 'grades'
import university.grades.calculator

# Χρήση μιας συνάρτησης compute_average που βρίσκεται μέσα στο module calculator
university.grades.calculator.compute_average([10, 20, 30])
```

Κ 88: Πρόσβαση στα επίπεδα της ιεραρχίας packages, subpackages, modules

3.3.5.3 Το αρχείο __init__.py

Στην πιο απλή του μορφή, το __init__.py **δεν περιέχει τίποτα**. Η ύπαρξη του αρχείου στον φάκελο "σηματοδοτήσει" στην Python ότι ο φάκελος αυτός πρέπει να αντιμετωπίζεται ως **Package** και όχι ως ένας τυχαίος κατάλογος αρχείων. Αποτελεί την πιο συνηθισμένη πρακτική για την οργάνωση των modules σε υποφακέλους.

Το αρχείο __init__.py λειτουργεί ως η "δημόσια πρόσοψη" (Interface) ενός πακέτου.

- **Αφαίρεση Εσωτερικής Πολυπλοκότητας:** Ο χρήστης δεν απαιτείται να γνωρίζει την ακριβή εσωτερική δομή ή τη διαδρομή των αρχείων (modules) μέσα στο πακέτο. Οι συναρτήσεις και οι κλάσεις μπορούν να **"εκτεθούν" (export)** απευθείας στο ανώτερο επίπεδο του πακέτου.
- **Απλοποίηση των Εισαγωγών (Imports):** Αντί για δαιδαλώδεις εντολές εισαγωγής π.χ.
    - from university.grades.calculator import compute_gpa
    - ο χρήστης μπορεί να χρησιμοποιεί μια απλή και διαισθητική σύνταξη
      from university import compute_gpa
- **Ευελιξία στη Συντήρηση:** Ο δημιουργός του πακέτου μπορεί να αναδιοργανώσει τα εσωτερικά modules ή να μετονομάσει αρχεία χωρίς να επηρεαστεί ο κώδικας των χρηστών. Αρκεί να ενημερωθούν οι αναφορές μέσα στο __init__.py, διατηρώντας έτσι τη συμβατότητα της εφαρμογής.

Παράδειγμα αρχείου **__init__.py** στο **package university**:

```
# Φέρνουμε τις συναρτήσεις από τα βάθη της ιεραρχίας στην "επιφάνεια"
from .students.profiles import get_student_name
from .grades.calculator import compute_gpa

# Προαιρετικά ορίζουμε και την έκδοση
__version__ = "2.1.0"
```

K 89: Περιεχόμενο __init__.py: φέρνει τις συναρτήσεις των modules στο πρώτο επίπεδο της ιεραρχίας

Χωρίς την απλοποίηση με την χρήση του __init__.py **:**

Για να χρησιμοποιήσει μια συνάρτηση ο προγραμματιστής πρέπει να γράψει όλη τη διαδρομή από την αρχή του package έως το module. Είναι εύκολο να κάνει λάθος στο όνομα του αρχείου ή του φακέλου.

```
from university.students.profiles import get_student_name
from university.grades.calculator import compute_gpa

name = get_student_name(101)
gpa = compute_gpa(101)
```

K 90: Συμβατός τρόπος χρησιμοποίησης συναρτήσεων σε modules που βρίσκονται σε packages χωρίς την χρήση του __init__.py

Με την απλοποίηση στο `__init__.py` :

Τώρα το πακέτο `university` συμπεριφέρεται σαν μια ενιαία εργαλειοθήκη.

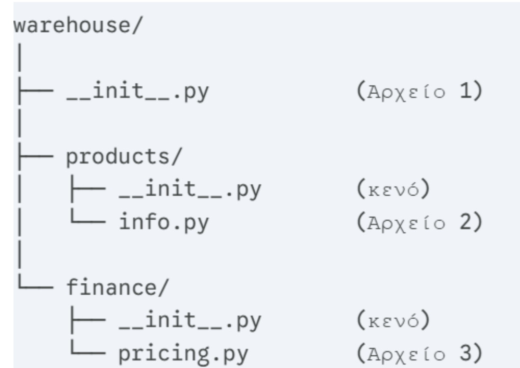
```python
# Ο χρήστης βλέπει μόνο το 'university' και τις βασικές του λειτουργίες
name = university.get_student_name(101)
gpa = university.compute_gpa(101)

print(f"Εφαρμογή University v{university.__version__}")
```

Κ 91: Με την χρήση __init__.py απλοποιείται ο κώδικας το package συμπεριφέρεται σαν μια ενιαία βιβλιοθήκη

3.3.5.4 Άσκηση 21 - Δημιουργία και χρήση Packages

Να δημιουργήσετε ένα πακέτο που διαχειρίζεται προϊόντα και τιμές, οργανωμένο σε υπο-πακέτα, και να διευκολύνετε τον χρήστη να καλεί τις βασικές συναρτήσεις.

- Βήμα 1. Δομή Αρχείων: Δημιουργήστε την παρακάτω δομή φακέλων και αρχείων:



- Βήμα 2: Περιεχόμενο Modules
    - Στο products/info.py: Ορίστε μια συνάρτηση get_product(id) που επιστρέφει το όνομα ενός προϊόντος (π.χ. "Laptop" για το ID 101).
    - Στο finance/pricing.py: Ορίστε μια συνάρτηση apply_discount(price, discount) που επιστρέφει την τελική τιμή μετά την έκπτωση.
- Βήμα 3: Απλοποίηση στο __init__.py: Στο κεντρικό αρχείο warehouse/__init__.py, εισάγετε τις δύο παραπάνω συναρτήσεις έτσι ώστε να είναι προσβάσιμες απευθείας από το πακέτο warehouse.
- Βήμα 4: Εκτέλεση (main.py): Δημιουργήστε ένα αρχείο main.py έξω από τον φάκελο warehouse και κάντε τα εξής:
    - Εισάγετε το πακέτο: import warehouse.
    - Καλέστε τη συνάρτηση warehouse.get_product(101).
    - Καλέστε τη συνάρτηση warehouse.apply_discount(1000, 10).
    - Εκτυπώστε τα αποτελέσματα.

3.4. Η Μεταβλητή __name__ και ο Ρόλος της στην Εκτέλεση Κώδικα

Στην Python, κάθε φορά που εκτελείται ένα αρχείο (module), η γλώσσα ορίζει αυτόματα κάποιες ειδικές μεταβλητές. Η σημαντικότερη από αυτές είναι η __name__. Ανάλογα με τον τρόπο εκτέλεσης ενός αρχείου, η μεταβλητή __name__ μπορεί να λάβει δύο διαφορετικές τιμές.

- Τιμές που επιτρέπουν να γνωρίζουμε αν το αρχείο εκτελείται **αυτόνομα** (ως κύριο πρόγραμμα)
- ή αν το αρχείο έχει εισαχθεί (**import**) σε ένα άλλο αρχείο.

Συγκεκριμένα, όταν εκτελείτε ένα αρχείο Python, η μεταβλητή __name__ λαμβάνει μία από τις δύο παρακάτω τιμές:

- **"__main__"**: Αυτή η τιμή αποδίδεται στη μεταβλητή αν το αρχείο εκτελείται απευθείας από τον χρήστη (π.χ. python script.py).
- **Το όνομα του αρχείου**: Αν το αρχείο εισάγεται ως module από άλλο πρόγραμμα χρησιμοποιώντας την εντολή import, τότε η __name__ ισούται με το όνομα του αρχείου (χωρίς την κατάληξη .py).

3.4.1. Γιατί είναι απαραίτητος ο έλεγχος if __name__ == "__main__"

Η δυνατότητα διάκρισης μεταξύ απευθείας εκτέλεσης και εισαγωγής ως module είναι εξαιρετικά χρήσιμη στον σχεδιασμό προγραμμάτων Python. Επιτρέπει στον προγραμματιστή να γράψει κώδικα που συμπεριφέρεται διαφορετικά ανάλογα με το πώς χρησιμοποιείται.

Επαναχρησιμοποιησιμότητα Κώδικα

Ένα αρχείο μπορεί να περιέχει χρήσιμες συναρτήσεις και κλάσεις που μπορούν να εισαχθούν και να χρησιμοποιηθούν από άλλα προγράμματα. Ταυτόχρονα, το ίδιο αρχείο μπορεί να περιέχει κώδικα που το καθιστά ένα αυτόνομο πρόγραμμα. Χωρίς τον έλεγχο της __name__, ο κώδικας εκτέλεσης θα τρέχει κάθε φορά που κάποιος εισάγει το module, κάτι που συνήθως δεν είναι επιθυμητό.

Δοκιμές και Επίδειξη

Ο κώδικας που βρίσκεται μέσα στο block if __name__ == "__main__": μπορεί να χρησιμοποιηθεί για την δοκιμή των συναρτήσεων του module ή την παρουσίαση παραδειγμάτων χρήσης. Αυτό είναι ιδιαίτερα χρήσιμο κατά την ανάπτυξη και την τεκμηρίωση του κώδικα.

Οργάνωση Προγράμματος

Η χρήση αυτού του προτύπου οδηγεί σε καλύτερα οργανωμένο κώδικα. Ο διαχωρισμός μεταξύ των ορισμών (συναρτήσεις, κλάσεις) και της κύριας λογικής εκτέλεσης κάνει τον κώδικα πιο ευανάγνωστο και πιο εύκολο στη συντήρηση.

Πρακτικό Παράδειγμα

Στο παρακάτω παράδειγμα υπάρχει ο μηχανισμός στο αρχείο **calculator.py**.

Αρχείο 1: calculator.py

```
def add(a, b):
    return a + b

# Αυτός ο κώδικας θα εκτελεστεί ΜΟΝΟ αν τρέξουμε το calculator.py απευθείας.
if __name__ == "__main__":
    print("Εκτέλεση δοκιμών για το module calculator:")
    print(f"Δοκιμή: 2 + 3 = {add(2, 3)}")
```

Αρχείο 2: main.py: Στο αρχείο αυτό έχουμε εισάγει τον κώδικα του calculator ως module με την χρήση της import.

```
import calculator

print("Καλώς ήρθατε στην εφαρμογή!")
result = calculator.add(10, 20)
print(f"Το αποτέλεσμα είναι: {result}")
```

- Αν εκτελεστεί το calculator.py, θα εμφανιστεί το μήνυμα "Εκτέλεση δοκιμών για το module calculator:".
- Αν εκτελεστεί το main.py, το μήνυμα "Εκτέλεση δοκιμών για το module calculator:" **δεν θα εμφανιστεί**, διότι στο calculator.py η μεταβλητή __name__ πλέον ισούται με "calculator" και όχι με "__main__".

3.4.2. Άσκηση 22 με τη χρήση της __name__

1. Δημιουργήστε ένα αρχείο greetings.py.
   - Μέσα σε αυτό, ορίστε μια συνάρτηση say_hello() που εκτυπώνει "Γεια σας από το module!".
   - Κάτω από τη συνάρτηση, προσθέστε έναν έλεγχο if __name__ == "__main__": που θα εκτυπώνει το μήνυμα: "Το module εκτελείται αυτόνομα".
2. Δημιουργήστε ένα δεύτερο αρχείο index.py.
   - Στο index.py, κάντε import greetings και καλέστε τη συνάρτηση greetings.say_hello().

- Εκτελέστε το index.py και παρατηρήστε ποια μηνύματα εμφανίζονται στην οθόνη.

Ερώτηση προς απάντηση: Γιατί το μήνυμα "Το module εκτελείται αυτόνομα" δεν εμφανίστηκε όταν τρέξατε το index.py;

3.5. Ερωτήσεις Κλειστού Τύπου

1. Ποιο είναι το αποτέλεσμα της εντολής list("HELLO");
   A. ["HELLO"]
   B. ['H', 'E', 'L', 'L', 'O']
   C. ['HELLO']
   D. TypeError
2. Ποιο είναι το αποτέλεσμα της εντολής list(range(2, 10, 3));
   A. [2, 5, 8]
   B. [2, 5, 8, 11]
   C. [3, 6, 9]
   D. [2, 4, 6, 8]
3. Δίνεται η λίστα data = [10, 20, 30, 40, 50]. Τι επιστρέφει η εντολή data[-3:];
   A. [10, 20, 30]
   B. [30, 40, 50]
   C. [20, 30, 40]
   D. [30, 40]
4. Ποια είναι η διαφορά μεταξύ append([4, 5]) και extend([4, 5]) σε μια λίστα [1, 2, 3]
   A. Και οι δύο δίνουν [1, 2, 3, 4, 5]
   B. Η append δίνει [1, 2, 3, [4, 5]] και η extend δίνει [1, 2, 3, 4, 5]
   C. Η append δίνει [1, 2, 3, 4, 5] και η extend δίνει [1, 2, 3, [4, 5]]
   D. Και οι δύο δίνουν [1, 2, 3, [4, 5]]
5. Δίνεται ο κώδικας:

```
nums = [1, 2, 3, 2, 4, 2]
nums.remove(2)
print(nums)
```

   Ποιο είναι το αποτέλεσμα;
   A. [1, 3, 4]
   B. [1, 2, 3, 4, 2]
   C. [1, 3, 2, 4, 2]
   D. ValueError

3.6. Ασκήσεις προς επίλυση

3.6.1. Άσκηση 23. Εύρεση Μικρότερου Αριθμού

Δίνεται η λίστα: `temperatures = [22, 18, 25, 14, 30, 19]`

Να γραφεί πρόγραμμα που:

1. Διασχίζει τη λίστα με βρόχο for και βρίσκει τη χαμηλότερη θερμοκρασία χωρίς τη χρήση της συνάρτησης min().
2. Εκτυπώνει τη χαμηλότερη θερμοκρασία.

3.6.2. Άσκηση 24: Πίνακας Βαθμολογιών

Δίνεται ο πίνακας βαθμολογιών 4 φοιτητών σε 3 μαθήματα:

```
grades = [
    [85, 70, 92],
    [60, 45, 55],
    [90, 88, 95],
    [40, 38, 50]
]
```

Ζητούνται:

1. Να υπολογιστεί ο μέσος όρος κάθε φοιτητή χρησιμοποιώντας εμφωλευμένους βρόχους.
2. Να εμφανιστεί αν κάθε φοιτητής πέρασε (μέσος όρος ≥ 50) ή απέτυχε, χρησιμοποιώντας enumerate() με start=1.
3. Να δημιουργηθεί βαθύ αντίγραφο (deepcopy) του πίνακα, να αυξηθούν όλοι οι βαθμοί του αντιγράφου κατά 5 μονάδες, και να αποδειχθεί ότι ο αρχικός πίνακας δεν επηρεάστηκε.

ΚΕΦΑΛΑΙΟ 4: ΣΥΝΘΕΤΟΙ ΤΥΠΟΙ ΔΕΔΟΜΕΝΩΝ ΠΕΡΑ ΑΠΟ ΤΙΣ ΛΙΣΤΕΣ

Στο προηγούμενο Κεφάλαιο εξετάστηκαν αναλυτικά οι λίστες, η πιο διαδεδομένη δομή δεδομένων της Python, ενώ έγινε μια πρώτη αναφορά στις πλειάδες, τα σύνολα και τα λεξικά. Στο παρόν κεφάλαιο, οι δομές αυτές αναλύονται με λεπτομέρεια, καλύπτοντας τη δημιουργία, την πρόσβαση στα στοιχεία τους, τις βασικές λειτουργίες και τη διάσχισή τους. Κάθε δομή εξυπηρετεί διαφορετικές ανάγκες: οι πλειάδες (tuples) εγγυώνται την ακεραιότητα σταθερών δεδομένων, τα σύνολα (sets) διαχειρίζονται μοναδικά στοιχεία και υποστηρίζουν πράξεις συνόλων, ενώ τα λεξικά (dictionaries) προσφέρουν γρήγορη αναζήτηση μέσω ζευγών κλειδιού-τιμής.

Η σωστή επιλογή και αξιοποίηση των σύνθετων τύπων αποτελεί βασικό στοιχείο για αποδοτικό προγραμματισμό στην Python, ειδικά όταν τα δεδομένα αυξάνονται σε μέγεθος και πολυπλοκότητα. Για παράδειγμα, μια λίστα students = ["Μαρία", "Γιάννης", "Ελένη"] μπορεί να τροποποιηθεί προσθέτοντας νέους μαθητές, ενώ ένα λεξικό grades = {"Μαρία": 18, "Γιάννης": 16} επιτρέπει την άμεση πρόσβαση στον βαθμό κάθε μαθητή. Κάθε σύνθετος τύπος έχει τις δικές του μεθόδους και χαρακτηριστικά που τον καθιστούν κατάλληλο για συγκεκριμένες εφαρμογές, και η επιλογή του σωστού τύπου δεδομένων βελτιώνει την απόδοση και την αναγνωσιμότητα του κώδικα.

Παράλληλα, εξετάζεται η έννοια της μεταβλητότητας (mutability) και της αμεταβλητότητας (immutability), η οποία αποτελεί θεμελιώδη διάκριση μεταξύ των δομών δεδομένων της Python και καθορίζει τον τρόπο με τον οποίο μπορούν να τροποποιηθούν τα δεδομένα μετά τη δημιουργία τους. Η κατανόηση αυτής της ιδιότητας είναι απαραίτητη για τη σωστή επιλογή δομής ανάλογα με τις απαιτήσεις κάθε προβλήματος.

Επιπλέον, παρουσιάζονται οι περιφραστικές λίστες και τα περιφραστικά λεξικά (comprehensions), μια συνοπτική και εκφραστική τεχνική δημιουργίας δομών δεδομένων, καθώς και οι διάφοροι τρόποι μορφοποίησης λεκτικών (f-strings, μέθοδος `format()`, τελεστής `%`). Το κεφάλαιο ολοκληρώνεται με χαρακτηριστικά προβλήματα που αναδεικνύουν πώς οι διάφορες δομές δεδομένων — μεμονωμένα ή σε συνδυασμό — αξιοποιούνται για την αποτελεσματική επίλυση πρακτικών προβλημάτων προγραμματισμού.

4.1. Η Δομή Δεδομένων των Πλειάδων (Tuples)

Οι πλειάδες (tuples) αποτελούν μία από τις θεμελιώδεις δομές δεδομένων της Python. Ανήκουν στην κατηγορία των ακολουθιών (sequences), όπως και οι λίστες και τα strings, αλλά διαφέρουν σημαντικά από αυτές λόγω του αμετάβλητου χαρακτήρα τους. Χρησιμοποιούνται για την αποθήκευση συλλογών αντικειμένων, που παρουσιάζουν μια σταθερή σχέση μεταξύ τους, σε μια μεταβλητή. Η κατανόηση των πλειάδων είναι απαραίτητη για την αποτελεσματική διαχείριση δεδομένων στην Python, ειδικά σε περιπτώσεις όπου απαιτείται η προστασία της ακεραιότητας των δεδομένων.

4.1.1. Ορισμός και Χαρακτηριστικά

Μια πλειάδα είναι μια **διατεταγμένη** και **αμετάβλητη** (immutable) συλλογή στοιχείων, η οποία περιβάλλεται από παρενθέσεις (). Τα στοιχεία μιας πλειάδας μπορούν να είναι οποιουδήποτε τύπου δεδομένων (integers, strings, floats, ακόμα και άλλες πλειάδες ή λίστες), και μία πλειάδα μπορεί να περιέχει στοιχεία διαφορετικών τύπων ταυτόχρονα.

Παράδειγμα Πλειάδας

```
thistuple = ("μήλο", "μπανάνα", "πορτοκάλι", "σταφύλι")
print(thistuple) # Αποτέλεσμα: ('μήλο', 'μπανάνα', 'πορτοκάλι', 'σταφύλι')
```

Κ 92: Παράδειγμα Πλειάδας

Τα κύρια χαρακτηριστικά που τις διαφοροποιούν από άλλες δομές, όπως οι λίστες, είναι τα εξής:

- **Διάταξη (Ordering)**: Τα στοιχεία μιας πλειάδας διατηρούν συγκεκριμένη σειρά. Η θέση κάθε στοιχείου είναι σταθερή και προσδιορίζεται από έναν δείκτη (index). Η πρόσβαση των στοιχείων πραγματοποιείται μέσω δεικτών (indexing).
- **Αμεταβλητότητα (Immutability)**: Μετά τη δημιουργία μιας πλειάδας, τα στοιχεία της δεν μπορούν να τροποποιηθούν, να προστεθούν ή να αφαιρεθούν από αυτή. Αυτό το χαρακτηριστικό διαφοροποιεί τις πλειάδες από τις λίστες και τις καθιστά ιδανικές για την αποθήκευση δεδομένων που δεν πρέπει να αλλάξουν.
- **Ετερογένεια (Heterogeneity)**: Μια πλειάδα δύναται να περιέχει στοιχεία διαφορετικών τύπων δεδομένων στην ίδια δομή (ακέραιους, συμβολοσειρές, λίστες κ.λπ.).
- **Επιτρέπονται Διπλότυπα**: Τα στοιχεία μπορούν να επαναλαμβάνονται μέσα στην πλειάδα.
- **Ταχύτητα**: Λόγω του αμετάβλητου χαρακτήρα τους, οι πλειάδες είναι υπολογιστικά ταχύτερες από τις λίστες.

4.1.2. Σύγκριση με Λίστες

Αν και οι πλειάδες και οι λίστες μοιράζονται πολλά κοινά χαρακτηριστικά, υπάρχουν σημαντικές διαφορές μεταξύ τους:

Πίνακας 7: Σύγκριση Πλειάδων με Λίστες

Χαρακτηριστικό	Πλειάδες (Tuples)	Λίστες (Lists)
Συντακτικό	Παρενθέσεις ()	Αγκύλες []
Αμεταβλητότητα	Αμετάβλητες	Μεταβλητές
Απόδοση	Ταχύτερες	Πιο αργές
Μνήμη	Λιγότερη	Περισσότερη
Χρήση	Σταθερά δεδομένα	Δυναμικά δεδομένα

Οι πλειάδες είναι προτιμότερες όταν:

- Τα δεδομένα δεν πρέπει να τροποποιηθούν
- Χρειάζεται μεγαλύτερη ταχύτητα εκτέλεσης

4.1.3. Δημιουργία Πλειάδας

Υπάρχουν διάφοροι τρόποι δημιουργίας πλειάδων στην Python:

4.1.3.1 Χρήση Παρενθέσεων

Ο πιο συνηθισμένος τρόπος είναι η χρήση παρενθέσεων:

```
new_tuple = (element1, element2, …,elementN)
```

Παραδείγματα

```
# Κενή πλειάδα
empty_tuple = ()

# Πλειάδα με ένα στοιχείο (προσοχή στο κόμμα)
single_element = (5,)

# Πλειάδα με πολλά στοιχεία
numbers = (1, 2, 3, 4, 5)
mixed = (1, "hello", 3.14, True)
```

Κ 93: Δημιουργία Πλειάδας- χρήση παρενθέσεων

Σημαντική Παρατήρηση: Για τη δημιουργία πλειάδας με ένα μόνο στοιχείο, το κόμμα είναι απαραίτητο. Χωρίς αυτό, οι παρενθέσεις θεωρούνται απλώς ομαδοποίηση έκφρασης.

4.1.3.2 Χωρίς Παρενθέσεις (Tuple Packing)

Η Python επιτρέπει τη δημιουργία πλειάδων χωρίς παρενθέσεις:

```
coordinates = 10, 20, 30
person = "Γιάννης", 25, "Αθήνα"
```

Κ 94: Δημιουργία Πλειάδας χωρίς παρενθέσεις

4.1.3.3 Χρήση του κατασκευαστή tuple()

Ο κατασκευαστής tuple() μετατρέπει άλλες ακολουθίες σε πλειάδες. Μία λίστα, ένα αλφαριθμητικό μπορούν να μετατραπούν σε πλειάδα.

Παραδείγματα

```
# Από λίστα
list_to_tuple = tuple([1, 2, 3, 4])

# Από string
string_to_tuple = tuple("Python")  # Αποτέλεσμα: ('P', 'y', 't', 'h', 'o', 'n')

# Από range
range_to_tuple = tuple(range(5))  # Αποτέλεσμα: (0, 1, 2, 3, 4)
```

Κ 95: Δημιουργία Πλειάδας με χρήση της συνάρτησης tuple()

4.1.4. Πρόσβαση σε στοιχεία Πλειάδας – Δεικτοδότηση Πλειάδων

Η πρόσβαση στα στοιχεία μιας πλειάδας γίνεται με τον ίδιο τρόπο όπως στις λίστες, χρησιμοποιώντας δείκτες (indices). Κάθε στοιχείο της πλειάδας έχει μια συγκεκριμένη θέση που προσδιορίζεται από έναν ακέραιο αριθμό, ο οποίος ονομάζεται δείκτης. Οι δείκτες στην Python ξεκινούν από το μηδέν (0) για το πρώτο στοιχείο και αυξάνονται κατά ένα για κάθε επόμενο στοιχείο. Η Python υποστηρίζει επίσης αρνητική δεικτοδότηση, η οποία επιτρέπει την πρόσβαση στα στοιχεία ξεκινώντας από το τέλος της πλειάδας, με τον δείκτη -1 να αναφέρεται στο τελευταίο στοιχείο, τον -2 στο προτελευταίο, και ούτω καθεξής. Η πρόσβαση σε μεμονωμένο στοιχείο επιτυγχάνεται με τη χρήση του ονόματος της πλειάδας ακολουθούμενου από τον δείκτη σε αγκύλες, με τη σύνταξη `tuple[index]`.

4.1.4.1 Θετική Δεικτοδότηση

Οι δείκτες ξεκινούν από το 0 για το πρώτο στοιχείο.

Παραδείγματα Θετικής Δεικτοδότησης

```
fruits = ("μήλο", "μπανάνα", "πορτοκάλι", "σταφύλι")

first = fruits[0]   # "μήλο"
second = fruits[1]  # "μπανάνα"
last = fruits[3]    # "σταφύλι"
```

Κ 96: Θετική Δεικτοδότηση σε Πλειάδες

4.1.4.2 Αρνητική Δεικτοδότηση

Η αρνητική δεικτοδότηση ξεκινά από το τέλος της πλειάδας.

Παραδείγματα Αρνητικής Δεικτοδότησης

```
fruits = ("μήλο", "μπανάνα", "πορτοκάλι", "σταφύλι")

last = fruits[-1]    # "σταφύλι"
second_last = fruits[-2]  # "πορτοκάλι"
first = fruits[-4]   # "μήλο"
```

Κ 97: Αρνητική Δεικτοδότηση σε Πλειάδες

4.1.4.3 Τεμαχισμός (Slicing)

Όπως και στις λίστες, υποστηρίζεται η εξαγωγή υπο-πλειάδων μέσω της τεχνικής του τεμαχισμού. Η

```
tuple[start:stop:step]
```

γενική σύνταξη του τεμαχισμού είναι:

Όπου:

- το ***start*** υποδεικνύει τη θέση έναρξης, συμπεριλαμβανομένης της θέσης αυτής (προεπιλογή: 0),
- το ***stop*** τη θέση λήξης, μη συμπεριλαμβανομένης της θέσης αυτής (προεπιλογή: len(tuple)),
- το ***step*** το βήμα με το οποίο επιλέγονται τα στοιχεία (προεπιλογή: 1).

Όλες οι παράμετροι είναι προαιρετικές: αν παραλειφθεί το *start*, η εξαγωγή ξεκινά από την αρχή της πλειάδας, αν παραλειφθεί το *stop*, συνεχίζεται μέχρι το τέλος, και αν παραλειφθεί το *step*, η

προεπιλεγμένη τιμή είναι 1. Οι αρνητικές τιμές επιτρέπονται και για τις τρεις παραμέτρους, επιτρέποντας την πλοήγηση από το τέλος προς την αρχή της πλειάδας. Το αποτέλεσμα του τεμαχισμού είναι πάντα μια νέα πλειάδα, ανεξάρτητα από το πλήθος των στοιχείων που εξάγονται.

Παραδείγματα Τεμαχισμού

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# Βασικός τεμαχισμός [start:stop]
subset1 = numbers[2:5]     # (2, 3, 4)
print(subset1)
subset2 = numbers[:4]      # (0, 1, 2, 3)
print(subset2)
subset3 = numbers[6:]      # (6, 7, 8, 9)
print(subset3)

# Τεμαχισμός με βήμα [start:stop:step]
even = numbers[::2]        # (0, 2, 4, 6, 8)

odd = numbers[1::2] # (1, 3, 5, 7, 9) - από θέση 1, κάθε 2ο στοιχείο

reverse = numbers[::-1]    # (9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
```

Κ. 98 Παραδείγματα τεμαχισμού πλειάδας

4.1.5. Τροποποίηση στοιχείων πλειάδας

Όπως αναφέραμε, οι πλειάδες χαρακτηρίζονται από την ιδιότητα του **αμετάβλητου (immutability)**. Αυτό σημαίνει ότι μετά τη δημιουργία τους, δεν επιτρέπεται η προσθήκη, η αφαίρεση ή η τροποποίηση των στοιχείων τους. Η ιδιότητα αυτή διασφαλίζει την ακεραιότητα των δεδομένων, καθιστώντας τις πλειάδες ιδανικές για την αποθήκευση σταθερών πληροφοριών που δεν πρέπει να αλλοιωθούν κατά την εκτέλεση του προγράμματος.

Ωστόσο, στην πράξη υπάρχει μια εναλλακτική μέθοδος (workaround) για την τροποποίηση των τιμών τους. Οι προγραμματιστές μπορούν να μετατρέψουν προσωρινά την πλειάδα σε **λίστα (list)** χρησιμοποιώντας τη συνάρτηση list(), να εκτελέσουν τις επιθυμητές αλλαγές στη λίστα και, στη συνέχεια, να τη μετατρέψουν ξανά σε πλειάδα με τη συνάρτηση tuple(). Με αυτόν τον τρόπο, επιτυγχάνεται η "έμμεση" ενημέρωση των δεδομένων, παρά τους περιορισμούς της αρχικής δομής.

Παράδειγμα:

```
# Αρχική πλειάδα
fruits = ("μήλο", "μπανάνα", "κεράσι")
```

```python
# Μετατροπή σε λίστα για να γίνει η αλλαγή
y = list(fruits)
y[1] = "ακτινίδιο"

# Μετατροπή ξανά σε πλειάδα
fruits = tuple(y)

print(fruits) # Έξοδος: ('μήλο', 'ακτινίδιο', 'κεράσι')
```

Κ 99: Τροποποίηση στοιχείων πλειάδας

4.1.6. Αποσυσκευασία (Unpacking)

Η αποσυσκευασία αποτελεί μία από τις πλέον ισχυρές λειτουργίες της Python, καθώς επιτρέπει στους προγραμματιστές να αναθέτουν τα στοιχεία μιας πλειάδας σε μεμονωμένες μεταβλητές με μία μόνο εντολή. Με αυτόν τον τρόπο, αποφεύγεται η ανάγκη για πολλαπλές γραμμές κώδικα και βελτιώνεται η αναγνωσιμότητα του προγράμματος.

```python
student = ("Μαρία", 22, "Πληροφορική")
name, age, department = student
print(f"name: {name}  age: {age}  department: {department}")
```

Κ 100: Πλειάδες - Αποσυσκευασία (Unpacking)

Για την επιτυχή εκτέλεση της αποσυσκευασίας, απαιτείται ο αριθμός των μεταβλητών στην αριστερή πλευρά της εξίσωσης να είναι **ακριβώς ίσος** με τον αριθμό των στοιχείων που περιέχονται στην πλειάδα (στο παραπάνω παράδειγμα ο αριθμός των μεταβλητών είναι 3- name, age, department) .

4.1.6.1 Εξειδικευμένες Τεχνικές Αποσυσκευασίας

Οι προγραμματιστές χρησιμοποιούν συχνά τις παρακάτω τεχνικές για να διαχειριστούν σύνθετα σύνολα δεδομένων:

Παράλειψη Στοιχείων με τη χρήση του Underscore (_): Στις περιπτώσεις όπου ορισμένα δεδομένα της πλειάδας κρίνονται ως περιττά για τη συγκεκριμένη λειτουργία, χρησιμοποιείται το σύμβολο _. Αυτό υποδηλώνει στην Python ότι η αντίστοιχη τιμή πρέπει να αγνοηθεί.

```python
user = ("Γιώργος", 25, "Αθήνα", "Προγραμματιστής")
name, _, _, profession = user
```

Κ 101: Αποσυσκευασία - Παράλειψη Στοιχείων με τη χρήση του Underscore (_)

Δυναμική Συλλογή Στοιχείων με τον Αστερίσκο (*): Αν η πλειάδα έχει πολλά στοιχεία και επιθυμούν να απομονώσουν τα πρώτα στοιχεία και να ομαδοποιήσουν τα υπόλοιπα, χρησιμοποιούν τον τελεστή *. Αυτό έχει ως αποτέλεσμα τη δημιουργία μιας νέας λίστας που περιλαμβάνει όλα τα πλεονάζοντα στοιχεία.

```
grades = (10, 9, 8, 7, 6, 5)
first, second, *others = grades
```

Στο παραπάνω παράδειγμα, το first παίρνει το 10, το second το 9, και η μεταβλητή others γίνεται μια **λίστα** που περιέχει όλα τα υπόλοιπα στοιχεία [8, 7, 6, 5].

4.1.7. Λειτουργίες και Μέθοδοι Πλειάδων

Παρόλο που οι πλειάδες είναι αμετάβλητες δομές, η Python παρέχει τη δυνατότητα χρήσης συγκεκριμένων τελεστών για τη δημιουργία νέων πλειάδων από υπάρχουσες, καθώς και μεθόδους για την ανάλυση των δεδομένων τους.

4.1.7.1 Ειδικοί Τελεστές Πλειάδων

4.1.7.1.1 Συνένωση (Concatenation)

Δύο ή περισσότερες πλειάδες μπορούν να ενωθούν χρησιμοποιώντας τον **τελεστή +**. Είναι σημαντικό να τονιστεί ότι η λειτουργία αυτή **δεν τροποποιεί** τις αρχικές πλειάδες, αλλά δημιουργεί μια νέα, η οποία περιέχει τα στοιχεία όλων των προηγούμενων.

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
combined = tuple1 + tuple2  # (1, 2, 3, 4, 5, 6)
```

Κ 102: Συνένωση (Concatenation) Πλειάδων με τον τελεστή +

4.1.7.1.2 Επανάληψη στοιχείων πλειάδας (Repetition)

Ο τελεστής * χρησιμοποιείται όταν απαιτείται η επανάληψη των στοιχείων μιας πλειάδας έναν συγκεκριμένο αριθμό φορών. Όπως και στη συνένωση, το αποτέλεσμα είναι μια νέα πλειάδα.

```
original = (1, 2)
repeated = original * 3  # (1, 2, 1, 2, 1, 2)
```

Κ 103: Επανάληψη (Repetition) στοιχείων πλειάδας

4.1.7.1.3 Έλεγχος Μέλους (Membership)

Οι τελεστές in και not in ελέγχουν την ύπαρξη στοιχείου:

```python
fruits = ("μήλο", "μπανάνα", "πορτοκάλι")

print("μήλο" in fruits)      # True
print("λεμόνι" in fruits)    # False
print("λεμόνι" not in fruits) # True
```
Κ 104: Έλεγχος Μέλους (Membership)

4.1.7.2 Μέθοδοι Πλειάδων

Λόγω της αμεταβλητότητάς τους, οι πλειάδες διαθέτουν μόνο δύο μεθόδους:

4.1.7.2.1 Μέθοδος count()

Επιστρέφει τον αριθμό των εμφανίσεων ενός στοιχείου. Η σύνταξή της είναι:

```python
tuple.count(value)
```

Παράδειγμα

```python
numbers = (1, 2, 3, 2, 4, 2, 5)
count_of_twos = numbers.count(2)  # 3
```
Κ 105: Μέθοδος count() στις πλειάδες

4.1.7.2.2 Μέθοδος index()

Επιστρέφει τον δείκτη της πρώτης εμφάνισης ενός στοιχείου. Η σύνταξή της είναι:

```python
tuple.index(value, start, end)
```

Όπου:

- **value:** Η τιμή που αναζητείται (υποχρεωτική παράμετρος)
- **start** (προαιρετικό): Ο δείκτης από τον οποίο ξεκινά η αναζήτηση
- **end** (προαιρετικό): Ο δείκτης στον οποίο σταματά η αναζήτηση (δεν συμπεριλαμβάνεται)

Εξαίρεση: Αν το στοιχείο δεν βρεθεί, προκαλείται ValueError.

Παράδειγμα

```python
fruits = ("μήλο", "μπανάνα", "πορτοκάλι", "μπανάνα")
position = fruits.index("μπανάνα", 2)  # 3 (αναζήτηση από θέση 2 και μετά)
print(position)
```

4.1.7.3 Ενσωματωμένες Συναρτήσεις

Οι ακόλουθες ενσωματωμένες συναρτήσεις της Python είναι χρήσιμες για την εργασία με πλειάδες:

4.1.7.3.1 len()

Επιστρέφει τον αριθμό των στοιχείων.

Παράδειγμα

```
numbers = (1, 2, 3, 4, 5)
length = len(numbers)  # 5
```

4.1.7.3.2 max() και min()

Επιστρέφουν το μέγιστο και ελάχιστο στοιχείο της πλειάδας.

Παράδειγμα

```
numbers = (5, 2, 8, 1, 9)
maximum = max(numbers)  # 9
minimum = min(numbers)  # 1
```

4.1.7.3.3 sum()

Επιστρέφει το άθροισμα των στοιχείων (μόνο για αριθμητικά στοιχεία) της πλειάδας:

Παράδειγμα

```
numbers = (1, 2, 3, 4, 5)
total = sum(numbers)  # 15
```

4.1.8. Διάσχιση στοιχείων Πλειάδας (Loop Tuples)

Σε αυτή την ενότητα, θα εξεταστούν οι μέθοδοι με τις οποίες οι προγραμματιστές προσεγγίζουν τα στοιχεία μιας πλειάδας (tuple) μέσω δομών επανάληψης (loops). Οι διαδικασίες αυτές θεωρούνται θεμελιώδεις για την επεξεργασία ή την προβολή συνόλων δεδομένων που έχουν αποθηκευτεί σε αυτή τη δομή.

4.1.8.1 Άμεση Επανάληψη στα Στοιχεία (For Loop)

Αποτελεί την πλέον διαδεδομένη μέθοδο. Οι προγραμματιστές αξιοποιούν έναν βρόχο for προκειμένου να διατρέξουν κάθε στοιχείο της πλειάδας απευθείας, χωρίς την ανάγκη διαχείρισης δεικτών.

```
thistuple = ("μήλο", "μπανάνα", "κεράσι")
for x in thistuple:
  print(x)
```

Κ 106: Άμεση Επανάληψη στα Στοιχεία (For Loop)

4.1.8.2 Επανάληψη μέσω των Δεικτών (Index Numbers)

Σε περιπτώσεις όπου κρίνεται απαραίτητη η γνώση της ακριβούς θέσης (index) κάθε στοιχείου, οι προγραμματιστές συνδυάζουν τις συναρτήσεις range() και len(). Με αυτόν τον τρόπο, αποκτούν πρόσβαση σε κάθε στοιχείο αναφερόμενοι στον δείκτη του.

```
thistuple = ("μήλο", "μπανάνα", "κεράσι")
for i in range(len(thistuple)):
  print(thistuple[i])
```

Κ 107: Επανάληψη μέσω των Δεικτών (Index Numbers)

4.1.8.3 Χρήση της Δομής While

Η επανάληψη μπορεί επίσης να υλοποιηθεί με τη χρήση ενός βρόχου while. Στην περίπτωση αυτή, οι προγραμματιστές ορίζουν έναν μετρητή (index variable) που εκκινεί από το 0 και αυξάνεται διαδοχικά, μέχρις ότου εξαντληθεί το σύνολο των στοιχείων της πλειάδας.

```
thistuple = ("μήλο", "μπανάνα", "κεράσι")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Κ 108: Χρήση της Δομής While

4.1.8.4 Παράδειγμα Εφαρμογής

Ας υποτεθεί ότι οι αναλυτές διαθέτουν μια πλειάδα με αριθμητικά δεδομένα και επιθυμούν να απομονώσουν μόνο όσα υπερβαίνουν ένα συγκεκριμένο όριο:

```
numbers = (10, 25, 5, 40, 15)
for n in numbers:
    if n > 20:
        print(f"Εντοπίστηκε τιμή άνω του ορίου: {n}")
```

4.1.8.5 Ασκήσεις Διάσχισης στοιχείων Πλειάδας

Άσκηση 1: Δίνεται η πλειάδα colors = ("κόκκινο", "πράσινο", "μπλε").

1. Να χρησιμοποιηθεί ένας βρόχος for για την εκτύπωση κάθε χρώματος.
2. Να υλοποιηθεί η ίδια διαδικασία με τη χρήση της δομής while.

Άσκηση 2: Δίνεται η πλειάδα grades = (8, 9, 7, 10).

Να αξιοποιηθεί η επανάληψη μέσω δεικτών (range(len())) ώστε να εμφανιστεί το μήνυμα: "Ο βαθμός στη θέση [δείκτης] είναι [βαθμός]".

4.1.9. Εμβάθυνση στις Πλειάδες (Tuples)

4.1.9.1 Η Εσωτερική Λειτουργία της Αμεταβλητότητας (Immutability)

Όταν ορίζουμε μια πλειάδα, η Python δεσμεύει ένα στατικό τμήμα μνήμης. Σε αντίθεση με τις λίστες, οι οποίες είναι "υπερ-δεσμευμένες" (over-allocated) για να επιτρέπουν μελλοντικές προσθήκες (`append`), οι πλειάδες καταλαμβάνουν ακριβώς όσο χώρο χρειάζονται τα στοιχεία τους.

4.1.9.2 Το "Παράδοξο" των Σύνθετων Πλειάδων

Μια πλειάδα είναι αμετάβλητη ως προς τα **αντικείμενα** που περιέχει, όχι απαραίτητα ως προς τις **τιμές** αυτών των αντικειμένων, αν αυτά είναι μεταβλητά.

```
# Πλειάδα που περιέχει μια λίστα
mixed_tuple = (1, 2, [3, 4])

# mixed_tuple[2] = [5, 6]  --> ERROR (Δεν αλλάζει η αναφορά της λίστας)
mixed_tuple[2][0] = 99     # SUCCESS (Η λίστα μέσα στην πλειάδα μπορεί να αλλάξει!)
print(mixed_tuple)         # (1, 2, [99, 4])
```

Κ 109: Το "Παράδοξο" των Σύνθετων Πλειάδων

4.2. Η Δομή Δεδομένων των Συνόλων (Sets)

Τα σύνολα (sets) αποτελούν μια θεμελιώδη δομή δεδομένων στην Python που βασίζεται στη μαθηματική έννοια του συνόλου. Πρόκειται για μια μη διατεταγμένη συλλογή μοναδικών στοιχείων, η οποία υποστηρίζει τις βασικές πράξεις της θεωρίας συνόλων όπως η ένωση, η τομή, η διαφορά και η συμμετρική διαφορά. Τα σύνολα χρησιμοποιούνται ευρέως σε προβλήματα που απαιτούν την εξάλειψη διπλότυπων στοιχείων, τον έλεγχο μέλους με υψηλή ταχύτητα, ή την εκτέλεση μαθηματικών πράξεων μεταξύ συλλογών δεδομένων.

4.2.1. Ορισμός και Χαρακτηριστικά

Ένα σύνολο είναι μια **μη διατεταγμένη** και **μεταβλητή** συλλογή **μοναδικών** στοιχείων. Τα στοιχεία ενός συνόλου πρέπει να είναι αμετάβλητα (immutable), πράγμα που σημαίνει ότι μπορούν να είναι αριθμοί, strings, ή tuples, αλλά όχι λίστες ή άλλα σύνολα. Οι προγραμματιστές χρησιμοποιούν τα σύνολα κυρίως όταν η σειρά των δεδομένων δεν έχει σημασία, αλλά η αποφυγή διπλότυπων εγγραφών είναι κρίσιμη.

Παράδειγμα

```
numbers = {1, 2, 3, 4, 5}
```

Σημείωση: Οι τιμές True και 1 θεωρούνται ίδιες τιμές σε σύνολα και αντιμετωπίζονται ως διπλότυπα:

```
thisset = {"μήλο", "μπανάνα", "πορτοκάλι", True, 1, 2}
print(thisset) #: {True, 2, 'μπανάνα', 'μήλο', 'πορτοκάλι'}
# Το 1 δεν εμφανίζεται γιατί θεωρείται ίδιο με το True
```

Σημείωση: Οι τιμές False και 0 θεωρούνται ίδιες τιμές σε σύνολα και αντιμετωπίζονται ως διπλότυπα:

```
thisset = {"μήλο", "μπανάνα", "πορτοκάλι", False, True, 0}
print(thisset) #{False, True, 'πορτοκάλι', 'μπανάνα', 'μήλο'}
# Το 0 δεν εμφανίζεται γιατί θεωρείται ίδιο με το False
```

Βασικά Χαρακτηριστικά:

1. **Μοναδικότητα (Uniqueness)**: Κάθε στοιχείο εμφανίζεται μόνο μία φορά στο σύνολο. Αν προστεθεί ένα στοιχείο που ήδη υπάρχει, το σύνολο δεν αλλάζει.
2. **Μη Διατεταγμένα (Unordered)**: Τα στοιχεία δεν διατηρούν συγκεκριμένη σειρά. Η σειρά με την οποία προστίθενται τα στοιχεία δεν εγγυάται τη σειρά με την οποία ανακτώνται. Δεν υποστηρίζουν δείκτες (indexing) ή τεμαχισμό (slicing).
3. **Μεταβλητότητα (Mutability)**: Τα σύνολα μπορούν να τροποποιηθούν μετά τη δημιουργία τους (προσθήκη ή αφαίρεση στοιχείων).
4. **Αμεταβλητότητα Στοιχείων**: Τα στοιχεία που περιέχονται σε ένα σύνολο πρέπει να είναι hashable (αμετάβλητα).

5. **Απόδοση**: Ο έλεγχος μέλους (in) είναι εξαιρετικά γρήγορος.

4.2.2. Σύγκριση με Λίστες και Πλειάδες

Τα σύνολα διαφέρουν σημαντικά από τις άλλες ενσωματωμένες δομές δεδομένων της Python. Σε αντίθεση με τις λίστες και τις πλειάδες, τα σύνολα δεν διατηρούν τη σειρά εισαγωγής. Ενώ οι λίστες είναι ιδανικές για αποθήκευση δεδομένων με συγκεκριμένη σειρά και οι πλειάδες για προστασία από αλλαγές, τα σύνολα επιλέγονται για ταχύτητα στον έλεγχο ύπαρξης στοιχείων και για μαθηματικές πράξεις. Αναλυτικά:

Πίνακας 8: Σύγκριση Συνόλων με Λίστες και Πλειάδες

Χαρακτηριστικό	Σύνολα (Sets)	Λίστες (Lists)	Πλειάδες (Tuples)
Συντακτικό	Άγκιστρα {} ή set()	Αγκύλες []	Παρενθέσεις ()
Διάταξη	Μη διατεταγμένα	Διατεταγμένες	Διατεταγμένες
Μοναδικότητα	Μοναδικά	Επιτρέπονται διπλότυπα	Επιτρέπονται διπλότυπα
Μεταβλητότητα	Μεταβλητά	Μεταβλητές	Αμετάβλητες
Πρόσβαση	Δεν υπάρχει index	Με index	Με index
Χρήση	Μαθηματικές πράξεις	Ακολουθίες	Σταθερά δεδομένα

Τα σύνολα χρησιμοποιούνται:

- Όταν απαιτείται εξάλειψη διπλότυπων
- Όταν η σειρά δεν έχει σημασία
- Όταν χρειάζονται μαθηματικές πράξεις συνόλων
- Όταν απαιτείται γρήγορος έλεγχος μέλους

4.2.3. Δημιουργία Συνόλου

Υπάρχουν διάφοροι τρόποι δημιουργίας συνόλων στην Python:

4.2.3.1 Χρήση Αγκυλών

Τα σύνολα δημιουργούνται με **αγκύλες { }** που περιέχουν στοιχεία διαχωρισμένα με κόμματα:

```
# Σύνολο με αριθμούς
```

```python
numbers = {1, 2, 3, 4, 5}

# Σύνολο με strings
fruits = {"μήλο", "πορτοκάλι", "μπανάνα"}

# Μικτό σύνολο
mixed = {1, "hello", 3.14, True}

# Αυτόματη αφαίρεση διπλότυπων
duplicates = {1, 2, 2, 3, 3, 3, 4}  # Αποτέλεσμα: {1, 2, 3, 4}
```

Κ 110: Δημιουργία Συνόλου με τη χρήση αγκυλών { }

Σημείωση: Για τη δημιουργία κενού συνόλου, οι προγραμματιστές χρησιμοποιούν αποκλειστικά τη set() που αναφέρεται παρακάτω, καθώς τα κενά άγκιστρα `{}` δημιουργούν λεξικό (dictionary).

4.2.3.2 Χρήση του Κατασκευαστή set()

Ο κατασκευαστής set() μπορεί να δημιουργήσει σύνολα από άλλες ακολουθίες:

```python
# Κενό σύνολο
empty_set = set()

# Από λίστα
list_to_set = set([1, 2, 3, 2, 1])  # {1, 2, 3}

# Από string (κάθε χαρακτήρας γίνεται στοιχείο)
string_to_set = set("hello")  # {'h', 'e', 'l', 'o'}

# Από tuple
tuple_to_set = set((1, 2, 3, 4))  # {1, 2, 3, 4}

# Από range
range_to_set = set(range(5))  # {0, 1, 2, 3, 4}
```

Κ 111: Δημιουργία Συνόλου με τη χρήση του κατασκευαστή set()

4.2.4. Πρόσβαση στα στοιχεία Συνόλου

Σε αντίθεση με τις λίστες και τις πλειάδες, τα σύνολα δεν υποστηρίζουν πρόσβαση σε μεμονωμένα στοιχεία μέσω ευρετηρίου (index), καθώς αποτελούν μη διατεταγμένες (unordered) δομές δεδομένων. Ωστόσο, είναι δυνατή η διάσχιση όλων των στοιχείων ενός συνόλου μέσω βρόχου for, καθώς και ο έλεγχος ύπαρξης ενός στοιχείου με τη χρήση του τελεστή in, τα οποία θα αναλυθούν σε επόμενες παραγράφους.

4.2.5. Τροποποίηση στοιχείων Συνόλου

Σε αντίθεση με τις πλειάδες, τα σύνολα (sets) στην Python είναι **μεταβλητές** (mutable) δομές δεδομένων. Αυτό σημαίνει ότι μετά τη δημιουργία τους, επιτρέπεται η προσθήκη νέων στοιχείων καθώς και η αφαίρεση υπαρχόντων στοιχείων. Ωστόσο, υπάρχει ένας σημαντικός περιορισμός: τα ίδια τα στοιχεία που περιέχονται σε ένα σύνολο πρέπει να είναι **αμετάβλητου** (immutable) **τύπου,** δεν τροποποιούνται. Για τον λόγο αυτό, ένα σύνολο μπορεί να περιέχει ακεραίους, συμβολοσειρές και πλειάδες, αλλά δεν μπορεί να περιέχει λίστες ή άλλα σύνολα.

4.2.6. Αποσυσκευασία (Unpacking)

Η αποσυσκευασία υποστηρίζεται και στα σύνολα, με τη διαφορά ότι, λόγω της μη διατεταγμένης φύσης τους, οι μεταβλητές θα λάβουν τα στοιχεία με τυχαία σειρά.

```
a, b, c = {"X", "Y", "Z"}
```

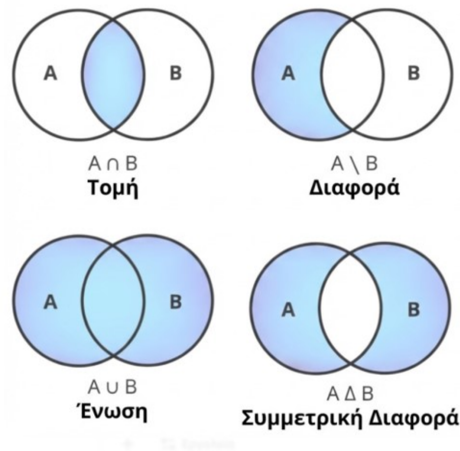
4.2.7. Λειτουργίες και Μέθοδοι Συνόλων

Τα σύνολα υποστηρίζουν μια πληθώρα λειτουργιών που βασίζονται στη θεωρία συνόλων και μεθόδων που διευκολύνουν τη διαχείριση των δεδομένων.

4.2.7.1 Λειτουργίες Συνόλων

Τα σύνολα στην Python υποστηρίζουν τις βασικές μαθηματικές πράξεις της θεωρίας συνόλων. Κάθε πράξη μπορεί να εκτελεστεί είτε με τελεστές είτε με μεθόδους. Οι λειτουργίες αφορούν τις πράξεις:

- **Ένωση (union()):** Συνδυασμός όλων των στοιχείων.
- **Τομή (intersection()):** Εντοπισμός κοινών στοιχείων.
- **Διαφορά (difference()):** Στοιχεία που υπάρχουν μόνο στο ένα σύνολο



Εικόνα 14: Πράξεις Συνόλων

4.2.7.1.1 Ένωση (Union)

Η ένωση δύο συνόλων επιστρέφει ένα νέο σύνολο που περιέχει **όλα τα στοιχεία** και από τα δύο σύνολα. Η σύνταξη της ένωσης είναι:

```
C = A | B
C = A.union(B)
```

Σύνδεση περισσότερων συνόλων: Για την ένωση περισσότερων συνόλων B, D, E η σύνταξη είναι:

```
C = A.union(B, D, E)
Ή
C = A | B | D | E
```

Παράδειγμα

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
C = A | B
print(C)  # {1, 2, 3, 4, 5, 6}

# Με πολλά σύνολα
D = {7, 8}
E = A.union(B, D)
print(E)  # {1, 2, 3, 4, 5, 6, 7, 8}
```

Κ 112: Ένωση (Union) Συνόλων

4.2.7.1.2 Τομή (Intersection) - & ή .intersection()

Η τομή επιστρέφει ένα νέο σύνολο με τα **κοινά στοιχεία** των συνόλων. Η σύνταξη είναι:

```
C = A & B
C = A.intersection(B)

C = A & B & D & E
C = A.intersection(B, D, E)
```

Παράδειγμα

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
C = A & B
print(C)  # {3, 4}

# Τομή πολλών συνόλων
D = {4, 5, 7}
E = A.intersection(B, D)
print(E)  # {4}
```

Κ 113: Τομή (Intersection) Συνόλων

4.2.7.1.3 Διαφορά (Difference) - - ή .difference()

Η διαφορά επιστρέφει τα στοιχεία που υπάρχουν στο πρώτο σύνολο αλλά **όχι** στο δεύτερο. Η σύνταξη είναι:

```
C = A - B
C = A.difference(B)
```

Παράδειγμα

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
C = A - B
print(C)  # {1, 2}

D = B - A
print(D)  # {5, 6}
```

Κ 114: Η Διαφορά (Difference) – Συνόλων

4.2.7.1.4 Συμμετρική Διαφορά (Symmetric Difference) ^ ή .symmetric_difference()

Επιστρέφει στοιχεία που υπάρχουν στο ένα **ή** στο άλλο σύνολο, αλλά **όχι και στα δύο**. Η σύνταξη είναι:

```
C = A ^ B
C = A.symmetric_difference(B)
```

Παράδειγμα

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
C = A ^ B
print(C)  # {1, 2, 5, 6}
```

Παρατήρηση: Σε αντίθεση με τις union(), intersection(), difference(), η μέθοδος symmetric_difference() δέχεται μόνο ένα όρισμα.

4.2.7.1.5 Υποσύνολο (Subset) - .issubset() ή <=

Ελέγχει αν όλα τα στοιχεία του πρώτου συνόλου υπάρχουν στο δεύτερο. Χρησιμοποιούνται ο τελεστής <= ή η μέθοδος .issubset() :

```
A = {1, 2, 3}
B = {1, 2, 3, 4, 5}

print(A.issubset(B))  # True
print(A <= B)         # True
print(B.issubset(A))  # False
```

Κ 115: Υποσύνολο (Subset) - .issubset() ή <=

4.2.7.1.6 Υπερσύνολο (Superset) - .issuperset() ή >=

Ελέγχει αν το πρώτο σύνολο περιέχει όλα τα στοιχεία του δεύτερου. Χρησιμοποιούνται ο τελεστής >= ή η μέθοδος . issuperset () :

```
A = {1, 2, 3, 4, 5}
B = {2, 3}

print(A.issuperset(B))  # True
print(A >= B)           # True
```

Κ 116: Υπερσύνολο (Superset) - .issuperset() ή >=

4.2.7.1.7 Ξένα Σύνολα (Disjoint) - .isdisjoint()

Ελέγχει αν δύο σύνολα **δεν έχουν κοινά στοιχεία**. Χρησιμοποιείται η μέθοδος . isdisjoint():

```
A = {1, 2, 3}
```

```
B = {4, 5, 6}
C = {3, 4, 5}

print(A.isdisjoint(B))  # True (δεν έχουν κοινά)
print(A.isdisjoint(C))  # False (έχουν κοινό το 3)
```

Κ 117: Ξένα Σύνολα (Disjoint) - .isdisjoint()

4.2.7.2 Μέθοδοι Συνόλων

4.2.7.2.1 Προσθήκη Στοιχείων

Μέθοδος add()

Η μέθοδος add() προσθέτει ένα μεμονωμένο στοιχείο στο σύνολο. Η σύνταξή της είναι:

```
set.add(element)
```

Παράδειγμα:

```
fruits = {"μήλο", "μπανάνα"}
fruits.add("πορτοκάλι")
# Αποτέλεσμα: {"μήλο", "μπανάνα", "πορτοκάλι"}

# Αν το στοιχείο υπάρχει ήδη, δεν αλλάζει τίποτα
fruits.add("μήλο")
# Αποτέλεσμα: {"μήλο", "μπανάνα", "πορτοκάλι"}
```

Κ 118: Προσθήκη στοιχείων με την χρήση της μεθόδου add

Μέθοδος update()

Η μέθοδος update() προσθέτει πολλαπλά στοιχεία. Η σύνταξή της είναι:

```
set.update(iterable1, iterable2, ...)
```

Παράδειγμα

```
numbers = {1, 2, 3}
numbers.update([4, 5, 6])
# Αποτέλεσμα: {1, 2, 3, 4, 5, 6}

# Με πολλαπλά arguments
numbers.update([7, 8], {9, 10}, (11, 12))
# Αποτέλεσμα: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

Κ 119: Προσθήκη στοιχείων με την χρήση της μεθόδου update

4.2.7.2.2 Αφαίρεση Στοιχείων

Μέθοδος remove()

Αφαιρεί ένα στοιχείο από το σύνολο. Αν το στοιχείο δεν υπάρχει, προκαλείται KeyError:

```
fruits = {"μήλο", "μπανάνα", "πορτοκάλι"}
fruits.remove("μπανάνα")
# Αποτέλεσμα: {"μήλο", "πορτοκάλι"}
```

Κ 120: Αφαίρεση στοιχείου με την χρήση της μεθόδου remove

```
fruits = {"μήλο", "μπανάνα", "πορτοκάλι"}
# Θα προκαλέσει KeyError
fruits.remove("λεμόνι")
```

Κ 121: Αφαίρεση στοιχείου με την χρήση της μεθόδου remove, αν το στοιχείο δεν υπάρχει προκαλείται KeyError

```
Traceback (most recent call last):
  File "c:\Users\[redacted]\python\set\set.py", line 68, in <module>
    fruits.remove("λεμόνι")
    ~~~~~~~~~~~~~^^^^^^^^^^
```

Κ 1: Το KeyError που προκαλείται όταν αφαιρείται στοιχείο που δεν υπάρχει στο σύνολο με την μέθοδο remove

Μέθοδος discard()

Αφαιρεί ένα στοιχείο από το σύνολο. Αν το στοιχείο δεν υπάρχει, δεν συμβαίνει τίποτα. Η σύνταξη είναι:

```
set.discard(element)
```

Παράδειγμα

```
fruits = {"μήλο", "μπανάνα", "πορτοκάλι"}
fruits.discard("μπανάνα")
# Αποτέλεσμα: {"μήλο", "πορτοκάλι"}

# Δεν προκαλεί σφάλμα
fruits.discard("λεμόνι")
# Αποτέλεσμα: {"μήλο", "πορτοκάλι"}
```

Μέθοδος pop()

Αφαιρεί και επιστρέφει ένα τυχαίο στοιχείο. Αν το σύνολο είναι κενό, προκαλείται KeyError. Η σύνταξη είναι:

```
set.pop()
```

Παράδειγμα

```
#pop
numbers = {1, 2, 3, 4, 5}
element = numbers.pop()
print(f"Αφαιρέθηκε: {element}")
# Το numbers έχει πλέον 4 στοιχεία
```

Μέθοδος clear()

Αφαιρεί όλα τα στοιχεία από το σύνολο. Η σύνταξη είναι:

```
set.clear()
```

Παράδειγμα

```
#clear
numbers = {1, 2, 3, 4, 5}
numbers.clear()
# Αποτέλεσμα: set() (κενό σύνολο)
```

4.2.7.2.3 Η Μέθοδος `copy()`

Η μέθοδος `copy()` δημιουργεί ένα **αντίγραφο (copy)** του συνόλου. Αυτό σημαίνει ότι το νέο σύνολο είναι ένα διαφορετικό αντικείμενο στη μνήμη, το οποίο όμως περιέχει τα ίδια στοιχεία με το αρχικό. Οποιαδήποτε αλλαγή στο νέο σύνολο δεν επηρεάζει το αρχικό.

Παραδείγματα

```
original_set = {"Python", "Java", "C++"}

# Δημιουργία ανεξάρτητου αντιγράφου
new_set = original_set.copy()

# Προσθήκη στοιχείου μόνο στο αντίγραφο
new_set.add("SQL")

print(original_set) # Έξοδος: {'Python', 'Java', 'C++'}
print(new_set)      # Έξοδος: {'Python', 'Java', 'C++', 'SQL'}
```

Κ 122: Η μέθοδος copy() των Συνόλων

4.2.7.3 Ο Τελεστής in (Έλεγχος Συμμετοχής)

Ο τελεστής in αποτελεί το βασικό εργαλείο των προγραμματιστών για την επαλήθευση της ύπαρξης ενός στοιχείου εντός του συνόλου.

- **Αποδοτικότητα:** Στα σύνολα, ο έλεγχος με το in είναι εξαιρετικά ταχύς (σχεδόν ακαριαίος), ανεξάρτητα από το μέγεθος του συνόλου, λόγω της εσωτερικής δομής κατακερματισμού (hashing) που χρησιμοποιεί η Python.
- **Αποτέλεσμα:** Επιστρέφει μια λογική τιμή (True ή False).

Παράδειγμα

```
colors = {"κόκκινο", "πράσινο", "μπλε"}
is_present = "πράσινο" in colors  # Αποτέλεσμα: True
```

Κ 123: Ο Τελεστής in (Έλεγχος Συμμετοχής) στα Σύνολα

4.2.8. Διάσχιση στοιχείων Συνόλου (Iteration - Loop Sets)

Η διάσχιση στοιχείων ενός συνόλου πραγματοποιείται συνήθως με τον βρόχο `for`, ο οποίος διατρέχει τα στοιχεία με τη σειρά όπως η Python τα έχει οργανώσει εσωτερικά στη μνήμη.

```
fruits = {"μήλο", "μπανάνα", "πορτοκάλι"}
for fruit in fruits:
    print(fruit)

# Η σειρά δεν είναι εγγυημένη
```

Κ 124: Επανάληψη σε Σύνολα (Loop Sets)

4.2.9. Ασκήσεις Συνόλων

Άσκηση 3: Φιλτράρισμα Δεδομένων: Δίνονται οι παρακάτω λίστες με αριθμούς:

```
lista1 = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]
lista2 = [2, 4, 6, 8, 10, 12, 14]
lista3 = [1, 3, 5, 7, 9, 11, 13]
```

Χρησιμοποιήστε σύνολα για να βρείτε:

1. Τους μοναδικούς αριθμούς από την lista1
2. Αριθμούς που υπάρχουν στις lista1 και lista2
3. Αριθμούς που υπάρχουν στην lista1 αλλά ΌΧΙ στην lista3

4.3. Η Δομή Δεδομένων των Λεξικών (Dictionaries)

4.3.1. Ορισμός και Χαρακτηριστικά

Τα λεξικά (dictionaries) αποτελούν μία από τις πιο σημαντικές και ευέλικτες δομές δεδομένων στην Python. Πρόκειται για μία απεικόνιση κλειδιού-τιμής (key-value mapping) που επιτρέπει την αποθήκευση δεδομένων με τρόπο που διευκολύνει την άμεση πρόσβαση μέσω μοναδικών αναγνωριστικών.

Ένα λεξικό στην Python ορίζεται ως μία **μεταβλητή συλλογή** (mutable collection) που αποθηκεύει ζεύγη κλειδιών-τιμών, όπου:

- Κάθε **κλειδί** (key) είναι μοναδικό και αμετάβλητο (immutable)
- Κάθε **τιμή** (value) μπορεί να είναι οποιουδήποτε τύπου δεδομένων
- Η απεικόνιση είναι **μονοσήμαντη**: κάθε κλειδί αντιστοιχεί σε ακριβώς μία τιμή

Παράδειγμα

```
# Παράδειγμα
φοιτητής = {
    "όνομα": "Μαρία",
    "ηλικία": 21,
    "βαθμός": 8.5
}
```

Βασικά Χαρακτηριστικά

1. **Μεταβλητότητα (Mutability)**
   - Τα λεξικά είναι μεταβλητές δομές δεδομένων
   - Επιτρέπεται η προσθήκη, τροποποίηση και διαγραφή ζευγών κλειδιού-τιμής
2. **Μη Ταξινόμηση (Unordered - μέχρι Python 3.6)**
   - Από Python 3.7+, τα λεξικά διατηρούν τη σειρά εισαγωγής (insertion order)
   - Προηγούμενες εκδόσεις δεν εγγυώνται συγκεκριμένη σειρά
3. **Μοναδικότητα Κλειδιών (Key Uniqueness)**
   - Κάθε κλειδί εμφανίζεται μόνο μία φορά

- Επαναπροσδιορισμός τιμής με υπάρχον κλειδί αντικαθιστά την προηγούμενη

4. **Δυναμικό Μέγεθος (Dynamic Size)**
    - Το μέγεθος του λεξικού μπορεί να αυξομειωθεί κατά την εκτέλεση
5. **Αποδοτική Πρόσβαση (Efficient Access)**
    - Η πρόσβαση μέσω κλειδιού έχει χρονική πολυπλοκότητα O(1) κατά μέσο όρο
    - Χρησιμοποιεί hash tables εσωτερικά

4.3.2. Σύγκριση με Άλλες Δομές Δεδομένων

Ενώ στις λίστες και στις πλειάδες η πρόσβαση στα δεδομένα εξαρτάται από τη σειριακή τους θέση (index), στα λεξικά η ανάκτηση βασίζεται σε σημασιολογικά κλειδιά. Αυτό καθιστά τα λεξικά ιδανικά για την αναπαράσταση οντοτήτων με συγκεκριμένες ιδιότητες (π.χ. προφίλ χρήστη, ρυθμίσεις συστήματος), σε αντίθεση με τις λίστες και πλειάδες που προτιμώνται για διατεταγμένες ακολουθίες.

Πίνακας 9: Σύγκριση Λεξικού με άλλες Δομές Δεδομένων

Χαρακτηριστικό	Λίστες	Πλειάδες	Σύνολα	Λεξικά
Συντακτικό	Αγκύλες []	Παρενθέσεις ()	Άγκιστρα {} ή set()	Άγκιστρα {} με :
Παράδειγμα	[1, 2, 3]	(1, 2, 3)	{1, 2, 3}	{"α": 1, "β": 2}
Κενή δομή	[]	()	set()	{}
Διάταξη	Διατεταγμένες	Διατεταγμένες	Μη διατεταγμένα	Σειρά εισαγωγής
Μοναδικότητα	Επιτρέπονται διπλότυπα	Επιτρέπονται διπλότυπα	Μόνο μοναδικά	Κλειδιά: μοναδικά Τιμές: διπλότυπα
Μεταλαξιμότητα	Μεταβλητές (Mutable)	Αμετάβλητες (Immutable)	Μεταβλητά (Mutable)	Μεταβλητά (Mutable)
Τύπος	Ακολουθία (Sequence)	Ακολουθία (Sequence)	Συλλογή (Collection)	Απεικόνιση (Mapping)

4.3.3. Δημιουργία Λεξικού

Η δημιουργία λεξικών στην Python μπορεί να πραγματοποιηθεί με πολλούς διαφορετικούς τρόπους, ο καθένας από τους οποίους είναι κατάλληλος για διαφορετικά σενάρια. Η Python προσφέρει αυτή την ποικιλία μεθόδων για να καλύψει διάφορες ανάγκες προγραμματισμού, από απλή δημιουργία μέχρι σύνθετους μετασχηματισμούς δεδομένων.

4.3.3.1 Χρήση αγκίστρων { }

Η πιο άμεση και συνήθης μέθοδος δημιουργίας λεξικού είναι η χρήση αγκίστρων { } με ρητή καταγραφή των ζευγών κλειδιού-τιμής. Αυτή η σύνταξη ονομάζεται "literal syntax" και είναι η πιο διαισθητική, καθώς αντανακλά άμεσα τη δομή του λεξικού.

Η σύνταξη είναι:

```
λεξικό = {
    κλειδί1: τιμή1,
    κλειδί2: τιμή2,
    ...,
    κλειδίN: τιμήN
}
```

Παράδειγμα

```
# Αρχικοποίηση λεξικού ρυθμίσεων διακομιστή
server_config = {
    "host": "127.0.0.1",
    "port": 8080,
    "debug": True
}

# Κενό λεξικό
κενό_λεξικό = {}
```

Κ 125: Δημιουργία Λεξικού με άγκυστρα { }

Η χρήση αυτής της μεθόδου είναι ιδανική όταν:

- Γνωρίζουμε εκ των προτέρων τα κλειδιά και τις τιμές

- Θέλουμε να δημιουργήσουμε ένα λεξικό με σταθερή δομή

- Η αναγνωσιμότητα του κώδικα είναι προτεραιότητα

4.3.3.2 Constructor dict()

Η συνάρτηση dict() είναι ο επίσημος κατασκευαστής (constructor) της κλάσης dictionary και προσφέρει πολλούς τρόπους δημιουργίας λεξικών. Αυτή η μέθοδος είναι ιδιαίτερα χρήσιμη όταν τα δεδομένα μας προέρχονται από άλλες δομές ή όταν θέλουμε να μετατρέψουμε υπάρχοντα δεδομένα σε λεξικό.

Η σύνταξη είναι:

```
dict(key1=value1, key2=value2)
```

Παραδείγματα

```
person = dict(
    name="Νίκος",
    age=25,
    profession="Μηχανικός Λογισμικού",
    city="Θεσσαλονίκη"
)
print(person)
```

```
# 1. Κενό Λεξικό
empty_registry = dict()

# 2. Χρήση Keyword Arguments
user = dict(username="yannis", access_level=5, active=True)

# 3. Χρήση Keyword Arguments
person = dict(
    name="Νίκος",
    age=25,
    profession="Μηχανικός Λογισμικού",
    city="Θεσσαλονίκη"
)
```

Κ 126: Δημιουργία λεξικού με χρήση constructor dict()

Ένα λεξικό μπορεί να δημιουργηθεί από μία λίστα που περιλαμβάνει πλειάδες:

```
# 2. Χρήση Λίστας από Πλειάδες (Mapping)
pairs = [("id", 101), ("status", "online")]
session = dict(pairs)
```

Κ 127: Δημιουργία λεξικού από λίστα με πλειάδες

4.3.4. Πρόσβαση σε στοιχεία Λεξικού και Διαχείριση Εξαιρέσεων

Σε αντίθεση με τις λίστες όπου η πρόσβαση γίνεται μέσω αριθμητικών δεικτών, στα λεξικά η πρόσβαση πραγματοποιείται μέσω των κλειδιών (keys). Αυτή η διαφορά δεν είναι απλώς συντακτική - αντανακλά μια διαφορετική φιλοσοφία οργάνωσης δεδομένων που κάνει τα λεξικά ιδανικά για αναπαράσταση δομημένων πληροφοριών. Για την ανάκτηση τιμών, χρησιμοποιούνται είτε οι αγκύλες [] είτε η μέθοδος .get().

4.3.4.1 Πρόσβαση με Αγκύλες []

Ο πιο άμεσος τρόπος πρόσβασης σε μια τιμή λεξικού είναι η χρήση του τελεστή αγκυλών [ ] με το αντίστοιχο κλειδί. Αυτή η μέθοδος είναι ανάλογη με την πρόσβαση σε στοιχεία λίστας, αλλά αντί για αριθμητικό δείκτη χρησιμοποιούμε ένα περιγραφικό κλειδί: `dictionary[key]`

Παράδειγμα

```
student = {
    "όνομα": "Ελένη Γεωργίου",
    "AM": "2021030045",
    "τμήμα": "Πληροφορική και Τηλεπικοινωνίες",
    "εξάμηνο": 4,
    "βαθμός": 8.2,
    "email": "egeorgiou@cs.example.gr",
    "ενεργός": True
}

# Πρόσβαση σε μεμονωμένα στοιχεία
student_name = student["όνομα"]
print(f"Όνομα: {student_name}")  # Όνομα: Ελένη Γεωργίου

# Πρόσβαση σε διάφορους τύπους τιμών
am = student["AM"]        # String
semester = student["εξάμηνο"] # Integer
```

Κ 128: Πρόσβαση σε τιμή Λεξικού με την χρήση αγκύλων [ ]: dictionary[key]

Σημείωση: Η μέθοδος έχει έναν σημαντικό περιορισμό: αν το κλειδί δεν υπάρχει στο λεξικό, προκαλείται σφάλμα **KeyError**. Αυτό μπορεί να είναι προβληματικό σε περιπτώσεις όπου δεν είμαστε σίγουροι για την ύπαρξη ενός κλειδιού.

Παράδειγμα διαχείρισής του

```
# Προσπάθεια πρόσβασης σε ανύπαρκτο κλειδί
try:
    phone = student["τηλέφωνο"]  # Το κλειδί δεν υπάρχει
except KeyError as e:
```

```
    print(f"Σφάλμα: Το κλειδί {e} δεν βρέθηκε στο λεξικό")
    print("Αυτό θα σταματούσε την εκτέλεση του προγράμματος χωρίς try-except!")

# Για να αποφύγουμε το σφάλμα, μπορούμε να ελέγξουμε την ύπαρξη πρώτα
if "τηλέφωνο" in student:
    phone = student["τηλέφωνο"]
else:
    phone = "Δεν έχει καταχωρηθεί"
    print(f"Τηλέφωνο: {phone}")
```
Κ 129: Πρόσβαση σε ανύπαρκτο κλειδί του Λεξικού

4.3.4.2 Πρόσβαση με τη Μέθοδο get()

Η μέθοδος get() προσφέρει έναν ασφαλέστερο και πιο ευέλικτο τρόπο πρόσβασης στα στοιχεία ενός λεξικού. Σε αντίθεση με τον τελεστή [], η get() δεν προκαλεί σφάλμα όταν το κλειδί δεν υπάρχει. Αντίθετα, επιστρέφει None (ή μια προεπιλεγμένη τιμή που μπορούμε να ορίσουμε).

Η σύνταξη της get είναι η παρακάτω. Αν το κλειδί υπάρχει, επιστρέφεται η αντίστοιχη τιμή. Αν δεν υπάρχει, επιστρέφεται η προεπιλεγμένη τιμή (ή `None` αν δεν έχει οριστεί προεπιλογή).

```
τιμή = λεξικό.get(κλειδί, προεπιλεγμένη_τιμή)
```
Παράδειγμα

```
# Χρήση get() χωρίς προεπιλεγμένη τιμή
email = student.get("email")
print(f"Email: {email}")  # Email: egeorgiou@cs.example.gr

phone = student.get("τηλέφωνο")
print(f"Τηλέφωνο: {phone}")  # Τηλέφωνο: None

# Χρήση get() με προεπιλεγμένη τιμή
phone_with_default_value = student.get("τηλέφωνο", "Δεν έχει καταχωρηθεί")
print(f"Τηλέφωνο: {phone_with_default_value}") # Τηλέφωνο: Δεν έχει καταχωρηθεί
```
Κ 130: Η χρήση της μεθόδου get() του Λεξικού

4.3.4.3 Τροποποίηση Στοιχείων Λεξικών

Η μεταβλητότητα των λεξικών αποτελεί ένα από τα πιο σημαντικά χαρακτηριστικά τους, επιτρέποντάς μας να προσαρμόζουμε δυναμικά το περιεχόμενό τους ανάλογα με τις ανάγκες του προγράμματός μας. Σε αντίθεση με τις αμετάβλητες δομές όπως οι πλειάδες, τα λεξικά μας παρέχουν την ελευθερία να τροποποιούμε τιμές, να προσθέτουμε νέα ζεύγη κλειδιού-τιμής, και να αφαιρούμε υπάρχοντα στοιχεία. Αυτή η ευελιξία καθιστά τα λεξικά ιδανικά για την αναπαράσταση δεδομένων που εξελίσσονται με την πάροδο του χρόνου.

Η αλλαγή της τιμής που αντιστοιχεί σε ένα υπάρχον κλειδί είναι μια από τις πιο συνήθεις λειτουργίες στον χειρισμό λεξικών. Η διαδικασία είναι εξαιρετικά απλή: χρησιμοποιούμε τον τελεστή ανάθεσης = με το κλειδί που θέλουμε να ενημερώσουμε,

```
dictionary[key] = new_value
```

Παράδειγμα

```
product = {
    "όνομα": "Laptop Dell XPS 13",
    "τιμή": 1299.99,
    "απόθεμα": 10,
    "κατηγορία": "Ηλεκτρονικά",
    "διαθέσιμο": True
}

print("Αρχικό προϊόν:")
print(product)

# Απλή τροποποίηση τιμής
product["τιμή"] = 1199.99  # Μείωση τιμής
print(f"\nΝέα τιμή: {product['τιμή']}€")

# Ενημέρωση αποθέματος μετά από πώληση
product["απόθεμα"] -= 1
print(f"Απομένουν {product['απόθεμα']} τεμάχια")

# Ενημέρωση boolean τιμής
if product["απόθεμα"] == 0:
    product ["διαθέσιμο"] = False
    print("Το προϊόν εξαντλήθηκε")
```

Κ 131: Τροποποίηση Υπαρχουσών Τιμών του Λεξικού (dictionary[key] = new_value)

4.3.5. Μέθοδοι Λεξικού

4.3.5.1 Η μέθοδος keys()

Επιστρέφει μια προβολή (view) όλων των κλειδιών του λεξικού. Αυτή η προβολή συμπεριφέρεται παρόμοια με σύνολο και ενημερώνεται δυναμικά αν το λεξικό τροποποιηθεί.

Η σύνταξη είναι:

```
dictionary.keys()
```

Παράδειγμα

```
product = {
    "κωδικός": "P001",
    "όνομα": "Smartphone",
    "τιμή": 399.99,
    "κατασκευαστής": "Samsung",
    "μοντέλο": "Galaxy S21"
}

# Πρόσβαση στα κλειδιά
keys = product.keys()
print(keys)  # dict_keys(['κωδικός', 'όνομα', 'τιμή', ...])
```

K 132: Η χρήση της μεθόδου keys() του Λεξικού

4.3.5.2 Η μέθοδος values()

Επιστρέφει μια προβολή όλων των τιμών. Σε αντίθεση με τα κλειδιά, οι τιμές μπορεί να επαναλαμβάνονται.

Η σύνταξη είναι:

```
dictionary.values()
```

Παράδειγμα

```
grades = {
    "Μαθηματικά": 85,
    "Φυσική": 90,
    "Χημεία": 85,  # Ίδιος βαθμός με Μαθηματικά
    "Πληροφορική": 92,
    "Αγγλικά": 88
}

# Πρόσβαση στις τιμές
values = grades.values()
print(values)  # dict_values([85, 90, 85, 92, 88])
```

K 133: Η χρήση της μεθόδου values() του Λεξικού

4.3.5.3 Η μέθοδος items()

Επιστρέφει μια προβολή ζευγών (κλειδί, τιμή) ως πλειάδες. Αυτή είναι η πιο χρήσιμη μέθοδος για επανάληψη όταν χρειαζόμαστε και τα κλειδιά και τις τιμές.

Η σύνταξη είναι:

```
dictionary.items()
```

Παράδειγμα

```
student = {
    "όνομα": "Κώστας Παπαδόπουλος",
    "AM": "2020030123",
    "εξάμηνο": 5,
    "μέσος_όρος": 7.8
}

# Πρόσβαση σε ζεύγη
key_values = student.items()
print(key_values)
# dict_items([('όνομα', 'Κώστας Παπαδόπουλος'), ('AM', '2020030123'), ...])
```

Κ 134: Η χρήση της μεθόδου items() του Λεξικού

4.3.5.4 Προσθήκη Νέων Ζευγών Κλειδιού-Τιμής

Ένα από τα πιο ισχυρά χαρακτηριστικά των λεξικών είναι η δυνατότητα δυναμικής προσθήκης νέων στοιχείων. Αυτό επιτρέπει στα προγράμματά μας να εξελίσσονται και να προσαρμόζονται σε νέες απαιτήσεις χωρίς να χρειάζεται να επαναπροσδιορίσουμε ολόκληρη τη δομή.

```
# Ξεκινάμε με βασικό προφίλ χρήστη
user = {
    "username": "maria_p",
    "email": "maria@example.com"
}

print("Αρχικό προφίλ:")
print(user)

# Προσθήκη νέων πληροφοριών σταδιακά
user["ονοματεπώνυμο"] = "Μαρία Παπαδοπούλου"
user["ημερομηνία_εγγραφής"] = "2024-01-15"
user["premium"] = False
user["προτιμήσεις"] = {
    "γλώσσα": "ελληνικά",
    "ειδοποιήσεις": True,
    "θέμα": "σκούρο"
}

print("\nΠλήρες προφίλ:")
for κλειδί, τιμή in user.items():
    print(f"  {κλειδί}: {τιμή}")
```

Κ 135: Προσθήκη Νέων Ζευγών Κλειδιού-Τιμής στο Λεξικό

Σημείωση: Μια σημαντική παρατήρηση είναι ότι αν το κλειδί που χρησιμοποιούμε ήδη υπάρχει, η νέα τιμή θα αντικαταστήσει την παλιά.

Παράδειγμα

```
# Παράδειγμα αντικατάστασης
configuration = {
    "volume": 50,
    "brightness": 70
}

print(f"Αρχική ένταση: {configuration['volume']}")
# Προσθήκη (ή αντικατάσταση αν υπάρχει)
configuration["volume"] = 80
print(f"Νέα ένταση: {configuration['volume']}")
```

Κ 136: Αντικατάσταση της παλιάς τιμής ενός κλειδιού σε νέα

4.3.5.5 Ενημέρωση με τη Μέθοδο update()

Η μέθοδος update() παρέχει έναν αποδοτικό τρόπο για να ενημερώσουμε πολλά ζεύγη κλειδιού-τιμής ταυτόχρονα. Αυτή η μέθοδος δέχεται ένα άλλο λεξικό (ή άλλες μορφές δεδομένων) και ενσωματώνει τα στοιχεία του στο υπάρχον λεξικό. Η σύνταξη είναι:

```
dictionary.update(second_dictionary)
```

Παράδειγμα

```
# Βασικό προφίλ φοιτητή
student = {
    "όνομα": "Κώστας",
    "επώνυμο": "Γεωργίου",
    "AM": "2021030089"
}

# Προσθήκη ακαδημαϊκών στοιχείων
academic_items = {
    "τμήμα": "Πληροφορική",
    "εξάμηνο": 5,
    "μέσος_όρος": 7.5,
    "ECTS": 120
}

student.update(academic_items)
print("Μετά την προσθήκη ακαδημαϊκών στοιχείων:")
print(student)
```

Κ 137: Ενημέρωση Λεξικού με τη μέθοδο update()

Όταν το κλειδί υπάρχει στο λεξικό ενημερώνεται η τιμή του κλειδιού με την νέα τιμή που περιλαμβάνεται στην update.

Παράδειγμα

```
student = {
    "όνομα": "Κώστας",
    "επώνυμο": "Γεωργίου",
    "AM": "2021030089",
    "τμήμα": "Πληροφορική",
    "εξάμηνο": 5,
    "μέσος_όρος": 7.5,
    "ECTS": 120
}
student.update({"εξάμηνο": 6, "μέσος_όρος": 7.8})
print(f"\nΜετά την προαγωγή: Εξάμηνο {student['εξάμηνο']}, ΜΟ {student['μέσος_όρος']}")
# Μετά την προαγωγή: Εξάμηνο 6, ΜΟ 7.8
```

Κ 138: Αντικατάσταση τιμής κλειδιού που υπάρχει στο λεξικό με την μέθοδο update()

4.3.5.6 Διαγραφή Στοιχείων

Η Python προσφέρει πολλούς τρόπους διαγραφής στοιχείων από λεξικά, ο καθένας με τα δικά του χαρακτηριστικά και περιπτώσεις χρήσης.

4.3.5.6.1 Διαγραφή με del

Η δήλωση del αφαιρεί ένα ζεύγος κλειδιού-τιμής από το λεξικό. Αν το κλειδί δεν υπάρχει, προκαλείται KeyError. Η σύνταξη είναι: `del dictionary[key]`

Παράδειγμα

```
product = {
    "κωδικός": "A123",
    "όνομα": "Tablet",
    "τιμή": 299.99,
    "απόθεμα": 0,
    "κατηγορία": "Ηλεκτρονικά",
    "παλιό_μοντέλο": True
}

print("Αρχικό προϊόν:")
print(product)

# Διαγραφή πεδίου που δεν χρειάζεται πια
```

```python
del product["παλιό_μοντέλο"]
print("\nΜετά τη διαγραφή 'παλιό_μοντέλο':")
print(product)

# Διαγραφή με έλεγχο ύπαρξης
if "απόθεμα" in product:
    del product["απόθεμα"]
    print("Διαγράφηκε το απόθεμα")

# Προσπάθεια διαγραφής ανύπαρκτου κλειδιού - ΣΦΑΛΜΑ
try:
    del product["ανύπαρκτο"]
except KeyError:
    print("Το κλειδί 'ανύπαρκτο' δεν βρέθηκε")
```

Κ 139: Διαγραφή στοιχείων Λεξικού με τη del

4.3.5.6.2 Διαγραφή με pop()

Η μέθοδος pop() αφαιρεί ένα στοιχείο και επιστρέφει την τιμή του. Αυτό είναι χρήσιμο όταν επιθυμείτε να χρησιμοποιηθεί η τιμή πριν διαγραφεί. Μπορεί επίσης να οριστεί μια προεπιλεγμένη τιμή για την περίπτωση που το κλειδί δεν υπάρχει. Η σύνταξη είναι :

```
dictionary.pop(key)
```

Παράδειγμα

```python
configuration = {
    "theme": "dark",
    "language": "el",
    "notifications": True,
    "sound": True,
    "auto_save": False
}

# Pop επιστρέφει την τιμή
old_theme = configuration.pop("theme")
print(f"Αφαιρέθηκε θέμα: {old_theme}")
print(f"Ρυθμίσεις: {configuration}")

# Pop με προεπιλεγμένη τιμή (ασφαλής)
privacy_mode = configuration.pop("privacy_mode", False)
print(f"Privacy mode: {privacy_mode}")  # False (προεπιλογή)
```

Κ 140: Διαγραφή στοιχείων Λεξικού με την μέθοδο pop

4.3.5.6.3 Διαγραφή με popitem()

Η μέθοδος popitem() αφαιρεί και επιστρέφει το τελευταίο ζεύγος (key, value) που προστέθηκε ως πλειάδα. Σε παλαιότερες εκδόσεις Python (πριν την 3.7) αφαιρούσε ένα τυχαίο ζεύγος. Η σύνταξη είναι:

```
dictionary.popitem()
```

Παράδειγμα

```
tasks = {
    "task_1": "Συγγραφή αναφοράς",
    "task_2": "Έλεγχος email",
    "task_3": "Ενημέρωση βάσης δεδομένων",
    "task_4": "Code review"
}

print("Εργασίες προς εκτέλεση:")
print(tasks)

# Επεξεργασία εργασιών με σειρά LIFO (Last In, First Out)
while tasks:
    task_id, task_name = tasks.popitem()
    print(f"\nΕκτελείται: {task_id} - {task_name}")
    # Κώδικας εκτέλεσης εργασίας...

print("Όλες οι εργασίες ολοκληρώθηκαν!")
```

Κ 141: Διαγραφή στοιχείων Λεξικού με την μέθοδο popitem()

4.3.5.6.4 Καθαρισμός όλων των στοιχείων με clear()

Η μέθοδος clear() αδειάζει εντελώς το λεξικό, αφαιρώντας όλα τα ζεύγη κλειδιού-τιμής. Η σύνταξη της είναι:

```
dictionary.clear()
```

Παράδειγμα

```
cache = {
    "user_123": {"name": "Μαρία", "last_seen": "10:30"},
    "user_456": {"name": "Νίκος", "last_seen": "11:45"},
    "user_789": {"name": "Άννα", "last_seen": "09:15"}
}

print(f"1. Cache με {len(cache)} χρήστες")
print(cache)
```

```
# Καθαρισμός cache
cache.clear()
print(f"2. Cache με {len(cache)} χρήστες")
print(cache)
```
Κ 142: Καθαρισμός όλων των στοιχείων του Λεξικού με την μέθοδο clear()

4.3.5.7 Αντιγραφή Λεξικών

Η αντιγραφή λεξικών απαιτεί προσοχή, ειδικά όταν τα λεξικά περιέχουν εμφωλευμένες δομές. Υπάρχουν διαφορετικοί τύποι αντιγραφής, ο καθένας κατάλληλος για διαφορετικές περιπτώσεις.

4.3.5.7.1 Ρηχή Αντιγραφή με την μέθοδο copy (Shallow Copy)

Η ρηχή αντιγραφή δημιουργεί ένα νέο λεξικό, αλλά τα εμφωλευμένα αντικείμενα (όπως λίστες ή λεξικά) παραμένουν αναφορές. Η σύνταξη της copy είναι:

```
new_dictionary = dictionary.copy()
```

Παράδειγμα

```
# Ρηχή αντιγραφή με copy()
prototype_dict = {"α": 1, "β": 2, "γ": 3}
copy_dict = prototype_dict.copy()

# Τώρα είναι ξεχωριστά λεξικά
copy_dict["α"] = 999
print(prototype_dict)  # {'α': 1, 'β': 2, 'γ': 3} - Αμετάβλητο!
print(copy_dict)  # {'α': 999, 'β': 2, 'γ': 3}

# ΑΛΛΑ προσοχή με εμφωλευμένα αντικείμενα!
nested_prototype_dict = {
    "όνομα": "Μαρία",
    "βαθμοί": [85, 90, 88],  # Λίστα (mutable)
    "πληροφορίες": {"ηλικία": 21}  # Εμφωλευμένο λεξικό
}
s_copy_dict = nested_prototype_dict.copy()
# Αλλαγή στην εμφωλευμένη λίστα
s_copy_dict["βαθμοί"].append(92)

# ΠΡΟΒΛΗΜΑ: Επηρεάζεται και το πρωτότυπο!
print(nested_prototype_dict["βαθμοί"])  # [85, 90, 88, 92]
print("Η εμφωλευμένη λίστα επηρεάστηκε!")
```
Κ 143: Αντιγραφή Λεξικού με την μέθοδο copy()

4.3.5.7.2 Βαθιά Αντιγραφή (Deep Copy)

Η βαθιά αντιγραφή αντιγράφει το λεξικό σε ένα νέο λεξικό. Στην περίπτωση που το λεξικό περιλαμβάνει εμφωλευμένες δομές αντιγράφονται και αυτές εξ ολοκλήρου. Η σύνταξη είναι:

```
import copy

new_dictionary = copy.deepcopy()
```

Σημείωση: Για να χρησιμοποιηθεί η deepcopy απαιτείται η εισαγωγή του module copy (import copy)

Παράδειγμα

```python
import copy
student_data = {
    "φοιτητής": {
        "όνομα": "Κώστας",
        "βαθμοί": {
            "Μαθηματικά": [85, 90],
            "Φυσική": [88, 92]
        }
    },
    "μαθήματα": ["Μαθηματικά", "Φυσική", "Χημεία"]
}

# Βαθιά αντιγραφή
full_copy = copy.deepcopy(student_data)

# Τροποποίηση εμφωλευμένης δομής
full_copy["φοιτητής"]["βαθμοί"]["Μαθηματικά"].append(95)
full_copy["μαθήματα"].append("Πληροφορική")

# Έλεγχος - Το πρωτότυπο παραμένει αμετάβλητο
print("Πρωτότυπο:")
print(student_data["φοιτητής"]["βαθμοί"]["Μαθηματικά"])  # [85, 90]
print(student_data["μαθήματα"])  # ['Μαθηματικά', 'Φυσική', 'Χημεία']

print("\nΑντίγραφο:")
print(full_copy["φοιτητής"]["βαθμοί"]["Μαθηματικά"])  # [85, 90, 95]
print(full_copy["μαθήματα"])  # ['Μαθηματικά', 'Φυσική', 'Χημεία', 'Πληροφορική']
```

Κ 144: Αντιγραφή Λεξικού με την μέθοδο deepcopy

4.3.6. Διάσχιση στοιχείων Λεξικού

Η διάσχιση ή η επανάληψη (iteration ή looping) με τη χρήση βρόχων σε λεξικά αποτελεί μια θεμελιώδη λειτουργία που επιτρέπει την πρόσβαση και επεξεργασία όλων των στοιχείων ενός

λεξικού. Σε αντίθεση με τις λίστες όπου η επανάληψη γίνεται με μία μόνο διάσταση (τα στοιχεία), τα λεξικά προσφέρουν τρεις διαφορετικές διαστάσεις επανάληψης: στα **κλειδιά** (keys), στις **τιμές** (values), ή στα **ζεύγη κλειδιού-τιμής** (items). Αυτή η ευελιξία καθιστά τα λεξικά εξαιρετικά ισχυρά εργαλεία για την επεξεργασία δομημένων δεδομένων.

Η Python όπως έχει ήδη αναφερθεί παρέχει τρεις ειδικές μεθόδους για την επανάληψη σε λεξικά:

- **keys()**: Επιστρέφει μια προβολή (view) όλων των κλειδιών
- **values()**: Επιστρέφει μια προβολή όλων των τιμών
- **items()**: Επιστρέφει μια προβολή ζευγών (κλειδί, τιμή) ως tuples

4.3.6.1 Επανάληψη στα κλειδιά

Κατά την εφαρμογή της επανάληψης σε ένα λεξικό η Python επιστρέφει αυτόματα τα κλειδιά του. Αυτή η προεπιλεγμένη συμπεριφορά είναι εξαιρετικά συχνή και χρήσιμη.

```
for κλειδί in φοιτητής:
   print(κλειδί)
```

Παράδειγμα βρόχου for

```
# Παράδειγμα λεξικού με στοιχεία φοιτητή
student = {
    "όνομα": "Μαρία Παπαδοπούλου",
    "AM": "2021030045",
    "τμήμα": "Πληροφορική",
    "εξάμηνο": 5,
    "μέσος_όρος": 8.2,
    "ενεργός": True
}

# Προεπιλεγμένη επανάληψη - επαναλαμβάνει στα ΚΛΕΙΔΙΑ
print("=== Προεπιλεγμένη Επανάληψη ===")
for item in student:
    print(item)

# Έξοδος:
# όνομα
# AM
# τμήμα
# εξάμηνο
# μέσος_όρος
# ενεργός
```

Κ 145: Επανάληψη στα κλειδιά με τη χρήση βρόχου for

Εναλλακτικά για την επιστροφή των κλειδιών ενός λεξικού μπορεί να χρησιμοποιηθεί η μέθοδος του λεξικού keys():

```
for x in φοιτητής.keys():
   print(x)
```

```
# Παράδειγμα λεξικού με στοιχεία φοιτητή
student = {
    "όνομα": "Μαρία Παπαδοπούλου",
    "ΑΜ": "2021030045",
    "τμήμα": "Πληροφορική",
    "εξάμηνο": 5,
    "μέσος_όρος": 8.2,
    "ενεργός": True
}

# Προεπιλεγμένη επανάληψη - επαναλαμβάνει στα ΚΛΕΙΔΙΑ
print("=== Προεπιλεγμένη Επανάληψη ===")
for x in student.keys():
    print(x)

# Έξοδος:
# όνομα
# ΑΜ
# τμήμα
# εξάμηνο
# μέσος_όρος
# ενεργός
```

Κ 146: Επανάληψη στα κλειδιά με τη χρήση βρόχου for και τη μέθοδο keys()

4.3.6.2 Πρόσβαση στις τιμές λεξικού με την χρήση κλειδιών

Για την προσπέλαση στις τιμές του λεξικού χρησιμοποιείται η προεπιλεγμένη επανάληψη που πραγματοποιείται πάνω στα κλειδιά:

```
for κλειδί in φοιτητής:
    print(φοιτητής[κλειδί])
```

Παράδειγμα

```
# Παράδειγμα λεξικού με στοιχεία φοιτητή
student = {
    "όνομα": "Μαρία Παπαδοπούλου",
    "ΑΜ": "2021030045",
    "τμήμα": "Πληροφορική",
    "εξάμηνο": 5,
```

```
    "μέσος_όρος": 8.2,
    "ενεργός": True
}
for item in student:
    print(student[item])
```

Κ 147: Πρόσβαση στις τιμές λεξικού με την χρήση κλειδιών

4.3.6.3 Πρόσβαση στις τιμές λεξικού με την χρήση της μεθόδου values()

Ένας άλλος τρόπος προσπέλασης των τιμών ενός λεξικού είναι με την χρήση της μεθόδου values() του λεξικού:

```
for x in φοιτητής.values():
    print(x)
```

Παράδειγμα

```
# Παράδειγμα λεξικού με στοιχεία φοιτητή
student = {
    "όνομα": "Μαρία Παπαδοπούλου",
    "AM": "2021030045",
    "τμήμα": "Πληροφορική",
    "εξάμηνο": 5,
    "μέσος_όρος": 8.2,
    "ενεργός": True
}
for item in student.values():
    print(item)
```

Κ 148: Πρόσβαση στις τιμές λεξικού με την μέθοδο values()

4.3.6.4 Πρόσβαση στα κλειδιά και στις τιμές λεξικού με την μέθοδο items()

Η μέθοδος items() είναι η πιο χρήσιμη και συχνά χρησιμοποιούμενη μέθοδος για επανάληψη σε λεξικά. Επιστρέφει ένα dictionary view object που περιέχει πλειάδες (tuples) της μορφής `(κλειδί, τιμή)`. Αυτό επιτρέπει την ταυτόχρονη πρόσβαση τόσο στο κλειδί όσο και στην τιμή κατά τη διάρκεια της επανάληψης, χωρίς επιπλέον lookup operations.

```
for κλειδί, τιμή in προϊόν.items():
    print(f"{κλειδί}: {τιμή}")
```

Παράδειγμα

```
product = {
    "κωδικός": "P001",
```

```
    "όνομα": "Smartphone Samsung Galaxy",
    "τιμή": 399.99,
    "απόθεμα": 15,
    "κατηγορία": "Ηλεκτρονικά"
}

print("\n=== Επανάληψη με items() ===")
for key, value in product.items():
    print(f"{key}: {value}")
```

Κ 149: Πρόσβαση στα κλειδιά και στις τιμές λεξικού με την μέθοδο items()

4.3.7. Ασκήσεις Λεξικών

Άσκηση 4 - Δημιουργία Λεξικού

Δημιούργησε ένα λεξικό που αντιπροσωπεύει ένα καλάθι αγορών. Κάθε κλειδί είναι το όνομα ενός προϊόντος και κάθε τιμή είναι η ποσότητά του.

Φτιάξε το λεξικό με 2 διαφορετικούς τρόπους και εκτύπωσέ το κάθε φορά για να επιβεβαιώσεις ότι είναι ίδιο:

- Με άγκιστρα {}
- Με τον constructor dict()

```
products = ["μήλα", "ψωμί", "γάλα", "τυρί"]
quantities = [3, 1, 2, 1]

# Αναμενόμενο αποτέλεσμα και με τους 2 τρόπους:
# {'μήλα': 3, 'ψωμί': 1, 'γάλα': 2, 'τυρί': 1}
```

Άσκηση 5 — Διάσχιση Λεξικού

Δίνεται το παρακάτω λεξικό με θερμοκρασίες εβδομάδας:

```
temperatures = {
    "Δευτέρα": 18, "Τρίτη": 22, "Τετάρτη": 17,
    "Πέμπτη": 25, "Παρασκευή": 23, "Σάββατο": 20, "Κυριακή": 16
}
```

Χρησιμοποιώντας βρόχο, εκτύπωσε:

1. Μόνο τις **μέρες** (κλειδιά)
2. Μόνο τις **θερμοκρασίες** (τιμές)
3. **Μέρα και θερμοκρασία** μαζί σε μορφή: Δευτέρα → 18°C

4. Μόνο τις μέρες που η θερμοκρασία είναι **πάνω από 20°C**

Άσκηση 6 — Χρήση Μεθόδων Λεξικού

Δίνεται ένα λεξικό με βαθμούς μαθητή:

```
grades = {"Μαθηματικά": 15, "Φυσική": 12, "Ιστορία": 18}
```

Υλοποιήστε τα παρακάτω **χρησιμοποιώντας μεθόδους** :

1. Προσθέστε το μάθημα "Χημεία" με βαθμό 14 — χρησιμοποιώντας την update()
2. Διαβάστε τον βαθμό της "Γεωγραφίας" χωρίς σφάλμα — χρησιμοποιώντας την get() με προεπιλεγμένη τιμή "Δεν βρέθηκε"
3. Αφαιρέστε τη "Φυσική" και **αποθηκεύστε την τιμή** της — χρησιμοποιώντας την pop()
4. Εκτυπώστε το τελικό λεξικό και επιβεβαιώστε ότι έχει 3 μαθήματα

4.4. Μεταλλαξιμότητα – Μεταβλητότητα (Mutability) και Μη Μεταλλαξιμότητα - Αμεταβλητότητα (Immutability)

Στην Python, κάθε αντικείμενο κατατάσσεται σε μία από δύο κατηγορίες βάσει της ικανότητάς του να αλλάζει την κατάστασή του μετά τη δημιουργία του:

- **Μεταβλητότητα - Μεταλλαξιμότητα (Mutability):** Τα αντικείμενα αυτά επιτρέπουν την τροποποίηση του περιεχομένου τους χωρίς να αλλάζει η ταυτότητά τους στη μνήμη (memory address). Παραδείγματα: lists, dictionaries, sets.
- **Αμεταβλητότητα - Μη Μεταλλαξιμότητα (Immutability):** Μόλις αρχικοποιηθούν, η κατάστασή τους παραμένει σταθερή. Οποιαδήποτε «αλλαγή» στην πραγματικότητα δημιουργεί ένα νέο αντικείμενο στη μνήμη. Παραδείγματα: int, float, string, tuple.

4.4.1. Εφαρμογή Αμεταβλητότητας - Μη Μεταλλαξιμότητας (Immutability)

Οι προγραμματιστές επιλέγουν **immutable** δομές (όπως πλειάδες) όταν απαιτείται ασφάλεια δεδομένων (π.χ. κλειδιά σε λεξικά ή σταθερές ρυθμίσεις), καθώς διασφαλίζουν ότι τα δεδομένα δεν θα τροποποιηθούν κατά λάθος από κάποιο άλλο τμήμα του κώδικα (side effects).

4.5. Περιφραστικές λίστες και λεξικά (comprehensions)

Οι **περιφραστικές συλλογές (comprehensions)** αποτελούν έναν από τους πιο ισχυρούς και κομψούς τρόπους (Pythonic way) για τη δημιουργία νέων δομών δεδομένων. Στην ουσία, επιτρέπουν τη συμπύκνωση ενός ολόκληρου βρόχου for και των συνοδευτικών συνθηκών if σε μία μόνο γραμμή κώδικα.

4.5.1. Περιφραστικές Λίστες (List Comprehensions)

Η περιφραστική λίστα προσφέρει μια σύντομη σύνταξη για τη δημιουργία μιας νέας λίστας με βάση τις τιμές μιας υπάρχουσας λίστας (ή οποιασδήποτε άλλης ακολουθίας) με χρήση βρόχου for και προαιρετικά συνθήκης.

Γενική Σύνταξη:

```
new_list = [expression for item in iterable if condition]
```

- **Expression:** Η πράξη ή η τιμή που θα αποθηκευτεί στη νέα λίστα.
- **For Item in Iterable:** Η κλασική επανάληψη πάνω στα δεδομένα.
- **If Condition (Προαιρετικό):** Ένα φίλτρο που καθορίζει ποια στοιχεία θα συμπεριληφθούν.

Α. Απλός Μετασχηματισμός (Χωρίς Συνθήκη)

Έστω ότι έχουμε μια λίστα με ακέραιους αριθμούς και θέλουμε να δημιουργήσουμε μια νέα λίστα με τα τετράγωνα αυτών:

```
numbers = [1, 2, 3, 4, 5]

# Σύνταξη με List Comprehension
squares = [x**2 for x in numbers]

print(squares) # Έξοδος: [1, 4, 9, 16, 25]
```

Κ 150: Παράδειγμα 1 Περιφραστικής Λίστας (List Comprehension) χωρίς συνθήκη

```
thislist = ["μήλο", "μπανάνα", "κεράσι"]
[print(x) for x in thislist]
```

Κ 151: Παράδειγμα 2 Περιφραστικής Λίστας (List Comprehension) χωρίς συνθήκη

Β. Μετασχηματισμός με Φιλτράρισμα (Με Συνθήκη)

Παράδειγμα: Έστω ότι θέλουμε να κρατήσουμε μόνο τα ονόματα των φρούτων που ξεκινούν από το γράμμα "μ".

```python
fruits = ["μήλο", "μπανάνα", "πορτοκάλι", "μάνγκο", "αχλάδι"]
# Σύνταξη με φίλτρο if
m_fruits = [f for f in fruits if f.startswith("μ")]
print(m_fruits) # Έξοδος: ['μήλο', 'μπανάνα', 'μάνγκο']
```

Κ 152: Παράδειγμα Περιφραστικής Λίστας (List Comprehension) με συνθήκη

Παράδειγμα: Έστω ότι έχουμε μια λίστα με ακαθάριστα ποσά και θέλουμε να κρατήσουμε μόνο όσα είναι πάνω από 100€, εφαρμόζοντας ταυτόχρονα μια έκπτωση 20%.

```python
gross_amounts = [120, 80, 250, 40, 150]

# Με list comprehension
discounted_prices = [price * 0.8 for price in gross_amounts if price > 100]
# Αποτέλεσμα: [96.0, 200.0, 120.0]
```

Κ 153: Δημιουργία Λίστας με Περιφραστικές Λίστες (List Comprehensions)

Ακολουθούν ενδεικτικά παραδείγματα με την χρήση συνθήκης ή μη:

```python
# List Comprehension
squares = [x**2 for x in range(10)]
print(squares) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# Επιλογή μόνο αν η συνθήκη (x % 2 == 0) είναι True
numbers = [8, 9, 11, 12, 13, 22, 25, 30]
evens = [x for x in numbers if x % 2 == 0]
print(evens) # [8, 12, 22, 30]

# Επεξεργασία και φιλτράρισμα μαζί
names = ["Γιώργος", "Άννα", "Μαρία", "Ανδρέας", "Αρετή"]
filtered_names = [name.upper() for name in names if name.startswith("Α")]
print(filtered_names) # ['ΑΝΝΑ', 'ΑΝΔΡΕΑΣ', 'ΑΡΕΤΗ']
```

Κ 154: Άλλα παραδείγματα με δημιουργία Λίστας με Περιφραστικές Λίστες (List Comprehensions)

Δυναμική Δημιουργία Πίνακα N x M

Η βέλτιστη πρακτική για τη δημιουργία ενός πίνακα N x M με μηδενικά είναι η χρήση List Comprehension:

```python
matrix = [[0 for j in range(M)] for i in range(N)]
```

Παράδειγμα:

```python
rows = 10
cols = 10
# Δημιουργία πίνακα 10x10 με μηδενικά
```

```
grid = [[0 for j in range(cols)] for i in range(rows)]
```
Κ 155: Δυναμική δημιουργία πίνακα με Περιφραστικές Λίστες

4.5.2. Περιφραστικά Λεξικά (Dictionary Comprehensions)

Αντίστοιχα με τις περιφραστικές λίστες, η Python υποστηρίζει και τη δημιουργία λεξικών μέσω περιφραστικής σύνταξης. Η τεχνική αυτή επιτρέπει τη δημιουργία ενός νέου λεξικού με συνοπτικό και ευανάγνωστο τρόπο, εφαρμόζοντας μετασχηματισμούς ή φίλτρα στα δεδομένα μιας υπάρχουσας δομής.

Γενική Σύνταξη:

```
new_dict = {key_expression: value_expression for item in iterable if condition}
```

- **Key_expression**: Η έκφραση που ορίζει το κλειδί κάθε εγγραφής.
- **Value_expression**: Η έκφραση που ορίζει την τιμή κάθε εγγραφής.
- **For item in iterable**: Η επανάληψη πάνω στα δεδομένα.
- **If condition (Προαιρετικό):** Ένα φίλτρο που καθορίζει ποιες εγγραφές θα συμπεριληφθούν.

Παράδειγμα: Έστω ότι έχουμε μια λίστα με ονόματα προϊόντων και θέλουμε να δημιουργήσουμε ένα λεξικό που αντιστοιχίζει κάθε προϊόν στο μήκος του ονόματός του.

```
# Με dictionary comprehension
products = ["laptop", "κινητό", "tablet", "ακουστικά"]
product_lengths = {product: len(product) for product in products}
# Αποτέλεσμα: {'laptop': 6, 'κινητό': 6, 'tablet': 6, 'ακουστικά': 9}
```
Κ 156: Δημιουργία Λεξικού με Περιφραστικά Λεξικά (Dictionary Comprehensions)

Ακολουθούν ενδεικτικά παραδείγματα με χρήση συνθήκης ή μη:

```
# Δημιουργία λεξικού με τετράγωνα αριθμών
squares = {x: x**2 for x in range(1, 6)}
print(squares)  # {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# Φιλτράρισμα: Μόνο προϊόντα με τιμή πάνω από 50€
prices = {"laptop": 800, "ποντίκι": 15, "οθόνη": 300, "καλώδιο": 5}
expensive = {item: price for item, price in prices.items() if price > 50}
print(expensive)  # {'laptop': 800, 'οθόνη': 300}
```
Κ 157: Άλλα παραδείγματα με δημιουργία Λεξικού με Περιφραστικά Λεξικά (Dictionary Comprehensions)

4.5.3. Ασκήσεις Περιφραστικές λίστες και λεξικά

Άσκηση 7: Περιφραστικές Λίστες

1. Να δημιουργηθεί, μέσω **List Comprehension**, μια λίστα που θα περιέχει τους **κύβους (x^3)** των αριθμών από το 1 έως το 10.
2. Να δημιουργηθεί μια λίστα `even_squares` που θα περιέχει τα τετράγωνα (x^2) **μόνο των ζυγών αριθμών** από το 1 έως το 20.

Άσκηση 8: Περιφραστικά Λεξικά

Δίνεται το λεξικό: `products = {"laptop": 800, "ποντίκι": 25, "οθόνη": 350, "καλώδιο": 8, "κάμερα": 120}`

μέσω **Περιφραστικών Λεξικών**, δημιούργησε λεξικό affordable που περιέχει μόνο τα προϊόντα με τιμή κάτω από 100€, με την τιμή τους μειωμένη κατά 10%.

4.6. Επίλυση Προβλημάτων με Λεξικά, Λίστες, Σύνολα και Πλειάδες

Η επιλογή της κατάλληλης δομής δεδομένων αποτελεί κρίσιμο βήμα στην επίλυση προβλημάτων, καθώς κάθε δομή προσφέρει διαφορετικά πλεονεκτήματα ανάλογα με τη φύση του προβλήματος. Στην ενότητα αυτή παρουσιάζονται χαρακτηριστικά σενάρια χρήσης που αναδεικνύουν πότε και γιατί επιλέγουμε κάθε δομή.

Πότε χρησιμοποιούμε κάθε δομή:

- **Λίστα (list)**: Όταν χρειαζόμαστε μια διατεταγμένη συλλογή στοιχείων που μπορεί να τροποποιηθεί (προσθήκη, αφαίρεση, ταξινόμηση). Ιδανική για ακολουθίες δεδομένων όπου η σειρά έχει σημασία.
- **Πλειάδα (tuple):** Όταν τα δεδομένα δεν πρέπει να αλλάξουν μετά τη δημιουργία τους. Ιδανική για σταθερές εγγραφές και δεδομένα που θέλουμε να προστατεύσουμε από τυχαία τροποποίηση.
- **Σύνολο (set):** Όταν χρειαζόμαστε μοναδικά στοιχεία ή θέλουμε να εκτελέσουμε πράξεις όπως ένωση, τομή και διαφορά μεταξύ συλλογών.
- **Λεξικό (dict)**: Όταν θέλουμε να αντιστοιχίσουμε κλειδιά σε τιμές, δηλαδή να αναζητούμε γρήγορα μια τιμή βάσει ενός μοναδικού αναγνωριστικού.

4.6.1. Πρόβλημα 1: Διαχείριση λίστας εργασιών

Δομή: Λίστα

Οι εργασίες πρέπει να διατηρούν τη σειρά προτεραιότητάς τους (διατεταγμένη δομή), ενώ ταυτόχρονα χρειαζόμαστε τη δυνατότητα προσθήκης νέων εργασιών και αφαίρεσης ολοκληρωμένων (μεταβλητή δομή). Η λίστα ικανοποιεί και τις δύο αυτές απαιτήσεις.

```
#Πρόβλημα 1 Διαχείριση λίστας εργασιών
tasks = ["Αγορά υλικών", "Σύνταξη αναφοράς", "Κλείσιμο ραντεβού"]

# Προσθήκη νέας εργασίας
tasks.append("Αποστολή email")

# Ολοκλήρωση εργασίας (αφαίρεση)
tasks.remove("Σύνταξη αναφοράς")

# Εμφάνιση εκκρεμών εργασιών
for i, task in enumerate(tasks, 1):
    print(f"{i}. {task}")
# Έξοδος:
# 1. Αγορά υλικών
# 2. Κλείσιμο ραντεβού
# 3. Αποστολή email
```

Κ 158: Πρόβλημα 1 Διαχείριση λίστας εργασιών (Δομή: Λίστα)

4.6.2. Πρόβλημα 2: Αποθήκευση γεωγραφικών συντεταγμένων

Δομή: Πλειάδα

Οι γεωγραφικές συντεταγμένες μιας πόλης είναι σταθερές τιμές που δεν πρέπει να τροποποιηθούν κατά τη διάρκεια εκτέλεσης του προγράμματος. Η αμεταβλητότητα (immutability) της πλειάδας εγγυάται την ακεραιότητα αυτών των δεδομένων. Επιπλέον, η πλειάδα μπορεί να χρησιμοποιηθεί ως κλειδί λεξικού, κάτι αδύνατο με λίστα λόγω της μεταβλητότητάς της.

```
#Πρόβλημα 2: Αποθήκευση γεωγραφικών συντεταγμένων
# Συντεταγμένες πόλεων (δεν αλλάζουν)
athens = (37.9838, 23.7275)
thessaloniki = (40.6401, 22.9444)

# Υπολογισμός απόστασης (απλοποιημένος)
import math
distance = math.sqrt((athens[0] - thessaloniki[0])**2 + (athens[1] -
thessaloniki[1])**2)
print(f"Απόσταση σε μοίρες: {distance:.2f}")
# Έξοδος: Απόσταση σε μοίρες: 2.79

# Πρακτική χρήση: πόλη → συντεταγμένες
cities = {
```

```
    "Αθήνα": (37.9838, 23.7275),
    "Θεσσαλονίκη": (40.6401, 22.9444)
}
print(cities["Αθήνα"])
# Έξοδος: (37.9838, 23.7275)

# Η πλειάδα μπορεί να χρησιμοποιηθεί και ως κλειδί
coordinates_to_city = {
    (37.9838, 23.7275): "Αθήνα",
    (40.6401, 22.9444): "Θεσσαλονίκη"
}
print(coordinates_to_city[(37.9838, 23.7275)])
# Έξοδος: Αθήνα
```
Κ 159: Πρόβλημα 2: Αποθήκευση γεωγραφικών συντεταγμένων (Δομή: Πλειάδα)

4.6.3. Πρόβλημα 3: Εύρεση κοινών και μοναδικών στοιχείων - Εγγεγραμμένοι Μαθητές

Δομή: Σύνολο

Κάθε μαθητής είναι μοναδικός — δεν μπορεί να εγγραφεί δύο φορές στο ίδιο μάθημα. Το σύνολο εξασφαλίζει αυτόματα τη μοναδικότητα των στοιχείων. Επιπλέον, οι ενσωματωμένες πράξεις συνόλων (τομή, ένωση, διαφορά) μας επιτρέπουν να απαντήσουμε εύκολα σε ερωτήματα όπως «ποιοι παρακολουθούν και τα δύο μαθήματα» χωρίς να γράψουμε σύνθετο κώδικα.

```
#Πρόβλημα 3: Εύρεση κοινών και μοναδικών στοιχείων
# Μαθητές εγγεγραμμένοι σε δύο μαθήματα
python_students = {"Μαρία", "Γιώργος", "Άννα", "Νίκος", "Ελένη"}
data_students = {"Άννα", "Δημήτρης", "Γιώργος", "Κατερίνα"}

# Ποιοι παρακολουθούν και τα δύο μαθήματα;
common = python_students & data_students
print(f"Και στα δύο: {common}")
# Έξοδος: Και στα δύο: {'Άννα', 'Γιώργος'}

# Ποιοι παρακολουθούν μόνο Python;
only_python = python_students - data_students
print(f"Μόνο Python: {only_python}")
# Έξοδος: Μόνο Python: {'Μαρία', 'Νίκος', 'Ελένη'}

# Πόσοι μοναδικοί μαθητές συνολικά;
all_students = python_students | data_students
print(f"Σύνολο μαθητών: {len(all_students)}")
# Έξοδος: Σύνολο μαθητών: 7
```
Κ 160: Πρόβλημα 3: Εύρεση κοινών και μοναδικών στοιχείων - Εγγεγραμμένοι Μαθητές (Δομή: Σύνολο)

4.6.4. Πρόβλημα 4: Τηλεφωνικός κατάλογος επαφών

Δομή: Λεξικό

Κάθε επαφή αναγνωρίζεται από ένα μοναδικό όνομα (κλειδί) και αντιστοιχεί σε έναν αριθμό τηλεφώνου (τιμή). Το λεξικό μας επιτρέπει να αναζητούμε, να προσθέτουμε και να ενημερώνουμε επαφές γρήγορα βάσει ονόματος.

```
#Πρόβλημα 4: Τηλεφωνικός κατάλογος επαφών
# Τηλεφωνικός κατάλογος
contacts = {
    "Μαρία": "6971234567",
    "Γιώργος": "6989876543",
    "Άννα": "6955551234"
}

# Αναζήτηση επαφής
name = "Μαρία"
print(f"Τηλέφωνο {name}: {contacts[name]}")
# Έξοδος: Τηλέφωνο Μαρία: 6971234567

# Προσθήκη νέας επαφής
contacts["Νίκος"] = "6941112233"

# Ενημέρωση υπάρχουσας επαφής
contacts["Γιώργος"] = "6980001111"

# Έλεγχος αν υπάρχει μια επαφή
search = "Δημήτρης"
if search in contacts:
    print(f"Βρέθηκε: {contacts[search]}")
else:
    print(f"Η επαφή '{search}' δεν βρέθηκε.")
# Έξοδος: Η επαφή 'Δημήτρης' δεν βρέθηκε.

# Εμφάνιση όλων των επαφών
for name, phone in contacts.items():
    print(f"{name}: {phone}")
# Έξοδος:
# Μαρία: 6971234567
# Γιώργος: 6980001111
# Άννα: 6955551234
# Νίκος: 6941112233
```

Κ 161: Πρόβλημα 4: Τηλεφωνικός κατάλογος επαφών (Δομή: Λεξικό)

4.6.5. Πρόβλημα 5: Κατάλογος προϊόντων με σταθερές κατηγορίες

Συνδυασμός δομών: Λεξικό + Πλειάδες + Λίστες

Οι κατηγορίες προϊόντων είναι προκαθορισμένες και δεν πρέπει να αλλάξουν — γι' αυτό αποθηκεύονται σε πλειάδα. Τα προϊόντα κάθε κατηγορίας, αντίθετα, αλλάζουν συνεχώς (προσθήκη, αφαίρεση) — γι' αυτό αποθηκεύονται σε λίστες. Το λεξικό συνδέει τα δύο, αντιστοιχίζοντας κάθε κατηγορία στα προϊόντα της.

```
#Πρόβλημα 5: Κατάλογος προϊόντων με σταθερές κατηγορίες
# Πλειάδες για σταθερές κατηγορίες (δεν αλλάζουν)
categories = ("Ηλεκτρονικά", "Ένδυση", "Τρόφιμα")

# Λεξικό: κατηγορία → λίστα προϊόντων (τροποποιήσιμη)
catalog = {
    categories[0]: ["Laptop", "Tablet"],
    categories[1]: ["Μπλούζα", "Παντελόνι"],
    categories[2]: ["Γάλα", "Ψωμί"]
}

# Προσθήκη νέου προϊόντος σε κατηγορία
catalog[categories[0]].append("Κινητό")

# Εμφάνιση καταλόγου
for category, products in catalog.items():
    print(f"\n{category}:")
    for product in products:
        print(f"  - {product}")
# Έξοδος:
# Ηλεκτρονικά:
#   - Laptop
#   - Tablet
#   - Κινητό
# Ένδυση:
#   - Μπλούζα
#   - Παντελόνι
# Τρόφιμα:
#   - Γάλα
#   - Ψωμί
```

Κ 162: Πρόβλημα 5: Κατάλογος προϊόντων με σταθερές κατηγορίες (Συνδυασμός δομών: Λεξικό + Πλειάδες + Λίστες)

4.6.6. Πρόβλημα 6: Σύστημα διαχείρισης μαθητών

Συνδυασμός δομών: Λεξικό + Λεξικά

Το εξωτερικό λεξικό μας δίνει γρήγορη πρόσβαση στα δεδομένα κάθε μαθητή μέσω του ονόματός του (κλειδί). Κάθε μαθητής αντιστοιχεί σε ένα εσωτερικό λεξικό όπου τα κλειδιά είναι τα μαθήματα και οι τιμές είναι οι βαθμοί. Η εμφωλευμένη αυτή δομή (λεξικό μέσα σε λεξικό) μας επιτρέπει να οργανώσουμε σύνθετα δεδομένα με φυσικό και ευανάγνωστο τρόπο.

```
#Πρόβλημα 6: Σύστημα διαχείρισης μαθητών
# Λεξικό: μαθητής → {μάθημα → βαθμός}
students = {
    "Μαρία": {
        "Μαθηματικά": 18,
        "Φυσική": 17,
        "Πληροφορική": 20
    },
    "Γιώργος": {
        "Μαθηματικά": 12,
        "Φυσική": 13,
        "Πληροφορική": 15
    },
    "Άννα": {
        "Μαθηματικά": 19,
        "Φυσική": 20,
        "Πληροφορική": 19
    }
}

# Βαθμολογίες κάθε μαθητή ανά μάθημα
for name, courses in students.items():
    print(f"\n{name}:")
    for course, grade in courses.items():
        print(f"  {course}: {grade}")
# Έξοδος:
# Μαρία:
#   Μαθηματικά: 18
#   Φυσική: 17
#   Πληροφορική: 20
# Γιώργος:
#   Μαθηματικά: 12
#   Φυσική: 13
#   Πληροφορική: 15
# Άννα:
#   Μαθηματικά: 19
#   Φυσική: 20
#   Πληροφορική: 19

# Γενικός μέσος όρος κάθε μαθητή
print("\nΓενικοί Μέσοι Όροι:")
```

```
for name, courses in students.items():
    avg = sum(courses.values()) / len(courses)
    print(f"  {name}: {avg:.1f}")
# Έξοδος:
# Γενικοί Μέσοι Όροι:
#   Μαρία: 18.3
#   Γιώργος: 13.3
#   Άννα: 19.3
```
Κ 163: Πρόβλημα 6: Σύστημα διαχείρισης μαθητών (Συνδυασμός δομών: Λεξικό + Λεξικά)

4.7. Μορφοποίηση λεκτικών με διάφορους τρόπους

Η Python παρέχει πολλαπλούς τρόπους μορφοποίησης λεκτικών (strings), επιτρέποντας την ενσωμάτωση μεταβλητών και εκφράσεων μέσα σε κείμενο με ευανάγνωστο και ευέλικτο τρόπο. Οι κυριότερες μέθοδοι μορφοποίησης είναι οι εξής:

4.7.1. f-strings (Formatted String Literals)

Αποτελεί τον πιο σύγχρονο και ευανάγνωστο τρόπος μορφοποίησης. Το πρόθεμα f τοποθετείται πριν από το λεκτικό και εισάγονται μεταβλητές ή εκφράσεις μέσα σε αγκύλες {}.

Γενική σύνταξη:

```
f"κείμενο {έκφραση:μορφοποίηση}"
```

Όπου:

- **έκφραση**: Οποιαδήποτε μεταβλητή ή οποιοσδήποτε έγκυρος κώδικας Python που επιστρέφει τιμή (μεταβλητή, πράξη, κλήση συνάρτησης/μεθόδου, συνθήκη κ.λπ.)..
- **:μορφοποίηση** (Προαιρετικό): Προσδιοριστής μορφοποίησης, π.χ. `:.2f` για 2 δεκαδικά, `:>10` για στοίχιση.

Παραδείγματα χρήσης:

```
name = "Μαρία"
age = 28
# χρήση σε print
print(f"Η {name} είναι {age} ετών.") # Έξοδος: Η Μαρία είναι 28 ετών.

# Ανάθεση σε μεταβλητή
greeting = f"Γεια σου {name}!"
print(greeting) # Έξοδος Γεια σου Μαρία!

# Σε λίστα
```

```
messages = [f"Χρήστης: {name}", f"Ηλικία: {age}"]
print(messages) # Έξοδος ['Χρήστης: Μαρία', 'Ηλικία: 28']

# Σε λεξικό
data = {f"user_{name}": age}
print(data) # Έξοδος {'user_Μαρία': 28}

# Σε συνάρτηση
def get_info(name, age):
    return f"Ο/Η {name} είναι {age} ετών." # Έξοδος: Ο/Η Μαρία είναι 28 ετών.

# Σε εγγραφή αρχείου
with open("output.txt", "w") as f_out:
    f_out.write(f"Όνομα: {name}, Ηλικία: {age}\n") # Έξοδος στο αρχείο: Όνομα: Μαρία,
Ηλικία: 28

# Σε συνθήκη
if f"{name}" in ["Μαρία", "Άννα"]:
    print("Βρέθηκε!") # Έξοδος: Βρέθηκε!

# Υποστηρίζει εκφράσεις και μορφοποίηση αριθμών
price = 49.567
print(f"Τιμή: {price:.2f}€") # Έξοδος: Τιμή: 49.57€

# Χρησιμοποιείται για στοίχιση
print(f"Τιμή: {price:>10}€") # Έξοδος: Τιμή:     49.567€
```

Κ 164: Χρήση της f-strings σε λεκτικά

4.7.2. Μέθοδος format()

Η μέθοδος format χρησιμοποιεί placeholders `{}` και περνάει τα δεδομένα ως ορίσματα στο τέλος της συμβολοσειράς. Είναι χρήσιμη όταν θέλουμε να διαχωρίσουμε το πρότυπο του κειμένου από τα δεδομένα. Η μέθοδος format ακολουθεί την σύνταξη:

```
"κείμενο {} κείμενο {}".format(τιμή1, τιμή2)
```

όπου: **{}**: Θέσεις υποκατάστασης (placeholders) που αντικαθίστανται από τα ορίσματα της `format()`.

Παράδειγμα

```
name = "Γιώργος"
age = 35
print("Ο {} είναι {} ετών.".format(name, age))
# Έξοδος: Ο Γιώργος είναι 35 ετών.
```

*Κ 165: Η μέθοδος format σε λεκτικά όπου **{}**: Θέσεις υποκατάστασης*

ή με δείκτες θέσης:

```
"κείμενο {0} κείμενο {1}".format(τιμή1, τιμή2)
```

όπου: **{0}, {1}**: Αριθμητικοί δείκτες θέσης για αναφορά σε συγκεκριμένο όρισμα.

Παράδειγμα:

```
name = "Γιώργος"
age = 35
# Με αριθμητικούς δείκτες θέσης
print("Η {0} μένει στην {1} και η {0} εργάζεται εκεί.".format("Άννα", "Θεσσαλονίκη"))
# Έξοδος: Η Άννα μένει στην Θεσσαλονίκη και η Άννα εργάζεται εκεί.
```

Κ 166: Η μέθοδος format σε λεκτικά όπου {0}, {1}: Αριθμητικοί δείκτες θέσης για αναφορά σε συγκεκριμένο όρισμα

ή με ονοματισμένες παραμέτρους:

```
"κείμενο {όνομα1} κείμενο {όνομα2}".format(όνομα1=τιμή1, όνομα2=τιμή2)
```

όπου: **{όνομα}**: Ονοματισμένη παράμετρος.

Παράδειγμα:

```
# Με ονοματισμένες παραμέτρους
print("Προϊόν: {product}, Τιμή: {price:.2f}€".format(product="Laptop", price=899.5))
# Έξοδος: Προϊόν: Laptop, Τιμή: 899.50€
```

*Κ 167: Η μέθοδος format σε λεκτικά όπου **{όνομα}**: Ονοματισμένη παράμετρος*

Παραδείγματα

```
# Ανάθεση σε μεταβλητή
greeting = "Γεια σου {}!".format(name) # Έξοδος: Γεια σου Γιώργος!

# Επιστροφή τιμής από συνάρτηση
def get_info(name, age):
    return "Ο/Η {} είναι {} ετών.".format(name, age) # Έξοδος: Ο/Η Γιώργος είναι 35
ετών.

# Σε λίστα
messages = ["Χρήστης: {}".format(name), "Ηλικία: {}".format(age)] #Εξοδος: ['Χρήστης:
Γιώργος', 'Ηλικία: 35']

# Εγγραφή σε αρχείο
with open("output.txt", "w") as file:
    file.write("Όνομα: {}, Ηλικία: {}\n".format(name, age)) # Έξοδος στο αρχείο: Όνομα:
Γιώργος, Ηλικία: 35

# Στην print (απλά ένα από τα πολλά σημεία χρήσης)
print("Ο {} είναι {} ετών.".format(name, age)) # Έξοδος: Ο Γιώργος είναι 35 ετών.
```

Κ 168: Παραδείγματα Χρήσης μεθόδου format σε λεκτικά

4.7.3. Τελεστής %

Η χρήση του τελεστή % αποτελεί τον παλαιότερο τρόπο (τύπου C) εφαρμογής μορφοποίησης σε λεκτικά. Η σύνταξη είναι:

```
"κείμενο %s κείμενο %d κείμενο %f" % (τιμή1, τιμή2, τιμή3)
```

Όπου οι χαρακτήρες υποκατάστασης είναι:

- **%s**: Λεκτικό (string)
- **%d**: Ακέραιος αριθμός (integer)
- **%f**: Δεκαδικός αριθμός (float)
- **%.2f**: Δεκαδικός με 2 δεκαδικά ψηφία

Παραδείγματα

```
name = "Μαρία"
age = 28
price = 49.567

# Ανάθεση σε μεταβλητή
greeting = "Γεια σου %s!" % name
print(greeting) # Έξοδος: Γεια σου Μαρία!

# Με πολλαπλές τιμές (χρήση πλειάδας)
info = "Η %s είναι %d ετών." % (name, age)
print(info) # Έξοδος: Η Μαρία είναι 28 ετών.

# Μορφοποίηση δεκαδικών
formatted_price = "Τιμή: %.2f€" % price
print(formatted_price) # Έξοδος: Τιμή: 49.57€

# Επιστροφή τιμής από συνάρτηση
def get_info(name, age):
    return "Ο/Η %s είναι %d ετών." % (name, age)

result = get_info("Μαρία", 28)
print(result) # Έξοδος: Ο/Η Μαρία είναι 28 ετών.

# Σε λίστα
messages = ["Χρήστης: %s" % name, "Ηλικία: %d" % age]
print(messages) # Έξοδος: ['Χρήστης: Μαρία', 'Ηλικία: 28']

# Εγγραφή σε αρχείο
with open("output.txt", "w") as file:
    file.write("Όνομα: %s, Ηλικία: %d\n" % (name, age))
# Στο αρχείο output.txt γράφεται: Όνομα: Μαρία, Ηλικία: 28
```

```
# Συνδυασμός %s, %d, %f
print("Η %s, ηλικίας %d, πλήρωσε %.2f€." % (name, age, price))
# Έξοδος: Η Μαρία, ηλικίας 28, πλήρωσε 49.57€.
```
Κ 169: Εφαρμογή Μορφοποίησης σε λεκτικά με τη χρήση του τελεστή %

4.7.4. Ασκήσεις Μορφοποίηση λεκτικών με διάφορους τρόπους

Άσκηση 9 — f-strings

Δίνονται τα παρακάτω δεδομένα:

```
name = "Ελένη"
age = 22
balance = 1534.678
products = ["laptop", "ποντίκι", "οθόνη"]
```
Χρησιμοποιώντας **αποκλειστικά f-strings**, εκτύπωσε:

1. Καλωσόρισες, Ελένη! Είσαι 22 ετών.
2. Το υπόλοιπό σου είναι: 1534.68€ *(2 δεκαδικά)*
3. Προϊόν 1: laptop — για **κάθε** προϊόν με for και f-string

Άσκηση 10 — Μέθοδος format()

Δίνονται τα παρακάτω δεδομένα:

```
city = "Θεσσαλονίκη"
country = "Ελλάδα"
temperature = 23.456
```
Χρησιμοποιώντας **αποκλειστικά** τη μέθοδο format(), εκτύπωσε:

1. **Η Θεσσαλονίκη βρίσκεται στην Ελλάδα.** — με **απλά** {}
2. **Η Θεσσαλονίκη είναι στην Ελλάδα και η Θεσσαλονίκη είναι όμορφη.** — με **αριθμητικούς δείκτες** {0}, {1}
3. **Πόλη: Θεσσαλονίκη, Χώρα: Ελλάδα, Θερμοκρασία: 23.46°C** — με **ονοματισμένες παραμέτρους** και 2 δεκαδικ

Άσκηση 11 — Τελεστής %

Δίνονται τα παρακάτω δεδομένα:

```
product = "Laptop"
quantity = 3
unit_price = 799.999
```

Χρησιμοποιώντας **αποκλειστικά** τον τελεστή %, εκτύπωσε:

1. **Προϊόν: Laptop** — με %s
2. **Τεμάχια: 3** — με %d
3. **Τιμή μονάδας: 800.00€** — με %.2f
4. **Αγόρασες 3 τεμάχια Laptop συνολικής αξίας 2400.00€** — συνδυασμός %d, %s, %.2f

4.8. Ερωτήσεις Κλειστού Τύπου

1. Ποια από τις παρακάτω εντολές δημιουργεί μια πλειάδα με **ένα στοιχείο**;
   A. t = (5)
   B. t = (5,)
   C. t = tuple(5)
   D. t = [5]

2. Τι τιμή έχει η μεταβλητή `others` στον παρακάτω κώδικα:

```
grades = (10, 9, 8, 7, 6, 5)
first, second, *others = grades
```

   A. (8, 7, 6, 5)
   B. [8, 7, 6, 5]
   C. 8
   D. [10, 9]

3. Ποιο είναι το αποτέλεσμα του παρακάτω κώδικας:

```
mixed = (1, 2, [3, 4])
mixed[2][0] = 99
print(mixed)
```

   A. TypeError - οι πλειάδες είναι αμετάβλητες
   B. (1, 2, [99, 4])
   C. (1, 2, [3, 4]) - η αλλαγή αγνοείται
   D. (1, 2, 99, 4)

4. Ποιο είναι το αποτέλεσμα του παρακάτω κώδικα;

```
s = {1, 2, 2, 3, True, False, 0}
print(len(s))
```

   A. 7
   B. 5
   C. 3
   D. 4

5. Τι επιστρέφει η έκφραση: A & B στα σύνολα:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
```

   A. {3, 4}
   B. {1, 2, 5, 6}
   C. {1, 2, 3, 4, 5, 6}
   D. {1, 2}

4.9. Ασκήσεις προς επίλυση

Άσκηση 1: Δημιούργησε ένα απλό σύστημα ελέγχου αδειών:

```
# Ορισμός ρόλων
admin_permissions = {"read", "write", "delete", "execute", "manage_users"}
editor_permissions = {"read", "write", "execute"}
viewer_permissions = {"read"}

# Τρέχων χρήστης
current_user_permissions = {"read", "write"}
```

Γράψε συναρτήσεις που:

1. Ελέγχουν αν ο χρήστης έχει όλες τις άδειες ενός editor
2. Βρίσκουν ποιες άδειες λείπουν για να γίνει admin
3. Ελέγχουν αν έχει τουλάχιστον τις άδειες viewer
4. Προτείνουν τον κατάλληλο ρόλο (admin/editor/viewer) με βάση τις άδειες

Άσκηση 2: Σύστημα Διαχείρισης Βιβλιοθήκης: Δίνεται το παρακάτω λεξικό που αναπαριστά μια βιβλιοθήκη:

```
library = {
    "978-0-13-468599-1": {
        "τίτλος": "Python Crash Course",
        "συγγραφέας": "Eric Matthes",
        "έτος": 2019,
        "διαθέσιμα": 3
    },
    "978-1-59327-584-6": {
        "τίτλος": "Automate the Boring Stuff",
        "συγγραφέας": "Al Sweigart",
        "έτος": 2020,
        "διαθέσιμα": 0
    },
```

```
    "978-1-491-95036-8": {
        "τίτλος": "Fluent Python",
        "συγγραφέας": "Luciano Ramalho",
        "έτος": 2022,
        "διαθέσιμα": 5
    }
}
```

1. Εμφανίστε τον τίτλο και συγγραφέα κάθε βιβλίου χρησιμοποιώντας βρόχο for με items()
2. Προσθέστε ένα νέο βιβλίο με ISBN "978-0-596-51774-8", τίτλο "Learning Python", συγγραφέα "Mark Lutz", έτος 2013, διαθέσιμα 2
3. Ενημερώστε τα διαθέσιμα αντίτυπα του "Fluent Python" από 5 σε 4 (δανεισμός)
4. Βρείτε και εκτυπώστε τα βιβλία που δεν είναι διαθέσιμα (διαθέσιμα == 0)
5. Διαγράψτε με pop() το βιβλίο με ISBN "978-1-59327-584-6" και εκτύπωσε τον τίτλο που αφαιρέθηκε
6. Υπολογίστε και εμφανίστε τον μέσο όρο των διαθέσιμων αντιτύπων

ΚΕΦΑΛΑΙΟ 5: ΔΙΑΧΕΙΡΙΣΗ ΑΡΧΕΙΩΝ ΔΙΑΦΟΡΩΝ ΤΥΠΩΝ

Η κύρια μνήμη των υπολογιστών (RAM), με τις διάφορες τεχνολογίες που χρησιμοποιούνται για την κατασκευή της μέχρι και σήμερα, χάνει τα περιεχόμενά της με την απώλεια της ηλεκτρικής ισχύος. Για τη μόνιμη διατήρηση των δεδομένων ανεξάρτητα της παροχής ρεύματος, χρησιμοποιούνται βοηθητικά αποθηκευτικά μέσα, όπως οι μαγνητικοί σκληροί δίσκοι (HDD, Hard Disk Drives), οι δίσκοι με τεχνολογία μνήμης flash (SSD, Solid State Disks), αντίστοιχες συσκευές που συνδέονται στο δίαυλο USB και, αρκετά σπανιότερα πια, οπτικοί δίσκοι (π.χ. DVD).

Συχνή είναι και η απομακρυσμένη αποθήκευση σε κάποια υπηρεσία νέφους (cloud), με υπηρεσίες όπως Google Drive, OneDrive, Dropbox κλπ., που προσφέρει άλλες δυνατότητες, αλλά έχει και μειονεκτήματα.

Οι πληροφορίες αποθηκεύονται σε αυτά τα μέσα με τη μορφή *αρχείου*. Τα περιεχόμενα ενός αρχείου μπορεί για παράδειγμα να είναι κείμενο που μπορεί να διαβαστεί (ίσως με τη χρήση κατάλληλου λογισμικού), μπορεί να είναι ο πηγαίος κώδικας ενός προγράμματος, ο εκτελέσιμος κώδικας μιας εφαρμογής, δεδομένα σε μορφή κειμένου ή δεδομένα σε δυαδική μορφή.

Συνήθως τα αρχεία έχουν ένα όνομα που παραπέμπει στα περιεχόμενά τους ή/και τη χρήση τους, ενώ ομαδοποιούνται κάτω από *φακέλους*, που και σε αυτούς δίνεται ένα όνομα. Η χρήση τους γίνεται μέσω ενός *συστήματος αρχείων (File System)*, το οποίο μπορεί να εξαρτάται από το λειτουργικό σύστημα και το ίδιο το μέσο αποθήκευσης. Μερικά γνωστά συστήματα αρχείων είναι το NTFS και το FAT σε διάφορες μορφές (για Windows), το APFS (για Apple Mac OS) και τα ext3 και ext4 (κυρίως για Linux).

5.1. Εργασία με αρχεία

Επειδή, όπως αναφέρθηκε, τα αρχεία αποθηκεύονται σε εξωτερική μνήμη, για να μπορέσει να γίνει κάποια επεξεργασία στα περιεχόμενα ενός αρχείου πρέπει να ακολουθηθεί συγκεκριμένη διαδικασία.

Πρώτα πρέπει να **ανοιχτεί** το αρχείο. Ουσιαστικά αυτό είναι μια ειδοποίηση προς το λειτουργικό σύστημα ότι θα γίνει χρήση του αρχείου.

Κατόπιν γίνεται η επεξεργασία του αρχείου. Μπορεί να είναι απλή **ανάγνωση** των περιεχομένων του, **ενημέρωση** ή **εγγραφή** νέων δεδομένων.

Τέλος, όταν ολοκληρωθεί η εργασία, το αρχείο πρέπει να **κλειστεί**.

Χωρίς να γίνει εμβάθυνση, θα αναφερθεί εδώ ότι όλη αυτή η διαδικασία είναι απαραίτητη επειδή η εξωτερική μνήμη είναι *πόρος* του λειτουργικού συστήματος και ένα πρόγραμμα που τρέχει ουσιαστικά ζητάει από το λειτουργικό σύστημα πρόσβαση σε αυτόν τον πόρο. Το λειτουργικό σύστημα λειτουργεί σαν ενδιάμεσος και στη διαδικασία της όποιας επεξεργασίας του αρχείου.

5.1.1. Άνοιγμα αρχείων κειμένου για ανάγνωση

Τα αρχεία κειμένου είναι αρχεία που περιέχουν μόνο συμβολοσειρές και δεν έχουν κανενός είδους μορφοποίηση. Μπορούν να διαβαστούν (αλλά και να επεξεργαστούν) με τη χρήση προγραμμάτων όπως το Σημειωματάριο (Notepad), το Notepad++ (πολύ διαδεδομένος διορθωτής κειμένου), ακόμα και το VS Code. Επίσης μπορούν να εμφανιστούν σε παράθυρο γραμμής εντολής (π.χ. PowerShell) με μια απλή εντολή εμφάνισης περιεχομένων όπως η `cat` (ή `Get-Content`).

Παλαιότερα τα αρχεία κειμένου περιορίζονταν στο σύνολο χαρακτήρων ASCII (αγγλικό αλφάβητο, αριθμοί, σημεία στίξης και σύμβολα) και μπορούσε να επεκταθεί σε κάποια γλώσσα κωδικοποιημένη σαν επέκταση του ASCII με κατάλληλες ρυθμίσεις και επεκτάσεις, συνήθως ασύμβατες μεταξύ τους.

Το πρόβλημα λύθηκε με τη γενική υιοθέτηση του πρότυπου Unicode, που υποστηρίζει από το σχεδιασμό του πάρα πολλές γλώσσες. Η πιο διαδεδομένη κωδικοποίηση Unicode υλοποιείται μέσω της κωδικοποίησης UTF-8.

Η Python αρχικά στις συμβολοσειρές χρησιμοποιούσε χαρακτήρες ASCII και για να χειριστεί συμβολοσειρά Unicode είχε χωριστό τύπο, τον τύπο `unicode`. Αυτό άλλαξε με την έκδοση 3, όπου οι συμβολοσειρές είναι όλες κωδικοποιημένες σαν UTF-8. Αυτό επιτρέπει την επεξεργασία αρχείων κειμένου ανεξάρτητα από τη γλώσσα του περιεχομένου τους.

Για να ανοίξει στην Python ένα αρχείο για επεξεργασία χρησιμοποιείται η εντολή `open()`. Στον κώδικα Κ. 170 ανοίγεται ένα αρχείο με επαφές, γίνεται ανάγνωση του περιεχομένου του με την `read()` και κάθε γραμμή του μεταφέρεται στο λεξικό `contacts_dict`. Στο τέλος τυπώνονται τα περιεχόμενα του λεξικού (Ε. 30).

```
contacts_file = open("D:/Python_A/contacts1.txt", "r", encoding="utf-8")
contacts = contacts_file.read()
contacts_dict = {}
for line in contacts.split("\n"):
    contact = line.split(":")
    contacts_dict[contact[0]] = contact[1].strip()
contacts_file.close()
```

```
print(contacts_dict)
```
Κ. 171 – Άνοιγμα αρχείου κειμένου και ανάγνωση των περιεχομένων του με τη read()

```
{'Ελένη': '2101111', 'Χρήστος': '69702222', 'Γιώργος': '2103333', 'Αντωνία': '69704444'}
```
Ε. 30 – Αποτέλεσμα ανάγνωσης του αρχείου.

Όπως φαίνεται στον κώδικα, η `read()` έχει τρεις παραμέτρους:

- Το όνομα του αρχείου που θα ανοιχτεί, το οποίο μπορεί να περιέχει την πλήρη διαδρομή του αρχείου.
- Τον τρόπο ανοίγματος του αρχείου. (προαιρετική)
- Την κωδικοποίηση των χαρακτήρων του αρχείου. (προαιρετική)

Αν στο πρώτο όρισμα περιέχεται μόνο το όνομα του αρχείου, τότε θεωρείται ότι βρίσκεται στον τρέχοντα φάκελο. Αν η διαδρομή ξεκινάει με «`/`» (ή «`C:/`», « `D:/`» κλπ. για Windows), τότε η διαδρομή είναι *απόλυτη*: περιέχει την πλήρη διαδρομή από τη ρίζα του συστήματος αρχείων μέχρι το οριζόμενο αρχείο. Αυτή η διαδρομή είναι μονοσήμαντη και καθορίζει απόλυτα τη θέση του αρχείου. Αν η διαδρομή *δεν* ξεκινάει με «`/`» (ή «`C:/`», « `D:/`» κλπ. για Windows), τότε είναι *σχετική* και ξεκινάει από τον τρέχοντα φάκελο. Για παράδειγμα, αν το πρόγραμμα εκτελείται από το φάκελο `/home/user1` και η διαδρομή είναι «`data-files/file1.txt`», τότε η πλήρης διαδρομή του αρχείου είναι η `/home/user1/data-files/file1.txt`.

Γενικά στην Python προτιμάται να χρησιμοποιείται ο χαρακτήρας «/» για τις διαδρομές στο σύστημα αρχείων και όχι ο «\» που χρησιμοποιείται στα Windows.

Ο τρόπος ανοίγματος του αρχείου μπορεί να είναι ένας από τους εξής:

- «`r`»: read, το αρχείο ανοίγεται για ανάγνωση των περιεχομένων του. Αυτή είναι και η προκαθορισμένη επιλογή, αν δεν περαστεί το όρισμα στην κλήση της `open()`. Αν το αρχείο δεν υπάρχει, προκαλείται λάθος.
- «`w`»: write, το αρχείο ανοίγεται για εγγραφή. Αν το αρχείο δεν υπάρχει, δημιουργείται. Αν το αρχείο υπάρχει, τότε χάνονται όλα τα περιεχόμενα που τυχόν είχε.
- «`x`»: το αρχείο δημιουργείται και ανοίγεται για εγγραφή. Αν υπάρχει ήδη, προκαλείται η εξαίρεση `FileExistsError`.
- «`a`»: append, το αρχείο ανοίγεται για προσθήκη (στο τέλος). Τα ήδη υπάρχοντα περιεχόμενα παραμένουν και οι νέες πληροφορίες γράφονται μετά από αυτές. Αν το αρχείο δεν υπάρχει, δημιουργείται.

- «+»: άνοιγμα για ενημέρωση (ανάγνωση και εγγραφή). Πριν από αυτό πρέπει να υπάρχει «r» («r+») ή «w» («w+»). Στην περίπτωση «w+» τυχόν ήδη υπάρχοντα περιεχόμενα διαγράφονται. Στην περίπτωση «r+» τυχόν ήδη υπάρχοντα περιεχόμενα παραμένουν (αρκεί να αναγνωστούν κάποια) και τα νέα περιεχόμενα γράφονται μετά από τα ήδη υπάρχοντα.
- «t»: text, το αρχείο θα επεξεργαστεί σαν αρχείο κειμένου. Είναι η προκαθορισμένη επιλογή και τα περιεχόμενα του αρχείου αντιμετωπίζονται σαν κείμενο (συμβολοσειρές). Μεταξύ άλλων, αυτό σημαίνει ότι είναι πιθανό να γίνεται μετάφραση ανάμεσα στην κωδικοποίηση των συμβολοσειρών στο αρχείο και στην κωδικοποίηση Unicode της Python, αλλά και ανάμεσα στις συμβολοσειρές που χρησιμοποιεί το λειτουργικό σύστημα για το τέλος γραμμής κειμένου και αυτών που χρησιμοποιεί η Python ('\n') . Το «t», αν υπάρχει, πρέπει να είναι μετά το «r» ή «w» και το «+», π.χ. «r+t».
- «b»: binary, το αρχείο θα επεξεργαστεί σαν αρχείο με δυαδικά δεδομένα. Δεν γίνεται καμία μετάφραση στα περιεχόμενά του κατά την εγγραφή ή την ανάγνωσή τους. Δηλώνεται στην ίδια θέση με το «t», π.χ. «r+b».

Αν δίνεται η προαιρετική παράμετρος της κωδικοποίησης, τότε είναι αυτή που θα χρησιμοποιηθεί για να αναγνωστούν τα υπάρχοντα δεδομένα του αρχείου ή να εγγραφούν τα νέα δεδομένα, για αρχεία που ανοίγονται σαν αρχεία κειμένου. Είναι κι αυτή μια μετάφραση που γίνεται στα περιεχόμενα του αρχείου. Αν δε δοθεί η παράμετρος, τότε σαν προκαθορισμένη επιλογή χρησιμοποιείται η ορισμένη στο λειτουργικό σύστημα κωδικοποίηση, η οποία όμως μπορεί να μην είναι ίδια με αυτή των περιεχομένων του αρχείου. Ένας τρόπος για να εμφανιστεί η προκαθορισμένη κωδικοποίηση του συστήματος είναι να δοθούν στο REPL οι εντολές:

```
import  locale

locale.getpreferredencoding()
```

Πρέπει να σημειωθεί ότι η χρήση της `read()` ενδείκνυται μόνο για αρχεία σχετικά μικρού μεγέθους, επειδή διαβάζονται *όλα* τα περιεχόμενα του αρχείου που ανοίχτηκε και μεταφέρονται στη μνήμη RAM του υπολογιστή. Το τι ακριβώς σημαίνει «μικρό μέγεθος» εξαρτάται τόσο από σταθερά μεγέθη, όπως το συνολικό μέγεθος της μνήμης RAM του υπολογιστή, όσο και από μεταβλητά, όπως οι υπόλοιπες διεργασίες που τυχόν τρέχουν στον υπολογιστή και οι δικές τους απαιτήσεις σε μνήμη RAM, ώστε να τρέχουν χωρίς προβλήματα.

Η τελευταία εντολή στον παραπάνω κώδικα (contacts_file.close) είναι απαραίτητη για την τακτική ολοκλήρωση της επεξεργασίας του αρχείου. Αν δεν εκτελεστεί, το αρχείο μπορεί να μείνει με εκκρεμείς εργασίες που δεν εκτελέστηκαν και να πάψουν να είναι σωστά τα περιεχόμενά του. Αυτό είναι ακόμα πιο κρίσιμο για αρχεία που ανοίγονται για κάποιου είδους εγγραφή.

Μια εκδοχή του παραπάνω προγράμματος, που διαβάζει το αρχείο μια γραμμή τη φορά φαίνεται στον κώδικα Κ. 172:

```
contacts_file = open("D:/Python_A/contacts1.txt", "r", encoding="utf-8")
contacts_dict = {}
for line in contacts_file:
    contact = line.split(":")
    contacts_dict[contact[0]] = contact[1].strip()
contacts_file.close()

print(contacts_dict) # {'Ελένη': ' 2101111', 'Χρήστος': ' 69702222', 'Γιώργος': ' 2103333',
'Αντωνία': ' 69704444'}
```

Κ. 172 - Άνοιγμα αρχείου για ανάγνωση ανά γραμμή.

Η δομή του αρχείου contacts1.txt είναι απλή: κάθε του γραμμή αποτελείται από μια συμβολοσειρά που περιέχει δύο στοιχεία, το κλειδί ενός μέλους λεξικού, το οποίο τερματίζεται με «:», κάποια κενά και μετά την τιμή του μέλους. Τα πιθανά κενά αφαιρούνται με την strip().

5.1.2. Άνοιγμα αρχείων κειμένου για εγγραφή

Όταν ανοίγεται ένα αρχείο σαν κείμενο για εγγραφή, υπάρχουν δύο εμφανείς επιλογές, όσον αφορά τα προϋπάρχοντα δεδομένα: να απορριφθούν και να γραφούν νέα, ή τα νέα δεδομένα να προστεθούν μετά από αυτά.

Στον κώδικα **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** εφαρμόζονται και οι δύο επιλογές.

Η εγγραφή δεδομένων σε ένα αρχείο που έχει ανοιχτεί για εγγραφή ή ενημέρωση γίνεται μέσω της μεθόδου write(). Η write() δέχεται μία παράμετρο τύπου συμβολοσειράς, οπότε ό,τι πρόκειται να γραφτεί στο αρχείο πρέπει να έχει μετατραπεί σε συμβολοσειρά, ενώ πρέπει να προστίθενται και οι χαρακτήρες νέας γραμμής ('\n') στα σημεία που χρειάζεται.

```
# Άνοιγμα και γράψιμο δύο γραμμών στο αρχείο
wr_file = open("write-file.txt", "w", encoding="utf-8")
i = 1
wr_file.write(f"Γραμμή {i} σε νέο αρχείο!\n")
```

```
i += 1
wr_file.write(f"Γραμμή αριθμός {i} στο αρχείο!\n")
i += 1
wr_file.close()

msg = """Άνοιγμα για ανάγνωση και ενημέρωση, διάβασμα μιας γραμμής, προσθήκη τρίτης:
"""
wr_file = open("write-file.txt", "r+", encoding="utf-8")
text = wr_file.readline()
print("1. " + msg , text)
wr_file.write(f"Άλλη μια γραμμή, αριθμός {i}.\n")
i += 1
wr_file.close()

msg = """Άνοιγμα για ανάγνωση, διάβασμα ολόκληρου του αρχείου (3 γραμμές):
"""
wr_file = open("write-file.txt", "r", encoding="utf-8")
text = wr_file.read()
print("2. " + msg, text)
wr_file.close()

msg = """Άνοιγμα για εγγραφή και ενημέρωση:
"""
wr_file = open("write-file.txt", "w+", encoding="utf-8")
text = wr_file.read() # δεν διαβάζεται τίποτα, το αρχείο ανοίχτηκε για εγγραφή
print("3. " + msg, text)
wr_file.write(f"Μόνο μία γραμμή, με αριθμό {i}.\n") # Τα προηγούμενα σβήνονται
wr_file.close()

msg = """Άνοιγμα για ανάγνωση, διάβασμα ολόκληρου του αρχείου (μία γραμμή):
"""
wr_file = open("write-file.txt", "r", encoding="utf-8")
text = wr_file.read()
print("4. " + msg, text)
wr_file.close()
```
Κ. 173- Άνοιγμα αρχείου για εγγραφή, ανάγνωση και ενημέρωση.

Η εκτέλεση του κώδικα παράγει την παρακάτω έξοδο:

```
1. Άνοιγμα για ανάγνωση και ενημέρωση, διάβασμα μιας γραμμής, προσθήκη τρίτης:
 Γραμμή 1 σε νέο αρχείο!


2. Άνοιγμα για ανάγνωση, διάβασμα ολόκληρου του αρχείου (3 γραμμές):
 Γραμμή 1 σε νέο αρχείο!
Γραμμή αριθμός 2 στο αρχείο!
Άλλη μια γραμμή, αριθμός 3.
```

```
3. Άνοιγμα για εγγραφή και ενημέρωση:

4. Άνοιγμα για ανάγνωση, διάβασμα ολόκληρου του αρχείου (μία γραμμή):
 Μόνο μία γραμμή, με αριθμό 4.
```
Ε. 31 - Έξοδος κώδικα εγγραφής, ανάγνωσης και ενημέρωσης αρχείου.

Οι πρώτες επτά γραμμές ανοίγουν ένα αρχείο για εγγραφή («w») και γράφουν σε αυτό δύο γραμμές. Αν το αρχείο υπήρχε ήδη, τα προηγούμενα περιεχόμενά του χάνονται.

Στο επόμενο τμήμα του κώδικα το ίδιο αρχείο ανοίγεται για ανάγνωση και ενημέρωση («r+»). Πρώτα διαβάζεται μία γραμμή μέσω της μεθόδου `readline()`, ώστε να υπάρχει ανάγνωση πριν την ενημέρωση του αρχείου και κατόπιν προστίθεται άλλη μία γραμμή στο αρχείο. Αυτό επιβεβαιώνεται και από το επόμενο τμήμα του κώδικα, ανοίγει το αρχείο για ανάγνωση και τυπώνει τις τρεις γραμμές που περιέχει. Αν δεν γίνει κάποια ανάγνωση, η εγγραφή σβήνει τα προηγούμενα περιεχόμενα.

Αμέσως παρακάτω στον κώδικα το αρχείο ανοίγεται για εγγραφή και ενημέρωση («w+»), οπότε η ανάγνωση που επιχειρείται επιστρέφει μια κενή συμβολοσειρά, ενώ η εγγραφή νέου περιεχομένου έχει σαν αποτέλεσμα την απώλεια των προηγούμενων περιεχομένων, όπως φαίνεται και από την εκτέλεση του τελευταίου τμήματος του κώδικα, που ανοίγει το αρχείο για ανάγνωση και τυπώνει τα περιεχόμενά του.

5.1.3. Διαγραφή αρχείου

Η διαγραφή ενός αρχείου γίνεται μέσω της βιβλιοθήκης `os`, που δίνει στα προγράμματα Python πρόσβαση σε διάφορες ενέργειες που γίνονται μέσω του λειτουργικού συστήματος. Η βιβλιοθήκη αυτή προσφέρει έναν ενιαίο τρόπο για την πραγματοποίηση διαφόρων ενεργειών, ο οποίος είναι ανεξάρτητος από το λειτουργικό σύστημα του μηχανήματος στο οποίο τρέχει ο κώδικας.

Μια από τις συναρτήσεις της βιβλιοθήκης `os` είναι και η `remove()`, η οποία διαγράφει το αρχείο το όνομα του οποίου της δίνεται σαν όρισμα.

Επειδή αν δεν υπάρχει το αρχείο θα προκληθεί εξαίρεση, ένας τρόπος να αποφευχθεί αυτό είναι πριν τη διαγραφή να γίνει έλεγχος ότι πράγματι υπάρχει το συγκεκριμένο αρχείο. Αυτό γίνεται με τη συνάρτηση `exists()`. Η συνάρτηση αυτή είναι μέρος του τμήματος κώδικα (module) `path` της βιβλιοθήκης `os`, το οποίο έχει να κάνει με το χειρισμό μονοπατιών του συστήματος αρχείων. Η `exists()` παίρνει σαν παράμετρο ένα όνομα μονοπάτι αρχείου και αν υπάρχει επιστρέφει `True`, αλλιώς επιστρέφει `False`.

5.2. Άνοιγμα αρχείου με την εντολή `with` – context managers

Η Python παρέχει και έναν άλλο τρόπο για άνοιγμα και επεξεργασία ενός αρχείου, την εντολή `with`. Με αυτήν φτιάχνεται ένα μπλοκ για την επεξεργασία του αρχείου. Επιπλέον, το αρχείο κλείνει αυτόματα με την έξοδο από το μπλοκ, οπότε δεν χρειάζεται να κληθεί η `contacts_file.close()`.

5.2.1. Άνοιγμα αρχείου για ανάγνωση με την εντολή `with`

Στον κώδικα Κ. 174 έχει ξαναγραφεί ο αρχικός κώδικας ανάγνωσης, αλλά με χρήση της `with`.

```
with open("D:/Python_A/contacts1.txt", "r", encoding="utf-8") as contacts_file:
    contacts = contacts_file.read()
    contacts_dict = {}
    for line in contacts.split("\n"):
        contact = line.split(":")
        contacts_dict[contact[0]] = contact[1]

print(contacts_dict) # {'Ελένη': ' 2101111', 'Χρήστος': ' 69702222', 'Γιώργος': ' 2103333',
'Αντωνία': ' 69704444'}
```

Κ. 174 - Άνοιγμα αρχείου για ανάγνωση με την with.

Η έξοδος του προγράμματος είναι η ίδια με αυτή της αρχικής εκδοχής, αλλά με τη χρήση της `with` ο κώδικας είναι πιο συνεκτικός και το αρχείο κλείνεται αυτόματα με το τέλος της ανάγνωσής του. Αν το αρχείο είχε ανοιχτεί για εγγραφή, τότε το αυτόματο κλείσιμό του γίνεται αφού γραφούν σε αυτό όλα τα προς εγγραφή δεδομένα, ώστε το κλείσιμο να γίνει με κάθε ασφάλεια. Γενικά το αρχείο κλείνεται κανονικά μετά το μπλοκ `with` ακόμα κι αν συμβεί κάποια εξαίρεση (exception).

5.2.2. Άνοιγμα αρχείου για εγγραφή με την εντολή `with`

Στον κώδικα, Κ. 175, χρησιμοποιείται η `with` για εγγραφή σε αρχείο. Αφού γίνει η εγγραφή, το αρχείο ανοίγεται για ανάγνωση και τυπώνονται τα περιεχόμενά του. Το αποτέλεσμα φαίνεται στο Ε. 32.

```
with open("D:/Python_A/write-file.txt", "w", encoding="utf-8") as wr_file:
    i = 1
    wr_file.write(f"Γραμμή {i} σε νέο αρχείο!\n")
    i += 1
    wr_file.write(f"Γραμμή αριθμός {i} στο αρχείο!\n")


with open("D:/Python_A/write-file.txt", "r", encoding="utf-8") as rd_file:
    for line in rd_file:
        print(line)
```

Κ. 175 - Άνοιγμα αρχείου για εγγραφή με χρήση της with.

```
Γραμμή 1 σε νέο αρχείο!

Γραμμή αριθμός 2 στο αρχείο!
```
Ε. 32 - Αποτέλεσμα της εκτέλεσης του κώδικα εγγραφής με την with.

5.2.3. Άνοιγμα αρχείου για προσθήκη δεδομένων με την εντολή `with`

Στο παράδειγμα Κ. 176 ανοίγεται το αρχείο που δημιουργήθηκε από τον προηγούμενο κώδικα και προστίθενται άλλες δύο γραμμές και στο Ε. 33 φαίνεται το αποτέλεσμα της εκτέλεσης.

```
with open("D:/Python_A/write-file.txt", "a", encoding="utf-8") as wr_file:
    i = 3
    wr_file.write(f"Αυτή είναι η γραμμή {i} στο αρχείο.\n")
    i += 1
    wr_file.write(f"Να και η γραμμή {i}.\n")


with open("D:/Python_A/write-file.txt", "r", encoding="utf-8") as rd_file:
    for line in rd_file:
        print(line)
```
Κ. 176 - Άνοιγμα αρχείου για προσθήκη δεδομένων με την with.

```
Γραμμή 1 σε νέο αρχείο!

Γραμμή αριθμός 2 στο αρχείο!

Αυτή είναι η γραμμή 3 στο αρχείο.

Να και η γραμμή 4.
```
Ε. 33 - Έξοδος του προγράμματος προσθήκης δεδομένων σε αρχείο.

Η εντολή `with` λειτουργεί με αντικείμενα τύπου *context manager* και τα αρχεία υποστηρίζουν αυτό το πρωτόκολλο. Το πρωτόκολλο context manager εκτός της ύλης του παρόντος συγγράμματος.

5.3. Άδειασμα περιεχομένων αρχείων

Στην παράγραφο 5.1.3 έχει ήδη παρουσιαστεί ένας τρόπος για να διαγραφεί ένα αρχείο με τη βοήθεια συναρτήσεων της βιβλιοθήκης `os`.

Υπάρχουν περιπτώσεις που χρειάζεται να διαγραφούν τα περιεχόμενα ενός αρχείου, αλλά το ίδιο το αρχείο να συνεχίσει να υπάρχει χωρίς περιεχόμενα. Αυτό μπορεί να γίνει εύκολα με μια μόνο γραμμή, ανοίγοντας το αρχείο για εγγραφή και κλείνοντάς το χωρίς να γραφούν δεδομένα:

```
open("D:/Python_A/write-file.txt", "w").close()
```

Αυτό γίνεται επειδή, όπως έχει αναφερθεί, το άνοιγμα ενός αρχείου για εγγραφή με την επιλογή «w» προκαλεί την απώλεια των δεδομένων που υπήρχαν στο αρχείο. Το άμεσο κλείσιμο επιτυγχάνει το αποτέλεσμα ενός κενού αρχείου.

5.4. Εργασία με αρχεία τύπου CSV

Τα αρχεία CSV είναι αρχεία κειμένου και χρησιμοποιούνται συχνά για μεταφορά, εξαγωγή ή αποθήκευση πληροφοριών που είναι σε μορφή πίνακα. Αυτό τα κάνει κατάλληλα για αποθήκευση δεδομένων από φύλλα εργασίας, πίνακες βάσεων δεδομένων κλπ. Αποτελούνται από γραμμές (κάθε γραμμή αποτελεί και μία εγγραφή) και στήλες (κάθε στήλη αποτελεί και ένα πεδίο). Τα αρχικά του ονόματός τους προέρχονται από τις λέξεις Comma Separated Values (τιμές χωρισμένες με κόμματα). Περιέχουν τιμές (πεδία) σε μορφή συμβολοσειράς, που χωρίζονται μεταξύ τους με ένα διαχωριστικό χαρακτήρα, συνήθως κόμμα. Η ονομασία είναι κάπως παραπλανητική, αφού ο διαχωριστικός χαρακτήρας εκτός από κόμμα, μπορεί να είναι για παράδειγμα ελληνικό ερωτηματικό, ο χαρακτήρας του κενού, ή ο χαρακτήρας tab.

Στο τέλος κάθε γραμμής, μετά την τελευταία τιμή δεν υπάρχει ο διαχωριστικός χαρακτήρας, μόνο ο χαρακτήρας (ή οι χαρακτήρες, ανάλογα με το λειτουργικό σύστημα) νέας γραμμής.

Υπάρχει επιλογή, ώστε η πρώτη γραμμή να μην περιέχει δεδομένα, αλλά τα ονόματα των στηλών (ή πεδίων). Μια άλλη επιλογή είναι τα πεδία που *αντιπροσωπεύουν* συμβολοσειρές να είναι κλεισμένα μέσα σε εισαγωγικά, π.χ. επειδή μπορεί να περιέχουν κενά ή τον διαχωριστικό χαρακτήρα, ή να είναι όλα τα πεδία κλεισμένα σε εισαγωγικά. (Όπως αναφέρθηκε, τα αρχεία CSV είναι αρχεία κειμένου, οπότε *όλα* τα πεδία τους αποτελούνται από συμβολοσειρές, αλλά κάποια από αυτά μπορεί να απεικονίζουν αριθμητικά ή άλλου τύπου δεδομένα).

Σε κάποιες περιπτώσεις μπορεί να απαιτείται αρκετή προσοχή και κάπως σύνθετος κώδικας για να διαβαστεί σωστά ένα αρχείο CSV, αφού μπορεί π.χ. μια τιμή (πεδίο) να περιέχει κενά ή και τον διαχωριστικό χαρακτήρα.

5.4.1. Δημιουργία αρχείου CSV

Έστω ότι υπάρχει μια λίστα με βιβλία, όπου για κάθε βιβλίο έχει τις εξής πληροφορίες: τίτλος, συγγραφέας (για λόγους ευκολίας είναι πάντα ένα πρόσωπο), αριθμός ISBN, έτος έκδοσης, εκδοτικός οίκος και αριθμός σελίδων. Για να σωθεί αυτή η λίστα σαν αρχείο τύπου CSV, μπορεί να χρησιμοποιηθεί κώδικας σαν τον Κ. 177, που παράγει την έξοδο Ε. 34, που είναι ένα σωστά δομημένο αρχείο τύπου CSV:

```python
books = [
    [
        "The Art of Computer Programming, VOLUME 1: Fundamental Algorithms",
        "Donald E. Knuth",
        "0201896834",
        1997,
        "Addison-Wesley",
        650
    ],
    [
        "Αλγόριθμοι, εισαγωγικά θέματα και παραδείγματα",
        "Θεόδωρος Σ. Παπαθεοδώρου",
        "9605300265",
        1999,
        "Εκδόσεις Πανεπιστημίου Πατρών",
        272
    ],
    [
        "The C Programming Language, 2nd Ed.",
        "Brian W. Kernighan & Denis M. Ritchie",
        "0131103628",
        1988,
        "Prentice Hall",
        272
    ],
    [
        "Digital Design, 2nd Ed.",
        "M. Morris Mano",
        "0132129949",
        1991,
        "Prentice Hall",
        516
    ],
]
# Εγγραφή των βιβλίων σαν αρχείο CSV με διαχωριστικό χαρακτήρα το ;
with open("D:/Python_A/compsci-books-file.csv", "w", encoding="utf-8") as book_file:
    book_file.write("Τίτλος;Συγγραφέας;ISBN;Έτος έκδοσης;Εκδότης;Αριθμός σελίδων\n")
    for book in books:
        line = ""
        for field in book:
            line += f"{field};"
        line = line[:-1] + "\n"
        book_file.write(line)
# Ανάγνωση του αρχείου για επιβεβαίωση σωστών περιεχομένων
with open("D:/Python_A/compsci-books-file.csv", "r", encoding="utf-8") as book_file:
    for book in book_file:
        print(book)
```

Κ. 177 - Πρόγραμμα για δημιουργία αρχείου CSV.

```
Τίτλος;Συγγραφέας;ISBN;Έτος έκδοσης;Εκδότης;Αριθμός σελίδων
The Art of Computer Programming, VOLUME 1: Fundamental Algorithms;Donald E.
Knuth;0201896834;1997;Addison-Wesley;650
Αλγόριθμοι, εισαγωγικά θέματα και παραδείγματα;Θεόδωρος Σ.
Παπαθεοδώρου;9605300265;1999;Εκδόσεις Πανεπιστημίου Πατρών;272
The C Programming Language, 2nd Ed.;Brian W. Kernighan & Denis M.
Ritchie;0131103628;1988;Prentice Hall;272
Digital Design, 2nd Ed.;M. Morris Mano;0132129949;1991;Prentice Hall;516
```

Ε. 34 - Έξοδος προγράμματος δημιουργίας αρχείου CSV.

5.4.2. Ανάγνωση αρχείου CSV

Για να διαβαστούν τα δεδομένα ενός αρχείου CSV μπορεί να χρησιμοποιηθεί η μέθοδος `split`, ώστε να διαχωριστούν τα πεδία κάθε γραμμής. Ο κώδικας Κ. 1 ανοίγει ένα αρχείο CSV για ανάγνωση και εισάγει όλα τα περιεχόμενά του στη λίστα `books`, παράγοντας την έξοδο Ε. 35.

```
with open("D:/Python_A/compsci-books-file.csv", "r", encoding="utf-8") as book_file:
    books = []
    for book in book_file:
        books.append(list(book[:-1].split(";")))
    print(books[1:]) # Να μην τυπωθεί η επικεφαλίδα με τα ονόματα των πεδίων
```

Κ. 1 - Ανάγνωση αρχείου CSV σε λίστα.

```
[['The Art of Computer Programming, VOLUME 1: Fundamental Algorithms', 'Donald E. Knuth', '0201896834', '1997', 'Addison-Wesley', '650'], ['Αλγόριθμοι, εισαγωγικά θέματα και παραδείγματα', 'Θεόδωρος Σ. Παπαθεοδώρου', '9605300265', '1999', 'Εκδόσεις Πανεπιστημίου Πατρών', '272'], ['The C Programming Language, 2nd Ed.', 'Brian W. Kernighan & Denis M. Ritchie', '0131103628', '1988', 'Prentice Hall', '272'], ['Digital Design, 2nd Ed.', 'M. Morris Mano', '0132129949', '1991', 'Prentice Hall', '516']]
```

Ε. 35 - Η λίστα με τα περιεχόμενα του αρχείου CSV.

Όταν ανοίγεται ένα αρχείο για ανάγνωση, φτιάχνεται και ένας δείκτης θέσης, που δείχνει το σημείο του αρχείου που έχει φτάσει η ανάγνωσή του. Μετά την ανάγνωση όλων των περιεχομένων του με την `with`, ο δείκτης αυτός δείχνει στο τέλος του αρχείου, οπότε αν χρησιμοποιηθεί ξανά η `with`, δεν υπάρχουν περισσότερα δεδομένα για να διαβαστούν.

Σε αυτό μπορεί να βοηθήσει η μέθοδος `seek()`, που μπορεί να μεταφέρει τον δείκτη στην αρχή του αρχείου.

Η χρήση της έχει κάποιους περιορισμούς. Δεν μπορεί να χρησιμοποιηθεί μέσα σε βρόχο for που χρησιμοποιείται για να διαβαστούν τα περιεχόμενα του αρχείου, επειδή ο βρόχος χρησιμοποιεί τον δείκτη θέσης. Επίσης δεν είναι ασφαλές να χρησιμοποιείται η seek για μετακίνηση σε τυχαίες θέσεις αρχείων UTF-8, επειδή οι χαρακτήρες τους δεν έχουν σταθερό μέγεθος σε bytes και μπορεί η

μετακίνηση να γίνει σε σημείο που δεν είναι το αρχικό byte ενός χαρακτήρα UTF-8, προκαλώντας λάθος ή εξαίρεση κατά την ανάγνωση. Χρειάζεται προσοχή, εμπειρία και γνώση της κωδικοποίησης UTF-8. Αντίθετα, είναι ασφαλής η χρήση της σε αρχεία που περιέχουν μόνο χαρακτήρες ASCII, επειδή έχουν όλοι μέγεθος ενός byte.

Επίσης είναι ασφαλής η μετακίνηση στην αρχή του αρχείου, όπως γίνεται στον κώδικα Κ. 178.

```
with open("D:/Python_A/compsci-books-file.csv", "r", encoding="utf-8") as book_file:
    books = []
    for book in book_file:
        books.append(list(book[:-1].split(";")))
    print(books)
    # Δεν τυπώνεται τίποτα επειδή ο δείκτης είναι στο τέλος του αρχείου
    print("Ανάγνωση στο τέλος αρχείου ")
    for book in book_file:
        print("Αυτό δεν τυπώνεται ποτέ, το for δεν εκτελείται ", book)
    # Μετακίνηση στην αρχή του αρχείου
    book_file.seek(0, 0)
    # Τώρα διαβάζονται τα περιεχόμενα του αρχείου
    print("Ξαναδιάβασμα από την αρχή του αρχείου:")
    for book in book_file:
        print(book)
```

Κ. 178 - Ανάγνωση αρχείου και επιστροφή στην αρχή για δεύτερη ανάγνωση.

Όπως φαίνεται στην Ε. 36, το αρχείο ξαναδιαβάζεται από την αρχή.

```
[['Τίτλος', 'Συγγραφέας', 'ISBN', 'Έτος έκδοσης', 'Εκδότης', 'Αριθμός σελίδων'], ['The Art
of Computer Programming, VOLUME 1: Fundamental Algorithms', 'Donald E. Knuth',
'0201896834', '1997', 'Addison-Wesley', '650'], ['Αλγόριθμοι, εισαγωγικά θέματα και
παραδείγματα', 'Θεόδωρος Σ. Παπαθεοδώρου', '9605300265', '1999', 'Εκδόσεις Πανεπιστημίου
Πατρών', '272'], ['The C Programming Language, 2nd Ed.', 'Brian W. Kernighan & Denis M.
Ritchie', '0131103628', '1988', 'Prentice Hall', '272'], ['Digital Design, 2nd Ed.', 'M.
Morris Mano', '0132129949', '1991', 'Prentice Hall', '516']]
Ανάγνωση στο τέλος αρχείου
Ξαναδιάβασμα από την αρχή του αρχείου:
Τίτλος;Συγγραφέας;ISBN;Έτος έκδοσης;Εκδότης;Αριθμός σελίδων

The Art of Computer Programming, VOLUME 1: Fundamental Algorithms;Donald E.
Knuth;0201896834;1997;Addison-Wesley;650

Αλγόριθμοι, εισαγωγικά θέματα και παραδείγματα;Θεόδωρος Σ.
Παπαθεοδώρου;9605300265;1999;Εκδόσεις Πανεπιστημίου Πατρών;272

The C Programming Language, 2nd Ed.;Brian W. Kernighan & Denis M.
Ritchie;0131103628;1988;Prentice Hall;272
```

```
Digital Design, 2nd Ed.;M. Morris Mano;0132129949;1991;Prentice Hall;516
```

Ε. 36 - Ανάγνωση αρχείου δεύτερη φορά από την αρχή.

Στον παραπάνω κώδικα αρχικά γίνεται ανάγνωση όλων των περιεχομένων του αρχείου, τα οποία μεταφέρονται σε μια λίστα το καθένα και τυπώνονται.

Κατόπιν γίνεται μια προσπάθεια να ξαναδιαβαστούν τα περιεχόμενα του αρχείου, αλλά δεν διαβάζεται τίποτα επειδή ο δείκτης είναι στο τέλος του αρχείου.

Με τη χρήση της `seek()` ο δείκτης μεταφέρεται ξανά στην αρχή του αρχείου και τα περιεχόμενά του μπορούν να ξαναδιαβαστούν.

Από τις δύο παραμέτρους που παίρνει η `seek()`, η δεύτερη ορίζει με βάση ποιο σημείο θα γίνει η μετακίνηση: 0 για την αρχή του αρχείου, 1 για την τρέχουσα θέση του δείκτη και 2 για το τέλος του αρχείου. Ειδικά για αρχεία κειμένου, που δεν έχουν ανοιχτεί σαν δυαδικά (binary), η μόνη επιτρεπτή τιμή είναι η 0, οπότε οποιαδήποτε μετακίνηση μπορεί να γίνει μόνο με αφετηρία την αρχή του αρχείου.

Η πρώτη παράμετρος ορίζει το μέγεθος της μετακίνησης σε bytes και για μετακίνηση με βάση την αρχή του αρχείου μπορεί να έχει τιμή θετική ή μηδέν.

Έτσι, η κλήση seek(0, 0) σημαίνει μετακίνηση στην αρχή του αρχείου, αφού ζητείται να μετακινηθεί ο δείκτης 0 bytes από την αρχή του αρχείου.

Εναλλακτικά, μπορούν να διαβαστούν όλα τα περιεχόμενα του αρχείου (για παράδειγμα σε μια λίστα, όπου κάθε μέλος της είναι μια γραμμή) και η οποιαδήποτε επεξεργασία να γίνει μέσω της λίστας, που μπορεί να διασχιστεί όσες φορές χρειάζεται.

Αυτό στον προηγούμενο κώδικα γίνεται στη γραμμή:

```
        books.append(list(book[:-1].split(";")))
```

Αυτό επιτρέπει να γίνουν ευκολότερα διάφορα είδη επεξεργασίας των περιεχομένων του, όπως π.χ. να βρεθεί το βιβλίο που εκδόθηκε πιο πρόσφατα ή αυτό με τις λιγότερες σελίδες, όπως γίνεται στον κώδικα **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**, που εμφανίζει τα αποτελέσματα των κριτηρίων στο Ε. 37.

```
with open("compsci-books-file.csv", "r", encoding="utf_8") as book_file:
    books = []
    for book in book_file:
        books.append(list(book[:-1].split(";")))
```

```
pub_year = 0; page_count = 65535
# Η πρώτη γραμμή είναι επικεφαλίδα με τα ονόματα των πεδίων
for book in books[1:]:
    year = int(book[3])
    pages = int(book[-1])
    if year > pub_year:
        pub_year = year
    if pages < page_count:
        page_count = pages
print(f"Πιο πρόσφατο έτος έκδοσης: {pub_year}.\n",
      f"Μικρότερος αριθμός σελίδων: {page_count}.")
```

Κ. 179- Ανάγνωση αρχείου CSV και επιλογή περιεχομένων με κριτήρια.

```
Πιο πρόσφατο έτος έκδοσης: 1999.
 Μικρότερος αριθμός σελίδων: 272.
```

Ε. 37 - Αποτελέσματα εφαρμογής των κριτηρίων.

Επειδή όπως αναφέρθηκε τα αρχεία CSV έχουν αρκετά μεγάλη ποικιλία δυνατών μορφών, υπάρχει η βιβλιοθήκη `csv`, που παρέχει πολλές ευκολίες στον χειρισμό αρχείων CSV.

Η βιβλιοθήκη περιέχει διάφορες κλάσεις που μπορεί να φανούν χρήσιμες στο χειρισμό αρχείων CSV. Κάποιες ενδεικτικές, με συνοπτική περιγραφή, είναι οι παρακάτω:

- `reader`: επιστρέφει ένα αντικείμενο αναγνώστη, που θα επεξεργαστεί τις γραμμές του αρχείου CSV. Κάθε γραμμή του αρχείου επιστρέφεται σαν μια λίστα από συμβολοσειρές.
- `writer`: επιστρέφει ένα αντικείμενο εγγραφέα, που μετατρέπει τα δεδομένα του χρήστη σε οριοθετημένες συμβολοσειρές που γράφονται στο αρχείο CSV.
- `DictReader`: δημιουργεί ένα αντικείμενο ανάγνωσης εισάγοντας τις τιμές των πεδίων κάθε γραμμής σε ένα λεξικό, χρησιμοποιώντας σαν ονόματα κλειδιών είτε τα περιεχόμενα της παραμέτρου `fieldnames`, που είναι μια *ακολουθία* (π.χ. λίστα ή πλειάδα), είτε τα στοιχεία της πρώτης γραμμής του αρχείου, αν δεν δίνεται η `fieldnames`.
- `DictWriter`: δημιουργεί ένα αντικείμενο εγγραφέα, αλλά αντιστοιχεί λεξικά σε γραμμές εξόδου. Εδώ η παράμετρος `fieldnames` είναι υποχρεωτική και υποδεικνύει τη σειρά με την οποία θα γραφούν οι τιμές του λεξικού στο αρχείο CSV.

Για σωστή λειτουργία της βιβλιοθήκης `csv`, όπως αναφέρεται και στην τεκμηρίωση της βιβλιοθήκης, το άνοιγμα των αρχείων, είτε για ανάγνωση, είτε για εγγραφή, πρέπει να γίνεται με την παράμετρο `newline=''`.

Στον Κ. 180 παρουσιάζεται ο προηγούμενος κώδικας τροποποιημένος, ώστε να χρησιμοποιεί τη csv.

```
import csv

"""
Η πρώτη γραμμή είναι επικεφαλίδα που περιέχει τα ονόματα των στηλών.
Χρησιμοποιείται για να διαβαστούν οι υπόλοιπες γραμμές σαν λεξικά,
με κλειδιά τα ονόματα των στηλών.
"""
filepath = "compsci-books-file.csv"
with open(filepath, "r", encoding="utf-8", newline='') as book_file:
    reader = csv.DictReader(book_file, delimiter=";")
    books = []
    for row in reader:
        books.append(row)
    #print(books)

    pub_year = 0; page_count = 65535
    for book in books:
        year = int(book["Έτος έκδοσης"])
        pages = int(book["Αριθμός σελίδων"])
        if year > pub_year:
            pub_year = year
        if pages < page_count:
            page_count = pages
    print(f"Πιο πρόσφατο έτος έκδοσης: {pub_year}.\n",
          f"Μικρότερος αριθμός σελίδων: {page_count}.")
```
Κ. 180 - Ο προηγούμενος κώδικας αλλαγμένος για να χρησιμοποιεί τη βιβλιοθήκη csv.

Λεπτομερής περιγραφή της βιβλιοθήκης μπορεί να βρεθεί στην επίσημη σελίδα τεκμηρίωσης της βιβλιοθήκης στον ιστότοπο της Python: https://docs.python.org/3/library/csv.html.

5.4.3. Άσκηση 1 – εργασία με αρχεία CSV

Γράψτε ένα πρόγραμμα που θα διαβάζει το αρχείο «compsci-books-file.csv» και θα κάνει τα εξής:

Α) Θα προσθέτει τις στήλες:

- Μια στήλη με όνομα «Τιμή»
- Μια στήλη με όνομα «Έκπτωση»
- Μια στήλη με όνομα «Τιμή έκπτωσης»
- Μια στήλη με όνομα «Βαθμολογία»

Β) Θα δώσει τιμές στα αντίστοιχα πεδία για κάθε βιβλίο. Βάλτε δικές σας τιμές και ποσοστά, υπολογίστε τις αντίστοιχες τιμές με έκπτωση και βάλτε δική σας βαθμολογία για κάθε βιβλίο (με τιμές 0 – 5). Σώστε το αρχείο με άλλο όνομα.

5.5. Εργασία με αρχεία JSON

Τα αρχεία μορφής JSON (προφέρεται «τζέισον», όπως το όνομα Jason) είναι αρχεία που χρησιμοποιούνται για αποθήκευση και ανταλλαγή δεδομένων. Τα δεδομένα είναι σε αναγνώσιμη μορφή και η μορφή τους μέσα στο αρχείο ακολουθεί το ομώνυμο πρότυπο. Το πρότυπο JSON έχει ευρύ πεδίο εφαρμογής και χρήσης και δεν εξαρτάται από κάποια συγκεκριμένη γλώσσα προγραμματισμού, αν και η καταγωγή του έλκει από τη γλώσσα JavaScript, όπως δείχνει και το όνομά του: JavaScript Object Notation. Τα αρχεία JSON έχουν την κατάληξη `.json` και τα περιεχόμενά τους είναι κωδικοποιημένα με το πρότυπο UTF-8.

Οι τύποι δεδομένων που έχει το JSON είναι:

- Αριθμοί, που μπορεί να είναι ακέραιοι, δεκαδικοί ή κινητής υποδιαστολής - δεν υπάρχει διαχωρισμός.
- Συμβολοσειρές. Είναι κωδικοποιημένες κατά Unicode, οροθετούνται από χαρακτήρες διπλών εισαγωγικών και υποστηρίζουν ακολουθίες χαρακτήρων διαφυγής με χρήση της ανάποδης καθέτου (backslash, «\»).
- Boolean, με δυνατές τιμές `true` και `false`.
- Διάνυσμα (array). Είναι μια διατεταγμένη λίστα από μηδέν ή περισσότερα στοιχεία. Κάθε στοιχείο μπορεί να είναι οποιουδήποτε τύπου. Τα στοιχεία χωρίζονται με κόμμα. Όλη η λίστα περιέχεται μέσα σε ζευγάρι αγκυλών (όπως και οι λίστες στην Python).
- Αντικείμενο. Είναι συλλογή από ζεύγη ονόματος-τιμής. Τα ονόματα (ονομάζονται και «κλειδιά») είναι συμβολοσειρές. Τα ζεύγη χωρίζονται με κόμμα. Το σύνολο του αντικειμένου περιέχεται μέσα σε ζευγάρι αγκίστρων. Είναι αντίστοιχο με το λεξικό της Python, αλλά σε αυτό το κλειδί κάθε μέλους του αντικειμένου πρέπει να είναι συμβολοσειρά.
- null. Μια κενή τιμή, χρησιμοποιεί τη λέξη `null`.

Ένα παράδειγμα μικρού και απλού αρχείου JSON παρουσιάζεται στον Κ. 181 .

```
{
  "first_name": "Ιωάννης",
  "last_name": "Παππάς",
  "birthday": {
```

```
    "year": 1977,
    "month": 5,
    "date": 1
},
  "address": {
    "street": "Rockwell",
    "number": 6502,
    "city": "Πάτρα",
    "postal_code": "265 00"
  },
  "phone_number": [
    {
      "type": "Οικία",
      "number": "2610 999999"
    },
    {
      "type": "Κινητό",
      "number": "6900 999999"
    }
  ],
  "car": {
    "make": "Ford",
    "model": "Focus"
  }
}
```
Κ. 181 -Παράδειγμα αρχείου JSON.

Ένα αρχείο JSON μπορεί εύκολα να γίνει περίπλοκο και μεγάλο σε μέγεθος, έτσι ο χειρισμός τους στην Python γίνεται με τη χρήση της βιβλιοθήκης `json`.

5.5.1. Μετατροπή από JSON σε Python

Για να μετατραπεί μια συμβολοσειρά JSON και να γίνει συμβατή με την Python, χρησιμοποιείται η μέθοδος `loads`. Αυτή διαβάζει τη συμβολοσειρά και επιστρέφει ένα λεξικό της Python. Ένα απλό παράδειγμα φαίνεται στον Κ. 182, όπου διαβάζει μια συμβολοσειρά JSON, τη μετατρέπει σε λεξικό της Python και τυπώνει μια φράση στο Ε. 38 με βάση τα δεδομένα που διαβάστηκαν.

```python
import json

# ένα αντικείμενο  JSON
dune_json = '{"title": "Dune", "author": "Frank Herbert", "year": 1965}'
# διάβασμα σε λεξικό Python
dune_dict = json.loads(dune_json)
print(dune_dict, "\n")
print(f"Ο {dune_dict['author']} εξέδωσε το {dune_dict['title']} το {dune_dict['year']}.")
```
Κ. 182 - Μετατροπή συμβολοσειράς JSON σε λεξικό Python.

```
{'title': 'Dune', 'author': 'Frank Herbert', 'year': 1965}


O Frank Herbert εξέδωσε το Dune το 1965.
```

Ε. 38 - Μετατροπή της JSON σε λεξικό και ενδεικτική χρήση.

5.5.2. Μετατροπή από Python σε JSON

Στον κώδικα του Κ. 183 γίνεται η αντίστροφη μετατροπή, από αντικείμενο Python σε μια συμβολοσειρά JSON γίνεται με τη μέθοδο `dumps`.

```
import json

# ένα αντικείμενο  JSON
dune_dict = {
    'title': 'Dune',
    'author': 'Frank Herbert',
    'year': 1965
    }
# διάβασμα σε λεξικό Python
dune_json = json.dumps(dune_dict)
print(dune_json) # {"title": "Dune", "author": "Frank Herbert", "year": 1965}
```

Κ. 183 - Μετατροπή από λεξικό Python σε συμβολοσειρά JSON.

Η μετατροπή επιστρέφει κάτι αντίστοιχο της αρχικής συμβολοσειράς JSON. Στον κώδικα εμφανίζεται σαν σχόλιο στην τελευταία γραμμή του κώδικα.

Η Python διαθέτει περισσότερους τύπους από το JSON, οπότε όταν γίνεται μετατροπή από την Python προς JSON:

- οι αριθμοί κινητής υποδιαστολής (float) και οι ακέραιοι (int) της Python μετατρέπονται στον τύπο number
- οι πλειάδες (tuples) και οι λίστες (lists) μετατρέπονται σε διανύσματα (arrays).
- τα λεξικά (dict) μετατρέπονται σε αντικείμενα (objects)
- οι συμβολοσειρές (str) στις αντίστοιχές τους (strings)
- οι τιμές True και False στις επίσης αντίστοιχές τους true και false
- η None μετατρέπεται σε null.

Τα σύνολα της Python δεν μπορούν να μετατραπούν άμεσα σε κάποιο τύπο JSON και η μέθοδος `json.dumps` εγείρει λάθος αν της δοθεί σαν όρισμα σύνολο. Για να μετατραπεί ένα σύνολο σε JSON πρέπει πρώτα να μετατραπεί, π.χ. σε λίστα ή πλειάδα.

Στον κώδικα Κ. 184 γίνονται όλες οι μετατροπές από Python σε JSON και τα αποτελέσματα εμφανίζονται στο Ε. 39, όπου κάτω από κάθε αντικείμενο της Python εμφανίζεται το αντίστοιχο αντικείμενο JSON στο οποίο μετατράπηκε.

```python
import json

python_types = [
    4095,
    6.28318,
    5.23E18,
    "William Gibson",
    ["Neuromancer", "Count Zero", "Mona Lisa Overdrive"],
    ("Kim Stanley Robinson", "Red Mars", "Greem Mars","Blue Mars"),
    {"author": "Peter Watts", "title": "Blindsight", 512: 2.71828},
    True,
    False,
    None
]

for item in python_types:
    json_str = json.dumps(item)
    print(f"Python object: {item}\nJSON string:{json_str}\n")
```

Κ. 184 - Μετατροπές τύπων από Python σε JSON.

```
Python object: 4095
JSON string:4095

Python object: 6.28318
JSON string:6.28318

Python object: 5.23e+18
JSON string:5.23e+18

Python object: William Gibson
JSON string:"William Gibson"

Python object: ['Neuromancer', 'Count Zero', 'Mona Lisa Overdrive']
JSON string:["Neuromancer", "Count Zero", "Mona Lisa Overdrive"]

Python object: ('Kim Stanley Robinson', 'Red Mars', 'Greem Mars', 'Blue Mars')
JSON string:["Kim Stanley Robinson", "Red Mars", "Greem Mars", "Blue Mars"]

Python object: {'author': 'Peter Watts', 'title': 'Blindsight', 512: 2.71828}
JSON string:{"author": "Peter Watts", "title": "Blindsight", "512": 2.71828}
```

```
Python object: True
JSON string:true

Python object: False
JSON string:false

Python object: None
JSON string:null
```
Ε. 39 - Αντικείμενα Python και η μετατροπή τους σε JSON.

Αξίζει να επισημανθεί η μετατροπή του κλειδιού 512 στο λεξικό Python του κώδικα από αριθμό (που επιτρέπεται στην Python) σε κλειδί-συμβολοσειρά στο JSON object (που μετατρέπεται το λεξικό), αφού στα JSON objects τα κλειδιά του αντικειμένου είναι πάντα συμβολοσειρές.

Επειδή η `dumps()` χωρίς τροποποίηση επιστρέφει τα αντικείμενα σε μια γραμμή, κάνοντας εξαιρετικά δύσκολη την ανάγνωση από άνθρωπο αν περιέχονται πολλά αντικείμενα, η `dumps()` έχει, μεταξύ άλλων, κάποιες παραμέτρους, ώστε να προσαρμόζει τη έξοδό της.

- Η παράμετρος `indent` καθορίζει το μέγεθος της εσοχής για κάθε επίπεδο. Αν έχει τιμή 0, τότε εισάγονται μόνο χαρακτήρες νέας γραμμής ('\n'), ενώ με μη μηδενικές τιμές κάθε επίπεδο αυξάνει την εσοχή με τον αντίστοιχο αριθμό κενών.
- Η παράμετρος `separators` μπορεί να χρησιμοποιηθεί για να αλλάξει τα διαχωριστικά μεταξύ αντικειμένων και μεταξύ κλειδιού και τιμής. Δέχεται σαν τιμή ένα ζευγάρι, όπου η πρώτη τιμή ορίζει το νέο διαχωριστικό μεταξύ αντικειμένων (προκαθορισμένο είναι το '`, `') και η δεύτερη το διαχωριστικό μεταξύ κλειδιού και τιμής στα λεξικά (προκαθορισμένη τιμή η '`: `').
- Η `sort_keys` αν είναι `True`, ταξινομεί την έξοδο των λεξικών με βάση τα κλειδιά.

Στο παράδειγμα Κ. 185 γίνεται χρήση τους, ώστε να είναι πιο ευανάγνωστο το αποτέλεσμα Ε. 40 - JSON σε ευανάγνωστη μορφή.

Η εσοχή ανά επίπεδο έχει αλλάξει, από τα τέσσερα κενά του `jobj`, στα δύο κενά που δόθηκαν με το όρισμα στην `indent`. Σαν διαχωριστικό αντικείμενων χρησιμοποιεί το ελληνικό ερωτηματικό, ενώ σαν διαχωριστικό μεταξύ κλειδιού και τιμής ένα βελάκι με κενό αριστερά και δεξιά του. Τέλος, ταξινομεί και τα λεξικά ως προς τα κλειδιά τους.

```python
import json

jobj = {
```

```
    "book": {
        "title":"Blindsight",
        "author": {
            "lastname": "Watts",
            "firstname": "Peter"
            }
    },
        "type": "paperback"
    }

pstr = json.dumps(jobj, indent=2, separators=(';', ' => '), sort_keys=True)
print(pstr)
```
Κ. 185 - Εκτύπωση JSON σε ευανάγνωστη μορφή.

```
{
  "book" => {
    "author" => {
      "firstname" => "Peter";
      "lastname" => "Watts"
    };
    "title" => "Blindsight"
  };
  "type" => "paperback"
}
```
Ε. 40 - JSON σε ευανάγνωστη μορφή.

5.6. Εργασία με αρχεία XML

Η XML (eXtensible Markup Language, επεκτάσιμη γλώσσα σήμανσης) είναι μια γλώσσα σήμανσης για *ιεραρχικά* δεδομένα. Όπως αναφέρει και το όνομά της, είναι επεκτάσιμη. Δημιουργήθηκε από τον W3C, τον οργανισμό που είναι υπεύθυνος για το World Wide Web για την αποθήκευση, μετάδοση και ανάπλαση δεδομένων με έμφαση στην απλότητα και την ευχρηστία. Αυτό είναι σε αντίθεση με την SGML (Standard Generalized Markup Language) από την οποία προέρχεται, που είναι κυριολεκτικά τεράστια και δύσχρηστη. Η σχεδίαση της XML είναι τέτοια, ώστε τα έγγραφα που δημιουργούνται να είναι αναγνώσιμα τόσο από τον άνθρωπο (μέχρι ενός σημείου πολυπλοκότητας), όσο και από μηχανή.

Η μορφή ενός αρχείου XML μοιάζει πολύ με τη μορφή ενός αρχείου HTML. Περιέχει ετικέτες (tags) που μπορεί να έχουν μέσα τους κάποια attributes (γνωρίσματα), ενώ οι ετικέτες περικλείουν κάποια δεδομένα.

Οι ετικέτες αντιπροσωπεύουν τη δομή των δεδομένων, τα γνωρίσματα (attributes) είναι μεταδεδομένα, ενώ τα ίδια τα δεδομένα περικλείονται από τις ετικέτες. Περιέχει ένα ριζικό στοιχείο (root element), μέσα στο οποίο περιέχονται όλα τα υπόλοιπα στοιχεία, σε ιεραρχική δομή μεταξύ τους. Αυτή η ιεραρχική δομή μπορεί να αναπαρασταθεί και με μια δομή δέντρου, με τα στοιχεία-παιδιά κάτω από τα στοιχεία-γονείς και πρόγονο όλων το ριζικό στοιχείο.

Η επεκτασιμότητα της XML προέρχεται από το γεγονός ότι οι χρήστες της ορίζουν τις δικές τους ετικέτες (tags), ενώ ο σκοπός είναι οι ετικέτες να περιγράφουν με το όνομά τους όσο είναι δυνατό τα δεδομένα που περικλείουν (αυτοπεριγραφικότητα, self-describing).

Επίσης είναι σχεδιασμένη ώστε να είναι ανεξάρτητη από οποιαδήποτε πλατφόρμα ή γλώσσα και γράφεται με κωδικοποίηση Unicode.

Πολλά πρότυπα ακολουθούν τη σύνταξη της XML, όπως τα XHTML, RSS, Atom, SVG, OpenDocument, αλλά και το πρωτόκολλο επικοινωνίας SOAP.

Στο παράδειγμα Κ. 2 εμφανίζεται ένα μικρό αρχείο XML.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<music>
  <album id="m1">
    <title>The Man-Machine</title>
    <artist>Kraftwerk</artist>
    <year>1978</year>
    <medium>CD</medium>
  </album>
  <album id="m4">
    <title>Blackstar</title>
    <artist>David Bowie</artist>
    <year>2016</year>
    <medium>mp3</medium>
  </album>
  <album id="m8">
    <title>Ο Σταυρός του Νότου</title>
    <artist>Θάνος Μικρούτσικος</artist>
    <year>1979</year>
    <medium>LP</medium>
  </album>
</music>
```

Κ. 2 - Παράδειγμα αρχείου XML.

Στην πρώτη γραμμή εμφανίζονται τρεις δηλώσεις:

- Ότι πρόκειται για αρχείο XML

- Ότι η έκδοση της XML που χρησιμοποιείται είναι η 1.0 (αυτή χρησιμοποιείται εκτός από ειδικές εξαιρέσεις)
- Ότι η κωδικοποίηση του αρχείου έγινε με το πρότυπο UTF 8 (η πιο συνηθισμένη με διαφορά)

Το ριζικό στοιχείο (root) είναι το στοιχείο `<music>`.

Κάτω από αυτό βρίσκονται τρία στοιχεία του είδους `<album>`, το καθένα από τα οποία αποτελείται από τα στοιχεία `<title>`, `<artist>`, `<year>` και `<medium>`. Το `<album>` έχει και ένα γνώρισμα (attribute), το `id`, που χρησιμοποιείται σαν μονοσήμαντο αναγνωριστικό για κάθε `<album>`.

Όταν ένα αρχείο XML είναι σωστά φτιαγμένο σύμφωνα με τους κανόνες της XML λέγεται ότι είναι well-formed (σωστά γραμμένο). Αυτό σημαίνει ότι:

- έχει ακριβώς ένα ριζικό στοιχείο
- όλες οι ετικέτες ανοίγουν και κλείνουν
- τα γνωρίσματα είναι μέσα σε εισαγωγικά
- οι ετικέτες δεν αλληλεπικαλύπτονται: ετικέτες που περιέχονται μέσα σε άλλες κλείνουν πριν κλείσουν οι ετικέτες που τις περιέχουν

Ο ορισμός της XML είναι αυστηρός ως προς τους κανόνες καλής διαμόρφωσης.

Στο παράδειγμα Κ. 186 το αρχείο δε είναι σωστά γραμμένο, επειδή λείπει η ετικέτα κλεισίματος `</artist>` και η ετικέτα κλεισίματος `</medium>` είναι μετά την ετικέτα κλεισίματος `</album>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<music>
  <album id="m1">
    <title>The Man-Machine</title>
    <artist>Kraftwerk
    <year>1978</year>
    <medium>CD
  </album>
    </medium>
</music>
```
Κ. 186 - Αρχείο XML που δεν είναι σωστά γραμμένο (well-formed).

Μαζί με την XML υπάρχουν και αρκετά *συστήματα σχημάτων* (schema systems), που λειτουργούν σαν κανόνες γραμματικής και συντακτικού για τις γλώσσες που βασίζονται στην XML. Ορίζουν μεταδεδομένα για την ερμηνεία και τον έλεγχο εγκυρότητας της XML.

Όταν ένα αρχείο XML είναι σωστά γραμμένο (well formed) και επιπλέον ακολουθεί και το σχήμα είναι valid (έγκυρο).

Υπάρχουν διάφορα πρότυπα για σχήματα, όπως το DTD (Document Type Definition), το XSD (XML Schema Definition) και άλλα.

Μαζί με την XML έχει αναπτυχθεί και η XSLT (eXtensible Stylesheet Language Transformations), μια γλώσσα που χρησιμοποιείται για τη μετατροπή είτε από μια XML σε άλλη XML, είτε από XML σε απλό κείμενο, HTML, JSON και άλλα πρότυπα.

5.6.1. Ανάγνωση ενός αρχείου XML

Για την ανάγνωση ενός αρχείου XML χρησιμοποιείται ένα μέρος του module `xml`, το `ElementTree`. Ο κώδικας Κ. 187 διαβάζει το (καλά γραμμένο) αρχείο XML που αναφέρθηκε παραπάνω και επεξεργάζεται κάποια στοιχεία του.

Αρχικά γίνεται διάσχιση του αρχείου XML (όπως έχει αναφερθεί έχει ιεραρχική δομή δέντρου), παίρνεται το ριζικό στοιχείο, μέσω του οποίου γίνεται η επεξεργασία των περιεχομένων του και στο Ε. 41 φαίνεται το αποτέλεσμα της επεξεργασίας.

```python
import xml.etree.ElementTree as ET
from pathlib import Path

# Τα αρχεία βρίσκονται στον ίδιο κατάλογο με το πρόγραμμα
workdir = Path(__file__).parent
xml_file = workdir / "albums.xml"

# Η χρονιά-βάση της αναζήτησης
test_year = 1978
# συμβολοσειρά που θα αναζητηθεί
test_str = "τ"

# XML parsing
tree = ET.parse(xml_file)
root = tree.getroot()
# Εκτύπωση των περιεχομένων
for album in root:
    print(album.attrib)

# Ανάγνωση επιλεγμένων δεδομένων
titles = [a.text for a in root.findall(".//title")]
artists = [a.text for a in root.findall(".//artist")]
print(f"Τίτλοι: {titles}\nΚαλλιτέχνες: {artists}")
# Αναζήτηση άλμπουμ που εκδόθηκαν μετά κάποια χρονιά
```

```python
found1 = [b.findtext("title") for b in root.findall("./album") if int(b.findtext("year")) >
test_year]
print(f"Μετά το {test_year}: {found1}")
# Αναζήτηση τίτλων άλμπουμ που περιέχουν μια συμβολοσειρά
found2 = [c.findtext("title") for c in root.findall("./album") if test_str in
c.findtext("artist")]
print(f"Περιέχει '{test_str}': {found2}")
```

Κ. 187 - Πρόγραμμα που διαβάζει ένα αρχείο XML.

```
{'id': 'm1'}
{'id': 'm2'}
{'id': 'm3'}
Τίτλοι: ['The Man-Machine', 'Blackstar', 'Ο Σταυρός του Νότου']
Καλλιτέχνες: ['Kraftwerk', 'David Bowie', 'Θάνος Μικρούτσικος']
Μετά το 1978: ['Blackstar', 'Ο Σταυρός του Νότου']
Περιέχει 'τ': ['Ο Σταυρός του Νότου']
```

Ε. 41 - Έξοδος της επεξεργασίας του αρχείου XML.

5.6.2. Προσθήκη σε αρχείο XML

Για να προστεθούν νέα στοιχεία σε ένα αρχείο XML, όπως φαίνεται στο Κ. 188, πρώτα ανοίγεται το αρχείο. Μετά αναλαμβάνει η συνάρτηση `add_item()` να προσθέσει ένα νέο στοιχείο στο αρχείο XML, με τη βοήθεια της `create_next_index()`, που διαβάζει το γνώρισμα (attribute) `id` όλων των στοιχείων που ήδη υπάρχουν και δημιουργεί το επόμενο.

Το αποτέλεσμα των προσθηκών φαίνεται στο Ε. 42.

```python
import xml.etree.ElementTree as ET
from pathlib import Path

# Τα αρχεία βρίσκονται στον ίδιο κατάλογο με το πρόγραμμα
workdir = Path(__file__).parent
xml_file = workdir / "albums.xml"

# Parse XML
tree = ET.parse(xml_file)
root = tree.getroot()

def create_next_index(root):
    """
    Διαβάζει τα attributes ταυτότητας (id) και δημιουργεί το επόμενο
    """
    idx = []
    for album in root:
        idx.append(int(album.get("id").replace("m", "")))
    return f"m{max(idx) + 1}"
```

```python
def add_item(root, title, artist, year, medium):
    """
    Προσθέτει ένα νέο άλμπουμ στα ήδη υπάρχοντα στο αρχείο XML.
    Αυξάνει αυτόματα το attribute id.
    Σώζει το ενημερωμένο αρχείο XML.
    """
    new_id = create_next_index(root)
    new_album = ET.Element("album", id=new_id)
    ET.SubElement(new_album, "title").text = title
    ET.SubElement(new_album, "artist").text = artist
    ET.SubElement(new_album, "year").text = year
    ET.SubElement(new_album, "medium").text = medium

    root.append(new_album)

    # Σώσιμο ενημερωμένου αρχείου
    upd_xml = workdir / "more_albums.xml"
    tree.write(upd_xml, encoding="utf-8", xml_declaration=True)
    print(f"Προστέθηκε το {title} με id {new_id} στο αρχείο {upd_xml.name}.")


albums_to_add = [
    ["Hounds of Love", "Kate Bush", 1985, "LP"],
    ["Φτηνή ποπ για την ελίτ", "Κόρε. Ύδρο.", 2006, "CD"],
    ["Από 'δω και πάνω", "Γιάννης Αγγελάκας και οι Επισκέπτες", 2005, "CD"],
]
# Βρόχος προσθήκης στοιχείων
for alb in albums_to_add:
    add_item(root, alb[0], alb[1], str(alb[2]), str(alb[3]))
```
Κ. 188 - Προσθήκη στοιχείων σε αρχείο XML.

```
Προστέθηκε το Hounds of Love με id m4 στο αρχείο more_albums.xml.
Προστέθηκε το Φτηνή ποπ για την ελίτ με id m5 στο αρχείο more_albums.xml.
Προστέθηκε το Από 'δω και πάνω με id m6 στο αρχείο more_albums.xml.
```
Ε. 42 - Αποτέλεσμα προσθήκης στοιχείων σε αρχείο XML.

5.6.3. Μετατροπή αρχείου XML σε αρχείο JSON

Για να μετατραπεί ένα αρχείο XML σε JSON πρέπει πρώτα να μετατραπεί σε λεξικό. Για να γίνει αυτό πρέπει να διασχιστεί όλο το δέντρο XML και να γίνει επεξεργασία σε κάθε στοιχείο του, ώστε να μεταφερθούν σωστά τόσο τα παιδιά του στοιχείου, όσο και τα πιθανά γνωρίσματά του (attributes).

Υπάρχει η βιβλιοθήκη `xmltodict`, η οποία όμως εγκαθίσταται μαζί με την Python μόνο σε κάποιες διανομές Linux. Στις περισσότερες, όπως και στα Windows, πρέπει να εγκατασταθεί χωριστά, π.χ. με χρήση της `pip`:

Αυτή αναλαμβάνει όλη την επεξεργασία του αρχείου XML και επιστρέφει ένα λεξικό. Η μετατροπή του λεξικού σε JSON έχει αναφερθεί στο 5.5.2. Μετατροπή από Python PS C:\Users\Ekdda> pip install xmltodict

σε JSON. Ο κώδικας Κ. 189 μετατρέπει ένα αρχείο XML αρχικά σε λεξικό της Python και μετά μετατρέπει το λεξικό σε αρχείο JSON. Τα δύο στάδια της μετατροπής φαίνονται στο Ε. 43, ενώ το αρχείο JSON που παράχθηκε εμφανίζεται στο Κ. 190.

```
from pathlib import Path
import json
import xmltodict

# Τα αρχεία βρίσκονται στον ίδιο κατάλογο με το πρόγραμμα
workdir = Path(__file__).parent
xml_file = workdir / "albums.xml"

with open(xml_file, "rb") as xfile:
    xml_dict = xmltodict.parse(xfile, encoding=None)
print(f"Το αρχείο XML σαν λεξικό:\n{xml_dict}")

print(f"Το αρχείο XML σαν JSON:\n{json.dumps(xml_dict)}")

jfile = workdir / "albums.json"
jfile.write_text(json.dumps(xml_dict, ensure_ascii=False, indent=4), encoding="utf-8")
print(f"Γράφηκε σαν JSON στο αρχείο {jfile.name}.")
```

Κ. 189 - Μετατροπή αρχείου XML σε αρχείο JSON.

```
Το αρχείο XML σαν λεξικό:
{'music': {'album': [{'@id': 'm1', 'title': 'The Man-Machine', 'artist': 'Kraftwerk', 'year': '1978', 'medium': 'CD'}, {'@id': 'm2', 'title': 'Blackstar', 'artist': 'David Bowie', 'year': '2016', 'medium': 'mp3'}, {'@id': 'm3', 'title': 'Ο Σταυρός του Νότου', 'artist': 'Θάνος Μικρούτσικος', 'year': '1979', 'medium': 'LP'}]}}
Το αρχείο XML σαν JSON:
{"music": {"album": [{"@id": "m1", "title": "The Man-Machine", "artist": "Kraftwerk", "year": "1978", "medium": "CD"}, {"@id": "m2", "title": "Blackstar", "artist": "David Bowie", "year": "2016", "medium": "mp3"}, {"@id": "m3", "title": "\u039f \u03a3\u03c4\u03b1\u03c5\u03c1\u03cc\u03c2 \u03c4\u03bf\u03c5 \u039d\u03cc\u03c4\u03bf\u03c5", "artist": "\u0398\u03ac\u03bd\u03bf\u03c2 \u039c\u03b9\u03ba\u03c1\u03bf\u03cd\u03c4\u03c3\u03b9\u03ba\u03bf\u03c2", "year": "1979", "medium": "LP"}]}}
```

```
Γράφηκε σαν JSON στο αρχείο albums.json.
```

Ε. 43 - Αποτέλεσμα της μετατροπής από XML σε JSON.


```json
{
    "music": {
        "album": [
            {
                "@id": "m1",
                "title": "The Man-Machine",
                "artist": "Kraftwerk",
                "year": "1978",
                "medium": "CD"
            },
            {
                "@id": "m2",
                "title": "Blackstar",
                "artist": "David Bowie",
                "year": "2016",
                "medium": "mp3"
            },
            {
                "@id": "m3",
                "title": "Ο Σταυρός του Νότου",
                "artist": "Θάνος Μικρούτσικος",
                "year": "1979",
                "medium": "LP"
            }
        ]
    }
}
```


Κ. 190 - Το αρχείο JSON που παράχθηκε.

Σημείωση: Τη στιγμή που γράφεται το παρόν, υπάρχει μια ειδοποίηση ασφαλείας (CVE-2025-9375) μέτριας επικινδυνότητας (6.9) για το συγκεκριμένο τμήμα (module). Ο συγγραφέας του κώδικα την αμφισβητεί, θεωρώντας ότι το πρόβλημα βρίσκεται αλλού. Η ειδοποίηση έχει να κάνει με πιθανή κρυφή εισαγωγή κώδικα (code injection). Αν η `xmltodict` δεν χρησιμοποιείται για επεξεργασία αρχείων XML με άγνωστη προέλευση ή/και περιεχόμενο, αλλά για αρχεία όπως αυτά των παραδειγμάτων, είναι ασφαλής. Σε οποιαδήποτε περίπτωση, αν υπάρχει η παραμικρή αμφιβολία ως προς την ασφάλεια της `xmltodict`, μπορούν να χρησιμοποιηθούν άλλες λύσεις για τη μετατροπή XML σε λεξικό.

5.6.4. Μετατροπή XML σε HTML και έλεγχος εγκυρότητας της XML

Όπως αναφέρθηκε παραπάνω, η XML μπορεί να μετατραπεί σε HTML. Αυτό γίνεται μέσω της XSLT, που φτιάχτηκε για αυτό το σκοπό, όπως έχει ήδη αναφερθεί. Η XSLT είναι μέρος της βιβλιοθήκης `lxml`, η οποία, αν δεν υπάρχει ήδη, μπορεί να εγκατασταθεί με την `pip`:

```
PS C:\Users\Ekdda> pip install lxml

Defaulting to user installation because normal site-packages is not writeable

Collecting lxml

  Downloading lxml-6.0.2-cp313-cp313-win_amd64.whl.metadata (3.7 kB)

Downloading lxml-6.0.2-cp313-cp313-win_amd64.whl (4.0 MB)

   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.0/4.0 MB 10.3 MB/s  0:00:00

Installing collected packages: lxml

Successfully installed lxml-6.0.2
```

Στην ίδια βιβλιοθήκη περιέχονται και μέθοδοι για τον έλεγχο εγκυρότητας ενός αρχείου XML ως προς ένα αρχείο σχήματος XSD. Ενδεικτική χρήση τους γίνεται στον κώδικα Κ. 191, που εκτελεί έλεγχο εγκυρότητας και εφόσον είναι εντάξει, μετατρέπει την XML σε HTML.

```python
from pathlib import Path
from lxml import etree

# Τα αρχεία βρίσκονται στον ίδιο κατάλογο με το πρόγραμμα
workdir = Path(__file__).parent
upd_xml = workdir / "more_albums.xml"

try:
    schema_file = workdir / "musiclibrary.xsd"
    schema_doc = etree.parse(str(schema_file))
    schema = etree.XMLSchema(schema_doc)

    doc = etree.parse(str(upd_xml))
    schema.assertValid(doc)
    print(f"Το αρχείο {upd_xml} είναι έγκυρο (valid) σύμφωνα με το αρχείο XSD
{schema_file}.")

    xslt_file = workdir / "musiclibrary.xslt"
    xslt_doc = etree.parse(str(xslt_file))
    xml2html = etree.XSLT(xslt_doc)

    html = xml2html(doc)
    html_file = workdir / "musiclibrary.html"
```

```
    html_file.write_text(str(html), encoding="utf-8")
    print(f"Το αρχείο XML {upd_xml} μετατράπηκε στο αρχείο HTML {html_file.name}.")

except Exception as e:
    print("Πρόβλημα με τα XSD/XSLT:", e)
```
Κ. 191 - Έλεγχος εγκυρότητας αρχείου XML και μετατροπή σε HTML.

Όλος ο κώδικας είναι μέσα σε ένα μπλοκ `try-except`. Έτσι, οποιοδήποτε πρόβλημα, τόσο στον έλεγχο εγκυρότητας όσο και κατά τη διαδικασία της μετατροπής σε HTML, σταματά τη διαδικασία δημιουργίας του αρχείου HTML.

Πρώτα γίνεται ένας έλεγχος εγκυρότητας του αρχείου XML με βάση ένα αρχείο σχήματος XSD. Διαβάζεται το αρχείο που περιέχει το σχήμα XSD (`musiclibrary.xsd`), γίνεται διάσχισή του και το σχήμα που περιέχει ανατίθεται στο αντικείμενο schema.

Στο αντικείμενο doc περνάνε (και αυτά με διάσχιση) τα περιεχόμενα του αρχείου XML. Αυτό το αντικείμενο ελέγχεταιι για την εγκυρότητά του από το αντικείμενο schema με τη μέθοδο `assertValid`.

Στο επόμενο στάδιο γίνεται ανάγνωση του αρχείου `musiclibrary.xslt`, με βάση το οποίο θα δημιουργηθεί το αρχείο HTML. Η διαδικασία είναι παρόμοια με αυτήν του ελέγχου εγκυρότητας: δημιουργείται ένα αντικείμενο `xml2html`, το οποίο παίρνει σαν όρισμα το αντικείμενο XML doc και επιστρέφει ένα αντικείμενο με τον κώδικα HTML.

Τελευταίο βήμα είναι η μετατροπή του αντικειμένου HTML σε συμβολοσειρά και η αποθήκευσή της σε ένα αρχείο HTML.

Το αρχείο XML είναι αυτό που δημιουργήθηκε στο τμήμα 5.6.2 Προσθήκη σε αρχείο XML .

Το αρχείο XSD είναι αυτό που εμφανίζεται στο Κ. 192, το αρχείο XSLT εμφανίζεται στο Κ. 193.

Στην Ε. 44 φαίνεται ότι το αρχείο XML είναι έγκυρο (valid) και ότι το αρχείο HTML δημιουργήθηκε επιτυχώς. Στο Ε. 45 φαίνεται η μορφή του αρχείου HTML όταν φορτωθεί σε φυλλομετρητή.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="music">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="album" type="albumType" maxOccurs="unbounded"/>
      </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>

  <xs:complexType name="albumType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="artist" type="xs:string" minOccurs="1"/>
      <xs:element name="year" type="xs:integer"/>
      <xs:element name="medium">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="single|EP|LP|CD single|CD|DVD|mp3|flac|wav"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required"/>
  </xs:complexType>
</xs:schema>
```

Κ. 192 - Το αρχείο XSD (musiclibrary.xsd).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes" />
  <xsl:template match="/">
    <html>
      <head>
        <meta charset="UTF-8"/>
        <title>Μουσική Βιβλιοθήκη</title>
      </head>
      <body>
        <h2>Μουσική Βιβλιοθήκη</h2>
        <table border="1" cellpadding="6">
          <tr>
            <th>Τίτλος</th>
            <th>Καλλιτέχνης</th>
            <th>Έτος</th>
            <th>Μέσον</th>
          </tr>
          <xsl:for-each select="music/album">
            <tr>
              <td><xsl:value-of select="title"/></td>
              <td><xsl:value-of select="artist"/></td>
              <td><xsl:value-of select="year"/></td>
              <td><xsl:value-of select="medium"/></td>
            </tr>
```

```
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Κ. 193 - Το αρχείο XSLT (musiclibrary.xslt).

```
Το αρχείο d:\Python_A\more_albums.xml είναι έγκυρο (valid) σύμφωνα με το αρχείο XSD
d:\Python_A\musiclibrary.xsd.
Το αρχείο XML d:\Python_A\more_albums.xml μετατράπηκε στο αρχείο HTML musiclibrary.html.
```

Ε. 44 - Εκτέλεση του προγράμματος: έγκυρο αρχείο XML και επιτυχής μετατροπή σε HTML.

Μουσική Βιβλιοθήκη

Τίτλος	Καλλιτέχνης	Έτος	Μέσον
The Man-Machine	Kraftwerk	1978	CD
Blackstar	David Bowie	2016	mp3
Ο Σταυρός του Νότου	Θάνος Μικρούτσικος	1979	LP
Hounds of Love	Kate Bush	1985	LP
Φτηνή ποπ για την ελίτ	Κόρε. Ύδρο.	2006	CD
Από 'δω και πάνω	Γιάννης Αγγελάκας και οι Επισκέπτες	2005	CD

Ε. 45 - Εμφάνιση του αρχείου HTML σε φυλλομετρητή.

5.7. Εργασία με αρχεία excel

Το excel είναι ένα λογισμικό που χρησιμοποιείται ευρύτατα, από ιδιώτες έως επιχειρήσεις, οργανισμούς, ακόμα και για επιστημονικούς σκοπούς. Βασικοί λόγοι είναι οι πολλές δυνατότητες που έχει, η μεγάλη βιβλιοθήκη συναρτήσεων για πολλά πεδία εφαρμογής, η ευρεία διάδοσή του και η σχετική ευκολία στη χρήση του.

Όπως έχει αναφερθεί νωρίτερα, ένας τρόπος να γίνει επεξεργασία δεδομένων από αρχείο excel είναι να μετατραπεί το αρχείο excel σε αρχείο CSV και να γίνει έτσι η επεξεργασία τους. Αυτός ο τρόπος, αν και απλός, έχει μειονεκτήματα.

Για την επεξεργασία των δεδομένων απ' ευθείας από το αρχείο excel χωρίς μετατροπές μπορεί να γίνει χρήση του εργαλείου pandas. Είναι ένα πολύ δυνατό εργαλείο για ανάλυση και επεξεργασία

δεδομένων και μπορεί να επεξεργαστεί δεδομένα σε πολλές μορφές (CSV, JSON, XML, LaTeX, κλπ.), ανάμεσά τους και αρχεία excel.

Επειδή η χρήση του pandas απαιτεί κάποια μελέτη, ακριβώς λόγω των δυνατοτήτων του, για απλούστερες εργασίες μπορεί να χρησιμοποιηθεί η βιβλιοθήκη openpyxl. Με αυτήν μπορεί να γίνει επεξεργασία αρχείων .xlsx (ανάγνωση, τροποποίηση και εγγραφή).

Η βιβλιοθήκη δεν εγκαθίσταται με την εγκατάσταση της Python, χρειάζεται να γίνει εγκατάσταση με το pip:

```
PS C:\Users\Ekdda> pip install openpyxl

Defaulting to user installation because normal site-packages is not writeable

Collecting openpyxl

  Downloading openpyxl-3.1.5-py2.py3-none-any.whl.metadata (2.5 kB)

Collecting et-xmlfile (from openpyxl)

  Downloading et_xmlfile-2.0.0-py3-none-any.whl.metadata (2.7 kB)

Downloading openpyxl-3.1.5-py2.py3-none-any.whl (250 kB)

Downloading et_xmlfile-2.0.0-py3-none-any.whl (18 kB)

Installing collected packages: et-xmlfile, openpyxl

Successfully installed et-xmlfile-2.0.0 openpyxl-3.1.5
```

Για τη λειτουργία της βιβλιοθήκης δεν είναι απαραίτητη η ύπαρξη του excel (σαν εφαρμογή) στον υπολογιστή. Άλλωστε, η βιβλιοθήκη χρησιμοποιείται σε διάφορα λειτουργικά συστήματα στα οποία μπορεί να μην υποστηρίζεται το excel.

Στον κώδικα Κ. 194 ανοίγεται ένα αρχείο excel, στο οποίο γίνεται διαφόρων ειδών επεξεργασία (ανάγνωση, προσθήκη δεδομένων σε κελιά, αλλαγή περιεχομένων σε κελιά, εγγραφή του επεξεργασμένου αρχείου – τα σχόλια πριν από κάθε τμήμα κώδικα αναφέρουν τη λειτουργία του).

Το άνοιγμα του αρχείου γίνεται σαν workbook (η ονομασία του excel για αρχεία με πολλά φύλλα εργασίας). Από αυτό επιλέγεται το φύλλο εργασίας που θα γίνει η επεξεργασία («compsci-books-file»).

Αρχικά διαβάζεται και τυπώνεται η πρώτη γραμμή, που περιέχει τα ονόματα των στηλών. Γίνεται χρήση μιας περιφραστικής συλλογής (comprehension), όπου η συνθήκη

```
if col.value is not None else ""
```

χρησιμοποιείται ώστε, αν δεν έχει δοθεί όνομα σε μια στήλη, να δίνεται σαν τιμή μια κενή συμβολοσειρά. Η έκφραση `sheet[1]` επιστρέφει αντικείμενο τύπου πλειάδας (tuple).

Παρακάτω διαβάζονται κάποια μεμονωμένα κελιά και τυπώνεται το περιεχόμενό τους.

Κατόπιν προστίθεται στο τέλος άλλη μια γραμμή με τα δεδομένα ενός ακόμη βιβλίου.

Τέλος, αλλάζει η τιμή ενός κελιού. Εδώ γίνονται επιπλέον έλεγχοι, ώστε αν το κελί περιέχει ήδη μια τιμή, είτε σαν δεκαδικό αριθμό, είτε σαν δεκαδικό αριθμό με τη μορφή συμβολοσειράς, να αφαιρείται από την τιμή ένα ποσό. Στις περιπτώσεις όπου το κελί δεν περιέχει τιμή ή το περιεχόμενο δεν είναι δεκαδικός αριθμός, γράφεται στο κελί μια αρχική τιμή.

Έχοντας ολοκληρωθεί η επεξεργασία, το αρχείο σώζεται, αντικαθιστώντας το αρχικό. Στην Ε. 46 εμφανίζεται η έξοδος που παράγεται από την εκτέλεση του κώδικα.

```
from openpyxl import load_workbook
from pathlib import Path

# Τα αρχεία βρίσκονται στον ίδιο κατάλογο με το πρόγραμμα
workdir = Path(__file__).parent
excel_file = workdir / "compsci-books-file.xlsx"

# Άνοιγμα αρχείου excel
workbook = load_workbook(excel_file)
# επιλογή φύλλου
sheet = workbook["compsci-books-file"]

# Ανάγνωση κεφαλίδας
print("Ονόματα στηλών:\n", ", ".join(str(col.value) if col.value is not None else "" for
col in sheet[1]))

# Ανάγνωση μεμονωμένων κελιών
title = sheet["A3"].value
author = sheet["B3"].value
price = sheet["G3"].value
print(f"Τίτλος: {title}, συγγραφέας: {author}, τιμή: {price}.")

# Προσθήκη γραμμής
sheet["A6"] = "Compilers - Principles, Techniques, and Tools"
sheet["B6"] = "Alfred Aho, Ravi Sethi & Jeffrey Ullman"
sheet["C6"] = "0201101947"
sheet["D6"] = 1997
sheet["E6"] = "Addison-Wesley"
```

```python
sheet["F6"] = "796"
sheet["G6"] = 55.75

# Αλλαγή τιμής κελιού (που περιέχει την τιμή του βιβλίου)
if isinstance(price, (int, float)):
    sheet["G3"] = price - 3.30
elif isinstance(price, str):
    price_parts = price.strip().split(',')
    if len(price_parts) == 2 and price_parts[0].isdigit() and price_parts[1].isdigit():
        sheet["G3"] = float(price.strip()) - 3.30
    else: # δεν είναι δεκαδικός, γίνεται ανάθεση τιμής
        sheet["G3"] = 31.60
else:
    # αν δεν έχει τιμή ή η τιμή δεν είναι αριθμός, γίνεται ανάθεση τιμής
    sheet["G3"] = 31.60

print(f"Νέα τιμή βιβλίου (κελί G3): {sheet["G3"].value}")

# Αποθήκευση των αλλαγών
workbook.save(excel_file)
print(f"Οι αλλαγές αποθηκεύτηκαν στο '{excel_file}'.")
```

Κ. 194 - Επεξεργασία αρχείου excel με χρήση της βιβλιοθήκης openpyxl.

```
Ονόματα στηλών:
 Τίτλος, Συγγραφέας, ISBN, Έτος έκδοσης, Εκδότης, Αριθμός σελίδων, Τιμή
Τίτλος: Αλγόριθμοι, εισαγωγικά θέματα και παραδείγματα, συγγραφέας: Θεόδωρος Σ.
Παπαθεοδώρου, τιμή: 31.00.
Νέα τιμή βιβλίου (κελί G3): 31.6
Οι αλλαγές αποθηκεύτηκαν στο 'd:\Python_A\compsci-books-file.xlsx'.
```

Ε. 46 - Αποτέλεσμα της εκτέλεσης του προγράμματος επεξεργασίας του αρχείου excel.

Ένας πολύ χρήσιμος σύνδεσμος με την τεκμηρίωση και παραδείγματα χρήσης για την openpyxl είναι ο ακόλουθος: https://openpyxl.readthedocs.io/en/stable/

5.7.1. Άσκηση 2 – εργασία με αρχεία excel

Γράψτε ένα πρόγραμμα που θα διαβάζει το αρχείο που δημιουργήσατε στην Άσκηση 1 – εργασία με αρχεία CSV και σώστε το σαν αρχείο excel, αλλάζοντας την κατάληξη από .csv σε .xlsx.

5.8. Σειριοποίηση και αποσειριοποίηση δεδομένων με το pickle

Σειριοποίηση (serialization) ονομάζεται η μετατροπή δεδομένων από την εσωτερική αναπαράστασή τους στον υπολογιστή (π.χ. σε ένα πρόγραμμα Python) σε μια μορφή που επιτρέπει είτε τη

μετάδοσή τους (π.χ. μέσω του Internet) σε άλλο υπολογιστή, είτε την αποθήκευσή τους σε ένα μέσο, όπως ο σκληρός δίσκος.

Ένας τρόπος που έχει η Python για να αποθηκεύει αρχεία, είναι το module `pickle` (κυριολεκτικά σημαίνει «τουρσί» ή «κάνω τουρσί», άρα συντηρώ με μια έννοια). Το `pickle` παίρνει μια ιεραρχία αντικειμένων που του δίνεται και τη σειριοποιεί σε δυαδική μορφή. Το αποτέλεσμα δεν είναι αναγνώσιμο από άνθρωπο. Επειδή κατά τη σειριοποίηση δεν μετατρέπει τύπους δεδομένων (π.χ. δεν μετατρέπει τα αντικείμενα σε συμβολοσειρές), όταν γίνεται η *αποσειριοποίηση* (deserialization) τα αντικείμενα έρχονται στην αρχική τους μορφή χωρίς να χρειάζεται η οποιαδήποτε μετατροπή.

5.8.1. Δημιουργία αρχείου με το pickle

Ο κώδικας στο Κ. 195 γράφει κάποια αντικείμενα Python σε ένα αρχείο με χρήση του `pickle`. Γράφονται διαδοχικά μια συμβολοσειρά, ένας πραγματικός αριθμός, μια λίστα και ένα λεξικό.

```
import pickle
from pathlib import Path
from math import sqrt

# Τα αρχεία βρίσκονται στον ίδιο κατάλογο με το πρόγραμμα
workdir = Path(__file__).parent
pickle_file = workdir / "pickle-file01.pickle"

with open(pickle_file, "wb") as pfile:
    pickle.dump("Spam, spam, wonderful spam!", pfile)
    pickle.dump(sqrt(2), pfile)
    pickle.dump([i**sqrt(i) for i in range(11)], pfile)
    pickle.dump({str(i): i**sqrt(i) for i in range(11)}, pfile)
```

Κ. 195 - Δημιουργία ενός αρχείου pickle.

5.8.2. Άνοιγμα αρχείου pickle για ανάγνωση και προσθήκη

Σαν επόμενο βήμα, ο κώδικας στο Κ. 196 ανοίγει το αρχείο που δημιουργήθηκε με το `pickle`, προσθέτει δύο αντικείμενα στο τέλος του αρχείου χρησιμοποιώντας τη συνάρτηση `append_obj_to_pickle_file()` και κατόπιν το ανοίγει ξανά, διαβάζει όλα τα περιεχόμενά του με τη συνάρτηση `read_pickle_file()`, που τα μεταφέρει όλα σε μια λίστα, και τέλος τα τυπώνει στο Ε. 47.

```
import pickle
import random
from pathlib import Path

# Τα αρχεία βρίσκονται στον ίδιο κατάλογο με το πρόγραμμα
```

```python
workdir = Path(__file__).parent
pickle_file = workdir / "pickle-file01.pickle"

def read_pickle_file(pf):
    """
    Διαβάζει όλα τα αντικείμενα από το αρχείο που παίρνει σαν παράμετρο και
    τα επιστρέφει σαν στοιχεία μιας λίστας.
    """
    pickle_objs = []
    with open(pickle_file, "rb") as pf:
        while True:
            try:
                obj = pickle.load(pf)
                pickle_objs.append(obj)
            except EOFError:
                break
    return pickle_objs

def append_obj_to_pickle_file(obj, pfile):
    """
    Προσθέτει το αντικείμενο της πρώτης παραμέτρου στο τέλος του αρχείου
    της δεύτερης παραμέτρου.
    """
    with open(pfile, "ab") as pf:
        pickle.dump(obj, pf)
    print(f"Το {obj} προστέθηκε στο {pfile}.")

# Προσθήκη δύο αντικειμένων

new_obj = f"Ένας τυχαίος αριθμός: {random.random()}."
append_obj_to_pickle_file(new_obj, pickle_file)
obj2 = {"circle": {"centre": {"x": 7, "y": -6}, "radius": 11}}
append_obj_to_pickle_file(obj2, pickle_file)

# Ανάγνωση του ενημερωμένου αρχείου
print("\nΤο αρχείο περιέχει τα παρακάτω:")
for o in read_pickle_file(pickle_file):
    print(o)
```

Κ. 196 - Άνοιγμα αρχείου pickle, προσθήκη και ανάγνωση ξανά..

```
Το Ένας τυχαίος αριθμός: 0.9345560222295298. προστέθηκε στο d:\Python_A\pickle-file01.pickle.
Το {'circle': {'centre': {'x': 7, 'y': -6}, 'radius': 11}} προστέθηκε στο d:\Python_A\pickle-file01.pickle.
```

```
Το αρχείο περιέχει τα παρακάτω:
Spam, spam, wonderful spam!
1.4142135623730951
[1.0, 1.0, 2.665144142690225, 6.704991853825879, 16.0, 36.554802677473226,
80.55147663887831, 172.1548440751979, 358.36386782970465, 729.0, 1453.0403018990435]
{'0': 1.0, '1': 1.0, '2': 2.665144142690225, '3': 6.704991853825879, '4': 16.0, '5':
36.554802677473226, '6': 80.55147663887831, '7': 172.1548440751979, '8':
358.36386782970465, '9': 729.0, '10': 1453.0403018990435}
Ένας τυχαίος αριθμός: 0.9345560222295298.
{'circle': {'centre': {'x': 7, 'y': -6}, 'radius': 11}}
```

Ε. 47 - Αποτελέσματα ενημέρωσης του αρχείου pickle.

Το `pickle`, παρ' όλη την προφανή χρησιμότητά του (μπορεί να αποθηκεύει οτιδήποτε είδους δεδομένα και αντικείμενα Python και να τα ανακτά χωρίς την ανάγκη μετατροπών και επεξεργασίας), έχει και μειονεκτήματα:

- Δεν είναι κατάλληλο για την αποθήκευση μεγάλου όγκου δεδομένων, γίνεται πολύ αργό.
- Είναι φτιαγμένο μόνο για την Python και δεν μπορεί να χρησιμοποιηθεί για μεταφορά ή ανταλλαγή δεδομένων σε εφαρμογές γραμμένες σε άλλη γλώσσα.
- οι νεότερες εκδόσεις toy `pickle` μπορεί να είναι ασύμβατες με παλιότερες εκδόσεις της Python., που έχουν και παλαιότερη έκδοση του `pickle`.
- Όπως γράφει και στην επίσημη σελίδα του `pickle` στο python.org, δεν πρέπει να χρησιμοποιείται για ανάγνωση και αποσειριοποίηση δεδομένων που προέρχονται από τρίτη πηγή ή το Διαδίκτυο. Είναι ανασφαλές και μπορεί να προκληθεί η εκτέλεση (κακόβουλου) κώδικα που είναι κρυμμένος μέσα σε ένα αρχείο `pickle`.

Πρέπει, λοιπόν, να χρησιμοποιείται μόνο για επεξεργασία μικρών σχετικά αρχείων που έχουν δημιουργηθεί τοπικά ή που είναι γνωστή και αξιόπιστη η πηγή τους.

5.8.3. Shelve: λεξικά στο ράφι

Το `shelve` είναι ένα module της Python που, όπως και το `pickle`, χρησιμοποιείται για τη μόνιμη αποθήκευση αντικειμένων. Τα αντικείμενα σώζονται με τη μορφή λεξικών, όπου το κλειδί πρέπει να είναι συμβολοσειρά, αλλά η τιμή μπορεί να είναι ουσιαστικά οποιοδήποτε αντικείμενο της Python. Ακριβέστερα, επειδή χρησιμοποιείται το `pickle` για να σειριοποιήσει τα αντικείμενα, οτιδήποτε μπορεί να σειριοποιηθεί από το `pickle`, μπορεί να είναι η τιμή ενός μέλους του λεξικού: στιγμιότυπα κλάσεων, αναδρομικοί τύποι δεδομένων, αντικείμενα που περιέχουν πολλά κοινά υποαντικείμενα, κλπ..

Τα περιεχόμενα σώζονται σε αρχείο που λέγεται shelf (= «ράφι»· το ρήμα shelve σημαίνει «βάζω στο ράφι».). Εκεί μπορούν να προσπελαστούν και να ανακτηθούν, να ενημερωθούν, να αφαιρεθούν και να ληφθεί μια λίστα με τα κλειδιά του ραφιού.

Για να ανοίξει ένα αρχείο shelf, χρησιμοποιείται η μέθοδος `open()`. Αυτή, εκτός από το όνομα του αρχείου, παίρνει και μια προαιρετική παράμετρο, την `flag`, που ορίζει με ποιο τρόπο θα ανοίξει το αρχείο. Η προεπιλεγμένη τιμή της `flag` είναι «`c`», που σημαίνει ότι το αρχείο ανοίγεται για ανάγνωση και εγγραφή και αν δεν υπάρχει, δημιουργείται.

Όταν γίνεται κάποια αλλαγή στα περιεχόμενα, καλό είναι να καλείται η μέθοδος `sync()`, ώστε να συγχρονίζονται τα περιεχόμενα του αρχείου shelf που είναι στον δίσκο με τα ενημερωμένα περιεχόμενα που βρίσκονται στην κύρια μνήμη του υπολογιστή. Αυτό γίνεται και με το κλείσιμο του αρχείου (π.χ. με τη μέθοδο `close()`), αλλά καλό είναι να γίνεται όταν ενημερώνονται στοιχεία, ώστε να μη χαθούν οι αλλαγές σε περίπτωση εξαίρεσης ή λάθους.

Στο πρόγραμμα Κ. 197 δημιουργείται ένα αρχείο shelf και σε αυτό προστίθενται στοιχεία, γίνεται ανάκλησή τους, ενημέρωση τιμών και αφαίρεση. Η παραγόμενη έξοδος εμφανίζεται στο Ε. 48Ε. 48.

```
import shelve
from pathlib import Path
import random

# Τα αρχεία βρίσκονται στον ίδιο κατάλογο με το πρόγραμμα
workdir = Path(__file__).parent
shelf_file = workdir / "shelf-file01.shelf"

with shelve.open("example1.shelf") as shelfile:
    # μια λίστα με συμβολοσειρές και ακέραιους
    lista = ["a", "b", "c", "d", 5, 6, 7, 8]
    shelfile["lista"] = lista
    # μεταφορά των δεδομένων στο δίσκο
    shelfile.sync()
    # ανάκληση από το αρχείο
    print(f"Αρχικές τιμές: {shelfile["lista"]}")

    # ενημέρωση τιμών
    shelfile["lista"] = ["a1", "b2", "b3", "b4", 55, 66, 77, 88]
    print(f"Ανανεωμένες τιμές: {shelfile['lista']}")
    # προσθήκη στοιχείων
    shelfile["τυχαίος1"] = random.random()
    shelfile["circle1"] = {"centre": {"x": -1.414, "y": 6.28}, "radius": 2.718}
    shelfile["quote1"] = "We are the knights who say Ni!"
    # μεταφορά των δεδομένων στο δίσκο
```

```
    shelfile.sync()

    # ανάκτηση των κλειδιών
    print(f"Τα κλειδιά πριν την αφαίρεση: {list(shelfile.keys())}")
    # αφαίρεση στοιχείου
    del shelfile["τυχαίος1"]
    print(f"Τα κλειδιά μετά την αφαίρεση: {list(shelfile.keys())}")
    # μεταφορά των δεδομένων στο δίσκο
    shelfile.sync()
```
Κ. 197 - Πρόγραμμα δημιουργίας αρχείου shelf, προσθήκης στοιχείων, ανάκλησης στοιχείων, ενημέρωσης στοιχείων και αφαίρεσης στοιχείων.

```
Αρχικές τιμές: ['a', 'b', 'c', 'd', 5, 6, 7, 8]
Ανανεωμένες τιμές: ['a1', 'b2', 'b3', 'b4', 55, 66, 77, 88]
Τα κλειδιά πριν την αφαίρεση: ['circle1', 'lista', 'quote1', 'τυχαίος1']
Τα κλειδιά μετά την αφαίρεση: ['circle1', 'lista', 'quote1']
```
Ε. 48 - Η έξοδος από την εκτέλεση του προγράμματος επεξεργασίας αρχείου shelf.

5.9. Εργασία με καταλόγους αρχείων: οι βιβλιοθήκες os και shutil

Πολλές φορές μια εφαρμογή χρειάζεται να αλληλοεπιδράσει με το λειτουργικό σύστημα για να εκτελέσει διάφορες λειτουργίες στο σύστημα αρχείων, όπως να βρει σε ποιο σημείο του συστήματος βρίσκεται, να βρει κάποιο αρχείο, να δημιουργήσει ή να αφαιρέσει ένα φάκελο.

Για τέτοιου είδους λειτουργίες μπορούν να χρησιμοποιηθούν δύο modules της Python, το `os` και το `shutils`.

Το `os` παρέχει μεθόδους για λειτουργίες «χαμηλού επιπέδου» του λειτουργικού συστήματος. Οι λειτουργίες αυτές έχουν πολύ ευρύ φάσμα και ένα μέρος τους είναι και αυτές που σχετίζονται με το χειρισμό αρχείων, φακέλων, τη διάσχιση (υπο-)δέντρων του συστήματος αρχείων, κλπ.

Το module `shutil` παρέχει μεθόδους για «υψηλού επιπέδου» λειτουργίες σε αρχεία, φακέλους και συλλογές αρχείων. Υποστηρίζονται αντιγραφές, μετακινήσεις και διαγραφές αρχείων, καθώς και αντιγραφές και διαγραφές ολόκληρων (υπο-)δέντρων του συστήματος αρχείων.

Όπως είναι φανερό, αρκετές από αυτές τις λειτουργίες χρειάζονται προσοχή. Για παράδειγμα, οι διαγραφές αρχείων και υποδέντρων είναι μη αναστρέψιμες, οπότε χρειάζεται ιδιαίτερη προσοχή στα ορίσματα που τους δίνονται.

Ένα μεγάλο μέρος των μεθόδων είναι ανεξάρτητες από το λειτουργικό σύστημα στο οποίο εκτελείται το πρόγραμμα, είτε αυτό είναι Linux/Unix, είτε MacOS, είτε Windows, αλλά υπάρχουν και κάποιες που έχουν να κάνουν μόνο με κάποιο συγκεκριμένο λειτουργικό σύστημα.

Όπως θα φανεί και στα παραδείγματα, για να μπορεί να γίνει σωστά η επεξεργασία των συμβολοσειρών από τις διάφορες μεθόδους, πρέπει οι συμβολοσειρές να είναι raw (= «ακατέργαστες», «χωρίς να υποστούν επεξεργασία» · έχουν το πρόθεμα «r» αμέσως πριν τα εισαγωγικά που ανοίγουν τη συμβολοσειρά) ώστε η Python να τις μεταφέρει στο λειτουργικό σύστημα ακριβώς όπως είναι, αλλιώς προκαλείται εξαίρεση.

5.9.1. Διαβάζοντας τα περιεχόμενα ενός φακέλου

Στο πρόγραμμα Κ. 198 περιέχεται μια συνάρτηση που παίρνει σαν παράμετρο το όνομα ενός φακέλου, ελέγχει ότι υπάρχει και ότι είναι πράγματι φάκελος και τέλος επιστρέφει τα περιεχόμενα του φακέλου χωρισμένα σε δύο λίστες, μια για τους φακέλους και μια για τα αρχεία.

```
from pathlib import Path
import os

def dir_full_list(path):
    """
    Διαβάζει τα περιεχόμενα του καταλόγου που δίνεται στην παράμετρο path
    και τα επιστρέφει σε δύο λίστες, μια με όλους τους φακέλους και μια
    με όλα τα αρχεία.
    Πρώτα ελέγχει ότι το path υπάρχει και ότι είναι φάκελος.
    """
    filelist = []
    dirlist = []

    """
    αντί του if/else μπορεί να χρησιμοποιηθεί το:
    try:
        dir_contents = os.listdir(path)
    except FileNotFoundError:
        print(f"Το μονοπάτι {path} δεν υπάρχει.")
        return dirlist, filelist
    """
    if os.path.exists(path) and os.path.isdir(path):
        dir_contents = os.listdir(path)
    else:
        print(f"Ο φάκελος {path} δεν υπάρχει.")
        return dirlist, filelist

    for entry in dir_contents:
```

```
        entrypath = os.path.join(path, entry)
        if os.path.isfile(entrypath):
            filelist.append(entrypath)
        else:
            dirlist.append(entrypath)

    return dirlist, filelist

# Δοκιμαστική κλήση
# Μετακίνηση στον αρχικό φάκελο
os.chdir(r"C:\XXX\YYY\ZZZ\Python_A")
# κλήση με τον επιθυμητό φάκελο
dirs, files = dir_full_list(r"os-examples")

if len(dirs) > 0:
    print("Κατάλογος φακέλων:")
    for item in dirs:
        print(item)

if len(files) > 0:
    print("\nΚατάλογος αρχείων:")
    for item in files:
        print(item)
```
Κ. 198 - Πρόγραμμα που διαβάζει τα περιεχόμενα ενός φακέλου και τα εμφανίζει.

Όπως αναφέρεται και στο σχόλιο μέσα στον κώδικα, κανονικά η ανάγνωση των περιεχομένων του φακέλου θα γινόταν μέσα σε ένα `try-except`, αλλά προτιμήθηκε το `if-else`, ώστε να χρησιμοποιηθούν κάποιες μέθοδοι του `os`.

Η εκτέλεση του κώδικα παράγει την έξοδο Ε. 49.

```
Κατάλογος φακέλων:
os-examples\albums
os-examples\books
os-examples\pickle-shelve

Κατάλογος αρχείων:
os-examples\contacts1.txt
os-examples\people.txt
os-examples\person.json
```
Ε. 49 - Οι κατάλογοι με τους φακέλους και τα αρχεία που περιέχονται στο φάκελο που διαβάζει το πρόγραμμα.

5.9.2. Δημιουργία και διαγραφή φακέλων και αρχείων

Το module os έχει τη μέθοδο mkdir, που δημιουργεί ένα φάκελο. Αν ο φάκελος υπάρχει ήδη ή αν κάποιος από τους γονικούς φακέλους στο μονοπάτι δεν υπάρχει, τότε προκαλούνται αντίστοιχες εξαιρέσεις/λάθη.

Η μέθοδος makedirs είναι πιο ευέλικτη. Αν κάποιοι ενδιάμεσοι φάκελοι δεν υπάρχουν, τότε τους δημιουργεί, ενώ αν στην παράμετρο exist_ok δοθεί η τιμή True, τότε δεν προκαλείται λάθος αν ο φάκελος που ζητείται να δημιουργηθεί υπάρχει ήδη.

Στον κώδικα Κ. 199 που ακολουθεί δημιουργείται ένας αρχικός φάκελος και από κάτω του φτιάχνονται μια σειρά από υποφακέλους, μερικοί και σε δεύτερο επίπεδο, και σε αυτούς φτιάχνονται μερικά (κενά) αρχεία μέσω της μεθόδου os.open() (αντίστοιχη της open, που θα μπορούσε επίσης να χρησιμοποιηθεί), ώστε να μην είναι άδειοι οι φάκελοι.

Στη συνέχεια κάποια επιλεγμένα αρχεία διαγράφονται, όπως και ένας φάκελος μαζί με όλα τα περιεχόμενά του. Στο Ε. 50 εμφανίζεται η αναφορά που δίνει το πρόγραμμα για κάποιες από τις λειτουργίες που εκτέλεσε.

Για τα αρχεία χρησιμοποιείται η μέθοδος os.remove(), που διαγράφει μόνο αρχεία, ενώ για τους φακέλους χρησιμοποιείται η os.deltree(), που διαγράφει φακέλους μαζί με όλα τα περιεχόμενά τους. Για την ακρίβεια, όπως φανερώνει και το όνομα της μεθόδου, διαγράφεται όλο το υποδέντρο κάτω από τον φάκελο.

Η μέθοδος deltree() πρέπει να χρησιμοποιείται με προσοχή και να της δίνονται τα σωστά ορίσματα, επειδή σε περίπτωση διαγραφής λάθος φακέλων και/ή αρχείων δεν υπάρχει τρόπος να ανακτηθούν.

```
from pathlib import Path
import os
import shutil

# Δημιουργία του αρχικού φακέλου με συνάρτηση για πιθανές πολλαπλές κλήσεις
def create_initial_dir(dname):
    try:
        os.mkdir(dname)
    except FileExistsError:
        print(f"Το {dname} υπάρχει ήδη!")

# Διαγραφή αρχείου ή φακέλου
def del_object(oname):
    """
```

```python
    Διαγράφει τον φάκελο ή το αρχείο με το όνομα που δίνεται.
    """
    if os.path.isdir(oname):
        shutil.rmtree(oname)
        print(f"Ο φάκελος {oname} διαγράφηκε.")
    elif os.path.isfile(oname):
        os.remove(oname)
        print(f"Το αρχείο {oname} διαγράφηκε.")
    else:
        print(f"Δεν υπάρχει φάκελος ή αρχείο με το όνομα {oname}.")

# Μετακίνηση στον αρχικό φάκελο
os.chdir(r"C:\XXX\YYY\ZZZ\Python_A\os-examples")
# Δημιουργία φακέλου με την mkdir
direx1 = r"dir-example"
create_initial_dir(direx1)

# Δημιουργία μερικών φακέλων και αρχείων
os.chdir(direx1)
new_dirs = [
    r"dirA",
    r"dirB",
    r"dirC",
    r"dirD",
    r"dirE",
]
c = r"A"
for dirname in new_dirs:
    if dirname in (r"dirB", r"dirD"):
        for subdir in [r"sd1", r"sd2", r"sd3"]:
            dirpath = os.path.join(dirname, subdir)
            os.makedirs(dirpath, exist_ok=True)
            for _ in range(3):
                fname = os.path.join(dirpath, r"file_" + c + r".txt")
                f = os.open(fname, os.O_CREAT)
                os.close(f)
                c = chr(ord(c) + 1)
    else:
        dirpath = dirname
        os.makedirs(dirpath, exist_ok=True)
        for _ in range(2):
            fname = os.path.join(dirpath, r"file_" + c + r".txt")
            f = os.open(fname, os.O_CREAT)
            os.close(f)
            c = chr(ord(c) + 1)

# Διαγραφή φακέλων και αρχείων
```

```
for d_suffix in (r"C", r"E"):
    for suffix in range(ord("M"), ord("X")):
        d = r"dir" + d_suffix
        f = r"file_" + chr(f_suffix) + r".txt"
        fpath = os.path.join(d, f)
        if os.path.exists(fpath):
            del_object(fpath)
del_object(r"dirB")
```

Κ. 199 - Πρόγραμμα που δημιουργεί ένα φάκελο με υποφακέλους και αρχεία και εκτελεί διάφορες ενέργειες σε αυτά.

```
Το αρχείο dirC\file_M.txt διαγράφηκε.
Το αρχείο dirE\file_W.txt διαγράφηκε.
Ο φάκελος dirB διαγράφηκε.
```

Ε. 50 - Έξοδος με την αναφορά του προγράμματος.

5.9.3. Αντιγραφή και μετακίνηση αρχείων και φακέλων

Η αντιγραφή αρχείων από ένα φάκελο σε έναν άλλο γίνεται με χρήση μεθόδων του module `shutil`.

Η `shutil.copy()` αντιγράφει το αρχείο που περνιέται στην πρώτη παράμετρο, στον προορισμό, που περνιέται στην δεύτερη παράμετρο. Αν ο προορισμός είναι φάκελος, τότε το αρχικό όνομα του αρχείου παραμένει. Αν είναι αρχείο, τότε το αντίγραφο παίρνει αυτό το όνομα. Σε κάθε περίπτωση, αν το αρχείο προορισμού υπάρχει, τα περιεχόμενά του αντικαθίσταται από τα περιεχόμενα του αρχείου που αντιγράφεται.

Η `shutil.copy2()` κάνει ό,τι κάνει και η `copy()`, αλλά στο αρχείο-προορισμό μεταφέρει και τα *μεταδεδομένα* του αρχικού αρχείου, όπως ο χρόνος δημιουργίας, χρόνος τελευταίας αλλαγής, δικαιώματα, κλπ. Για τα μεταδεδομένα υπάρχει και η `shutil.copystat()`, η οποία αντιγράφει *μόνο* τα μεταδεδομένα του πηγαίου αρχείου στο αρχείο προορισμού, χωρίς να πειράζει τα περιεχόμενα του αρχείου προορισμού. Τα δύο αρχεία καταλήγουν να έχουν το καθένα τα δικά του περιεχόμενα, αλλά τα ίδια χαρακτηριστικά, όπως χρόνο δημιουργίας, ιδιοκτήτη, δικαιώματα εγγραφής/ανάγνωσης, κλπ..

Για την αντιγραφή ενός φακέλου (και των περιεχομένων του, φυσικά) χρησιμοποιείται η μέθοδος `shutil.copytree()`. Με αυτήν ο προς αντιγραφή φάκελος αντιγράφεται κάτω από το φάκελο προορισμού. Αν ο φάκελος υπάρχει (σαν όνομα) στον φάκελο προορισμού, τότε εγείρεται λάθος, εκτός αν η παράμετρος `dirs_exist_ok` έχει την τιμή `True`. Αν στο φάκελο προορισμού υπάρχει αρχείο με το όνομα του προς αντιγραφή φακέλου, τότε προκαλείται λάθος.

Η μετακίνηση αρχείου ή φακέλου σε άλλη θέση γίνεται με την `shutil.move()`. Αν ο προορισμός είναι φάκελος, το προς μετακίνηση αντικείμενο (φάκελος ή αρχείο) τοποθετείται κάτω από αυτόν.

Ο κώδικας στο Κ. 200 εκτελεί αρκετές από αυτές τις ενέργειες στους φακέλους και τα αρχεία που δημιουργήθηκαν από το πρόγραμμα Κ. 199.

```
import os
import shutil

# Μετακίνηση στον αρχικό φάκελο
os.chdir(r"C:\XXX\YYY\ZZZ\Python_A\os-examples")

# Αντιγραφή αρχείου σε άλλο φάκελο
shutil.copy(r"people.txt", r"dir-example/dirA")
# Αντιγραφή αρχείου στη θέση αρχείου που ήδη υπάρχει, μεταφέροντας
# και τα μεταδεδομένα
shutil.copy2(r"people.txt", r"dir-example/dirE/file_X.txt")

# Αντιγραφή φακέλου με τα περιεχόμενά του
shutil.copytree(r"dir-example/dirD", r"dir-example/dirX")
# Η ίδια αντιγραφή, αλλά με το dirs_exist_ok να έχει την τιμή True,
# ώστε αν οι φάκελοι προορισμού υπάρχουν ήδη να μην προκαλείται
# λάθος.
shutil.copytree(r"dir-example/dirD", r"dir-example/dirX", dirs_exist_ok=True)

# Μετακίνηση αρχείου
shutil.move(r"dir-example/dirD/sd2/file_Q.txt", r"dir-example/dirC")
# Μετακίνηση αρχείου σε ήδη υπάρχον αρχείο
shutil.move(r"dir-example/dirD/sd2/file_R.txt", r"zzz.txt")

# Μετακίνηση φακέλου
shutil.move(r"dir-example/dirD/sd1", r"dir-example/dirA")
# Μετακίνηση φακέλου σε ήδη υπάρχοντα φάκελο
shutil.move(r"dir-example/dirD/sd3/file_V.txt", r"zzz.txt")
```

Κ. 200 - Εκτέλεση διαφόρων ενεργειών σε φάκελο και στα περιεχόμενά του.

Στην περίπτωση μετακίνησης αρχείου, αν το αρχείο-προορισμός υπήρχε πριν τη μετακίνηση, αντικαθίσταται. Αν πρόκειται να μετακινηθεί φάκελος και στον προορισμό υπάρχει αρχείο με το ίδιο όνομα, το αν θα αντικατασταθεί ή όχι εξαρτάται από κάποια στοιχεία υλοποίησης της μεθόδου `os.rename()`.

Να σημειωθεί εδώ ότι η μετακίνηση αρχείου ή φακέλου, όταν δεν μεσολαβεί αλλαγή διαμέρισης δίσκου (partition) ή μέσου αποθήκευσης (π.χ. διαφορετικός δίσκος), έχει ελάχιστο κόστος σε επεξεργασία και χρόνο εκτέλεσης. Αλλάζουν μόνο κάποιες εγγραφές στο σύστημα αρχείων ώστε να

γίνει η μετακίνηση, χωρίς φυσική μεταφορά των φακέλων ή/και των αρχείων. Αυτό δεν ισχύει αν ο προορισμός είναι σε άλλο φυσικό ή λογικό μέσο, όπως δεν ισχύει και για την αντιγραφή.

5.9.4. Διασχίζοντας ένα μονοπάτι του συστήματος αρχείων με την os.walk

Η `os.walk()` είναι μια μέθοδος που παίρνει σαν πρώτη παράμετρο ένα μονοπάτι και το διατρέχει. Για κάθε φάκελο επιστρέφει μια τριάδα (πλειάδα τριών στοιχείων), που αποτελείται από το τρέχον μονοπάτι, μια λίστα με τους φακέλους που περιέχονται στον τρέχοντα φάκελο και μια λίστα με όλα τα αρχεία που περιέχονται στον τρέχοντα φάκελο. Αυτή η διαδικασία γίνεται διαδοχικά (αναδρομικά) για κάθε έναν από τους υποφακέλους του αρχικού φακέλου και κατόπιν για τους υποφακέλους αυτών, μέχρι να φτάσει σε όλα τα στοιχεία του υποδέντρου.

Η `os.walk()` επιστρέφει ένα αντικείμενο που μπορεί να χρησιμοποιηθεί σε ένα βρόχο `for` για να γίνει επεξεργασία των στοιχείων. Για τα στοιχεία του κάθε φορά τρέχοντος φακέλου (αρχεία και φακέλους) επιστρέφει μόνο το όνομά τους μέσα στο φάκελο.

Για να φτιαχτεί το πλήρες μονοπάτι του κάθε στοιχείου πρέπει να γίνει χρήση της `os.path.join()`, περνώντας της σαν ορίσματα το πρώτο μέλος της τριάδας που επιστρέφει η `os.walk()`, που είναι το μονοπάτι του τρέχοντα φακέλου, και το όνομα του αρχείου ή του υποφακέλου.

Η μέθοδος αυτή είναι πολύ χρήσιμη για περιπτώσεις όπου χρειάζεται να γίνει κάποιο είδος επεξεργασίας σε όλα τα στοιχεία ενός τμήματος (ή όλου) του συστήματος αρχείων, για παράδειγμα αναζήτηση μιας συμβολοσειράς στα περιεχόμενα όλων των αρχείων κειμένου (.txt).

Η `os.walk()` έχει και ένα προαιρετικό όρισμα, το `topdown`, που καθορίζει τη φορά με την οποία γίνεται η διάσχιση του δέντρου. Αν είναι `True` (είναι η προκαθορισμένη τιμή), η διάσχιση γίνεται με τη φορά που περιγράφηκε παραπάνω: ξεκινάει από τον τρέχοντα φάκελο και πηγαίνει προς τα πιο απομακρυσμένα στοιχεία. Αν είναι `False`, ξεκινάει από τα πιο απομακρυσμένα σημεία στο βάθος του δέντρου και επιστρέφει προς τον τρέχοντα φάκελο.

Στον κώδικα Κ. 201 γίνεται αναζήτηση για αρχεία και φακέλους που περιέχουν μια συγκεκριμένη συμβολοσειρά στο όνομά τους (`"file_S"` και `"sd3"`, αντίστοιχα). Τα αποτελέσματα της αναζήτησης εμφανίζονται στο Ε. 51.

```
import os


def find_files(dirpath, pattern):
```

```python
    results = []
    for path, dirs, files in os.walk(dirpath):
        for f in files:
            if pattern in f:
                results.append(os.path.join(path, f))
    return results

def find_dirs(dirpath, pattern):

    results = []
    for path, dirs, files in os.walk(dirpath):
        for d in dirs:
            if pattern in d:
                results.append(os.path.join(path, d))
    return results

# Μετακίνηση στον αρχικό φάκελο
root_dir = r"C:\XXX\YYY\ZZZ\Python_A\os-examples"
os.chdir(root_dir)

# Αναζήτηση αρχείου που περιέχει συγκεκριμένη συμβολοσειρά στο όνομά του
filelist = find_files(root_dir, "file_S")
# Αναζήτηση φακέλου που περιέχει συγκεκριμένη συμβολοσειρά στο όνομά του
dirlist = find_dirs(root_dir, "sd3")
print("Βρέθηκαν τα αρχεία:")
for f in filelist:
    print(f)
print("Βρέθηκαν οι φάκελοι:")
for d in dirlist:
    print(d)
```
Κ. 201 - Διάσχιση δέντρου και αναζήτηση αρχείων με συγκεκριμένο χαρακτηριστικό στο όνομά τους.

```
Βρέθηκαν τα αρχεία:
C:\XXX\YYY\ZZZ\Python_A\os-examples\dir-example\dirD\sd2\file_S.txt
C:\XXX\YYY\ZZZ\Python_A\os-examples\dir-example\dirX\sd2\file_S.txt
Βρέθηκαν οι φάκελοι:
C:\XXX\YYY\ZZZ\Python_A\os-examples\dir-example\dirD\sd3
C:\XXX\YYY\ZZZ\Python_A\os-examples\dir-example\dirX\sd3
```
Ε. 51 - Αποτελέσματα της αναζήτησης με τη διάσχιση του μονοπατιού.

5.9.5. Άσκηση 3 – επεξεργασία φακέλων και αρχείων

Γράψτε ένα πρόγραμμα που θα δημιουργεί ένα φάκελο με το όνομα backups σε σημείο που θα ορίζεται με χρήση της `input` και κάτω από αυτό θα φτιάξει άλλους τρεις φακέλους, με ονόματα python, excel και text.

Κατόπιν θα διατρέχει τον τρέχοντα κατάλογο και τους υποκαταλόγους του και όλα τα αρχεία με κατάληξη .py και .ipynb θα τα αντιγράφει στον κατάλογο backups/python, τα αρχεία .xlsx και .csv στον κατάλογο backups/excel και τα αρχεία με κατάληξη .txt κάτω από τον κατάλογο backups/text. Η αντιγραφή να γίνεται με τρόπο που να μεταφέρει και τα μεταδεδομένα των αρχείων.

5.10. Ερωτήσεις κλειστού τύπου

1. Τα δεδομένα σε ένα αρχείο XML είναι σε μορφή:
   A. Αναδρομική
   B. Επίπεδη
   C. Ιεραρχική
   D. Δυαδική
2. Ποια από τις παρακάτω είναι η σωστή κλήση της open για άνοιγμα αρχείου κειμένου με όνομα `textfile.txt` για ανάγνωση και ενημέρωση;
   A. `f = open("textfile.txt", "r+", encoding="unicode")`
   B. `f = open("textfile.txt", "r+", encoding="utf-8")`
   C. `f = open("textfile.txt", "w-")`
   D. `f = open("textfile.txt", "w", encoding="utf-8")`
3. Τα δεδομένα σε ένα αρχείο JSON είναι σε μορφή
   A. Λίστας
   B. Πλειάδων
   C. Συμβολοσειρών
   D. Λεξικού
4. Με ποια μέθοδο διαγράφεται ένας φάκελος;
   A. `shutil.rmtree`
   B. `os.rmtree`
   C. `shutil.rmdir`
   D. `shutil.remove`
5. Η μετατροπή ενός αρχείου XML σε HTML γίνεται με χρήση του:
   A. XSD
   B. XSLT
   C. XML2HTML
   D. SGML

5.11. Ασκήσεις προς επίλυση

1. Γράψτε ένα πρόγραμμα όπου θα ανοίγετε ένα αρχείο κειμένου για ανάγνωση και στα περιεχόμενά του θα γίνεται έρευνα και αντικατάσταση κειμένου. Τα αλλαγμένα περιεχόμενα θα

σώζονται στο αρχικό αρχείο. Το όνομα του αρχείου, το προς αντικατάσταση κείμενο και το κείμενο που θα πάει στη θέση του να ζητούνται κατά την έναρξη του προγράμματος.

2. Σας δίνεται ένα αρχείο CSV που περιέχει πληροφορίες για τουριστικά ταξίδια. Πρώτη στήλη είναι ο προορισμός, δεύτερη η διάρκεια σε ημέρες, τρίτη το κόστος ανά άτομο για μονόκλινο δωμάτιο, τέταρτη το κόστος ανά άτομο για δίκλινο και Πέμπτη στήλη το κόστος για επιπλέον παροχές, π.χ. προαιρετικές εκδρομές κλπ.. Το αρχείο έχει κεφαλίδα, η πρώτη γραμμή έχει τα ονόματα των στηλών. Φτιάξτε ένα ενδεικτικό τέτοιο αρχείο με μερικούς δικούς σας προορισμούς, διάρκειες, κλπ..

Γράψτε ένα πρόγραμμα που ανοίγει ένα αρχείο CSV για ανάγνωση και το μετατρέπει σε λεξικά, τα οποία αποθηκεύονται με χρήση της `shelve`. Τέλος, μετατρέψτε το σε φύλλο εργασίας και αποθηκεύστε το σαν αρχείο excel (.xlsx).

3. Το προηγούμενο αρχείο, αφού μετατραπεί σε λεξικά, να μετατρέπεται σε αρχείο JSON, όπου κάθε γραμμή είναι ένα αντικείμενο (object) JSON με κλειδί το όνομα της γραμμής και τιμή το λεξικό με τις τιμές (των στηλών) της γραμμής.

4. Συνεχίστε μετατρέποντας το ίδιο αρχείο σε αρχείο XML.

5. Γράψτε ένα πρόγραμμα που θα ψάχνει σε ένα μονοπάτι που του δίνεται για αρχεία CSV. Κάθε ένα που βρίσκει να το ανοίγει για ανάγνωση, να το μετατρέπει σε φύλλο εργασίας και να το αποθηκεύει στο ίδιο σημείο με το αρχικό, κρατώντας το ίδιο όνομα, αλλά αλλάζοντας την κατάληξή του σε .xlsx.

6. Το μέσο αποθήκευσής σας ξεμένει από χώρο. Για να βρείτε υποψήφια αρχεία και φακέλους για σβήσιμο ή μεταφορά σε άλλο μέσο, γράψτε ένα πρόγραμμα που θα παίρνει σαν είσοδο ένα μονοπάτι (να είναι φάκελος) και ψάχνει σε αυτό και όλους τους υποφακέλους του. Για κάθε αρχείο ή φάκελο που βρίσκει να διαβάζει το μέγεθος του αρχείου (θα βοηθήσει η `os.stat` και το `st_size` που επιστρέφει). Για κάθε φάκελο να τυπώνει το συνολικό μέγεθος των περιεχομένων του. Για σωστά αποτελέσματα, ρυθμίστε όπου χρειαστεί να μην ακολουθούνται οι λογικοί σύνδεσμοι (logical links – δώστε στην παράμερο `followlinks` την τιμή `False`).

ΚΕΦΑΛΑΙΟ 6: ΧΕΙΡΙΣΜΟΣ ΛΑΘΩΝ ΚΑΙ ΕΙΔΙΚΕΣ ΒΙΒΛΙΟΘΗΚΕΣ

Στο κεφάλαιο αυτό θα εξεταστεί ο χειρισμός λαθών χρόνου εκτέλεσης, δηλαδή λαθών που μπορούν να προκύψουν λόγω μη αναμενόμενης εισόδου από τον χρήστη ή λόγω καταστάσεων που δεν αντιστοιχούν στη συνηθισμένη ροή εκτέλεσης ενός προγράμματος όπως για παράδειγμα η απόπειρα πρόσβασης στα περιεχόμενα ενός αρχείου που δεν εντοπίζεται στη διαδρομή που δίνεται. Θα περιγραφεί αρχικά ο μηχανισμός των εξαιρέσεων που επιτρέπει το χειρισμό αυτών των καταστάσεων με τις εντολές `try-except-else-finally` και στη συνέχεια θα δοθούν παραδείγματα αμυντικού προγραμματισμού, δηλαδή δόμησης του κώδικα έτσι ώστε να αποτρέπονται λανθασμένες είσοδοι όπως για παράδειγμα λάθος τιμές από τον χρήστη. Επιπλέον, στο παρόν κεφάλαιο θα εξεταστούν τρόποι χειρισμού ημερομηνιών και χρονικών στιγμών και η χρήση τυχαιότητας ως δυνατότητα κατά την εκτέλεση του κώδικα. Τέλος, θα περιγραφεί η εγκατάσταση της βιβλιοθήκης jupyter και κατόπιν η χρήση notebooks (σημειωματάριων). Τα σημειωματάρια επιτρέπουν την καταγραφή υπολογιστικών διαδικασιών, βήμα προς βήμα, σε μορφή που τα αποτελέσματα είναι εύκολο να αναπαραχθούν εκ νέου, ενώ παράλληλα ο κώδικας συνοδεύεται από τεκμηρίωση με μορφοποιημένο κείμενο, εικόνες, συνδέσμους, γραφήματα κ.α.

6.1. Η ανάγκη για χειρισμό λαθών

Ο χειρισμός λαθών (error handling) αποτελεί θεμελιώδη πτυχή του προγραμματισμού, καθώς τα σφάλματα είναι αναπόφευκτα σε κάθε μη τετριμμένο πρόγραμμα. Είσοδοι από τον χρήστη, πρόσβαση σε αρχεία, δικτυακές επικοινωνίες ή περιορισμοί πόρων μπορούν εύκολα να οδηγήσουν σε καταστάσεις αποτυχίας. Χωρίς συστηματικό χειρισμό λαθών, ένα πρόγραμμα τερματίζει απότομα, παράγει ασαφή μηνύματα ή, ακόμη χειρότερα, οδηγείται σε λανθασμένα αποτελέσματα χωρίς προειδοποίηση. Ο σωστός χειρισμός λαθών βελτιώνει την αξιοπιστία και τη συντηρησιμότητα του κώδικα, επιτρέποντας στο πρόγραμμα να αντιδρά με συνεπή τρόπο σε μη αναμενόμενες καταστάσεις.

Η συστηματική χρήση του χειρισμού λαθών στην Python δεν αφορά μόνο την αποφυγή κατάρρευσης ενός προγράμματος, αλλά και το σωστό σχεδιασμό λογισμικού. Ο ορθός χειρισμός λαθών αποτελεί ένδειξη προγραμματιστικής ωριμότητας, καθώς δείχνει ότι ο προγραμματιστής έχει προβλέψει και ελέγξει τόσο τις οριακές όσο και τις μη αναμενόμενες συνθήκες εκτέλεσης. Επίσης, αποτελεί προαπαιτούμενη γνώση όταν κάποιος εργάζεται ως μέλος μιας ομάδας ανάπτυξης λογισμικού.

6.2. Εξαιρέσεις

Μια εξαίρεση (exception) είναι ένας μηχανισμός αναφοράς σφάλματος ή ασυνήθιστης κατάστασης που προκύπτει κατά την εκτέλεση ενός προγράμματος. Όταν συμβεί μια εξαίρεση, η κανονική ροή εκτέλεσης διακόπτεται και το πρόγραμμα προκαλεί την εξαίρεση (raises the exception). Αν η εξαίρεση δεν αντιμετωπιστεί, το πρόγραμμα τερματίζεται και εμφανίζεται σχετικό μήνυμα σφάλματος.

Στην Python, οι εξαιρέσεις υλοποιούνται ως αντικείμενα που ανήκουν σε μια ιεραρχία κλάσεων με βασική κλάση την `BaseException`. Κάθε τύπος εξαίρεσης περιγράφει ένα συγκεκριμένο είδος προβλήματος, όπως `ValueError` για μη αποδεκτές τιμές, `TypeError` για ασυμβατότητα τύπων ή `ZeroDivisionError` για διαίρεση με το μηδέν. Οι εξαιρέσεις μπορούν να προκύψουν αυτόματα από τη γλώσσα ή να δημιουργηθούν από τον προγραμματιστή με την εντολή `raise`.

Ο ρόλος μιας εξαίρεσης δεν είναι απλώς να δηλώσει ότι «κάτι πήγε στραβά», αλλά να επιτρέψει στο πρόγραμμα να αντιδράσει ελεγχόμενα. Με τη χρήση των `try-except`, ο προγραμματιστής μπορεί να ανιχνεύσει το σφάλμα και να επιχειρήσει να το διορθώσει προκειμένου να συνεχιστεί η εκτέλεση του προγράμματος με ασφάλεια.

6.2.1. Χειρισμός εξαιρέσεων με την try-except-else-finally

Η γενική σύνταξη της try-except, που προαιρετικά περιέχει τα τμήματα else και finally είναι η ακόλουθη:

```
try:
    # κώδικας που ενδεχόμενα θα προκαλέσει εξαίρεση
except ExceptionType [as e]:
    # κώδικας που εκτελείται αν συμβεί εξαίρεση τύπου ExceptionType
[else:
    # κώδικας που εκτελείται αν δεν συμβεί εξαίρεση]
[finally:
    # Κώδικας που εκτελείται σε κάθε περίπτωση (cleanup code)]
```

Σε έναν κώδικα με `try-except-else-finally`, το μπλοκ `try` περιέχει εντολές που ενδέχεται να προκαλέσουν εξαίρεση, και αν κατά την εκτέλεσή τους προκύψει εξαίρεση συγκεκριμένου τύπου, τότε η κανονική ροή διακόπτεται και εκτελείται το αντίστοιχο μπλοκ `except`, όπου γίνεται ο χειρισμός του σφάλματος. Αν δεν προκύψει καμία εξαίρεση μέσα στο `try`, εκτελείται το μπλοκ `else`, το οποίο συνήθως περιέχει κώδικα που βασίζεται στην επιτυχή ολοκλήρωση του `try`. Τέλος, το μπλοκ `finally` εκτελείται πάντα, ανεξάρτητα από το αν προέκυψε ή όχι εξαίρεση, και χρησιμοποιείται για ενέργειες αποδέσμευσης πόρων, όπως για παράδειγμα κλείσιμο αρχείων, εξασφαλίζοντας ότι το πρόγραμμα αφήνεται σε συνεπή κατάσταση.

Ένα παράδειγμα χειρισμού εξαιρέσεων παρουσιάζεται στον κώδικα Κ. 202, όπου ο χρήστης εισάγει δύο αριθμούς που θα πρέπει να είναι ακέραιοι, ο πρώτος αριθμός διαιρείται με τον δεύτερο και εκτυπώνεται το αποτέλεσμα της διαίρεσης. Υπάρχουν δύο περιπτώσεις όπου μπορεί να προκύψει σφάλμα. Η πρώτη περίπτωση είναι όταν ο χρήστης εισάγει τιμή που δεν είναι ακέραια, όπως για παράδειγμα το κείμενο `'ένα'`, οπότε θα προκύψει `ValueError` λόγω της αδυναμίας μετατροπής του κειμένου σε ακέραιο με τη συνάρτηση `int()`. Η περίπτωση αυτή καλύπτεται με το `except ValueError`. Η δεύτερη περίπτωση που μπορεί να προκύψει σφάλμα είναι όταν εισαχθεί για τη μεταβλητή `y` η τιμή 0, οπότε προκύπτει διαίρεση ακεραίων με παρονομαστή μηδέν, που προκαλεί `ZeroDivisionError`. Η περίπτωση αυτή καλύπτεται με το `except DivisionError`. Από την άλλη μεριά αν δεν προκληθεί καμία εξαίρεση, στο μπλοκ κώδικα του `try` τότε θα εκτελεστεί το μπλοκ κώδικα του `else` που εμφανίζει το αποτέλεσμα της διαίρεσης. Σε κάθε περίπτωση θα εμφανιστεί το μήνυμα «`Τερματισμός εκτέλεσης`» πριν ολοκληρωθεί η εκτέλεση του προγράμματος.

```
try:
    x = int(input("Δώσε έναν ακέραιο: "))
    y = int(input("Δώσε έναν ακόμη ακέραιο: "))
```

```python
    result = x / y
except ValueError:
    print("Σφάλμα: εισάγετε ακέραιες τιμές")
except ZeroDivisionError:
    print("Σφάλμα: διαίρεση με το μηδέν δεν επιτρέπεται")
else:
    print("Αποτέλεσμα:", result)
finally:
    print("Τερματισμός εκτέλεσης")
```

Κ. 202 – Παράδειγμα χειρισμού εξαιρέσεων με try-except

Η εκτέλεση του παραπάνω κώδικα για τρεις διαφορετικές περιπτώσεις εισόδου από τον χρήστη φαίνεται στην έξοδο Ε. 52.

Δώσε έναν ακέραιο: 1 Δώσε έναν ακόμη ακέραιο: 2 Αποτέλεσμα: 0.5 Τερματισμός εκτέλεσης	Δώσε έναν ακέραιο: 1 Δώσε έναν ακόμη ακέραιο: 0 Σφάλμα: διαίρεση με το μηδέν δεν επιτρέπεται Τερματισμός εκτέλεσης	Δώσε έναν ακέραιο: ένα Σφάλμα: εισάγετε ακέραιες τιμές Τερματισμός εκτέλεσης

Ε. 52 – Έξοδος προγράμματος για τρεις διαφορετικές εισόδους του χρήστη.

6.2.2. Η εντολή raise

Η εντολή raise χρησιμοποιείται για τη ρητή (explicit) πρόκληση μιας εξαίρεσης από τον ίδιο τον προγραμματιστή. Με τον τρόπο αυτό, ο κώδικας μπορεί να δηλώσει ότι έχει εντοπιστεί μια μη αποδεκτή ή επικίνδυνη κατάσταση, ακόμη κι αν δεν έχει προκύψει κάποιο σφάλμα από το ίδιο το σύστημα. Η raise μπορεί να χρησιμοποιηθεί είτε με μια ενσωματωμένη εξαίρεση (π.χ. ValueError, TypeError) είτε με εξαίρεση που έχει ορίσει ο ίδιος ο προγραμματιστής.

Στον κώδικα Κ. 203 παρουσιάζεται ένα παράδειγμα που προκαλείται η εξαίρεση ValueError στη συνάρτηση set_age() όταν περάσει ως όρισμα μια αρνητική τιμή. Η εκτέλεση του κώδικα που αφορά την κλήση της συνάρτησης με μη αρνητική τιμή θα γίνει κανονικά, ενώ η κλήση της συνάρτησης με αρνητική τιμή θα οδηγήσει είτε στον άμεσο τερματισμό του κώδικα, όπως στο παράδειγμα, είτε εφόσον υπάρχει κώδικας χειρισμού της εξαίρεσης με try-except στην εκτέλεση του κώδικα που χειρίζεται τη συγκεκριμένη εξαίρεση.

```python
def set_age(age):
    if age < 0:
        raise ValueError("Η ηλικία δεν μπορεί να είναι αρνητική")
    print("Η ηλικία ορίστηκε σε:", age)

set_age(20)     # κανονική εκτέλεση
set_age(-5)     # προκαλεί ValueError
```

Κ. 203 – Πρόκληση εξαίρεσης με την εντολή raise.

Επιπλέον, μέσα σε ένα μπλοκ except, η εντολή raise χωρίς όρισμα προκαλεί την ίδια εξαίρεση, διατηρώντας το αρχικό traceback. Αυτό είναι ιδιαίτερα χρήσιμο όταν θέλουμε να καταγράψουμε ή να χειριστούμε μερικώς ένα σφάλμα, αλλά να αφήσουμε την τελική ευθύνη χειρισμού σε ανώτερο επίπεδο του προγράμματος. Ένα παράδειγμα που δείχνει αυτό τον τρόπο χρήσης του raise μέσα σε ένα μπλοκ except παρουσιάζεται στον κώδικα Κ. 204 που υλοποιεί το χειρισμό εξαιρέσεων σε δύο επίπεδα, διαχωρίζοντας τον μερικό από τον τελικό χειρισμό του σφάλματος. Ειδικότερα, η συνάρτηση read_number_from_file() προσπαθεί να διαβάσει έναν ακέραιο από αρχείο και αν προκύψει οποιαδήποτε εξαίρεση (π.χ. το αρχείο δεν υπάρχει ή το περιεχόμενο δεν είναι αριθμός), η εξαίρεση «συλλαμβάνεται» στο except, όπου γίνεται καταγραφή του σφάλματος μέσω ενός ενημερωτικού μηνύματος που περιλαμβάνει το αντικείμενο της εξαίρεσης (e). Στη συνέχεια, η εντολή raise χωρίς όρισμα επαναπροκαλεί την ίδια εξαίρεση, διατηρώντας το αρχικό traceback. Στη main(), η εξαίρεση «συλλαμβάνεται» σε ανώτερο επίπεδο, όπου εμφανίζεται το πλήρες traceback με την κλήση της μεθόδου traceback.print_exc(), δίνοντας πλήρη εικόνα της αλυσίδας κλήσεων που οδήγησε στο σφάλμα. Τέλος, τυπώνεται ένα γενικό μήνυμα που δηλώνει ότι ο τελικός χειρισμός του σφάλματος έγινε στο ανώτερο επίπεδο του προγράμματος.

```
import traceback

def read_number_from_file(filename):
    try:
        with open(filename, "r") as f:
            return int(f.read())
    except Exception as e:
        # Μερικός χειρισμός: καταγραφή του σφάλματος
        print("Σφάλμα κατά την ανάγνωση αρχείου:", e)
        # Πρόκληση της ίδιας εξαίρεσης με το αρχικό traceback
        raise


def main():
    try:
        number = read_number_from_file("data.txt")
        print("Ο αριθμός είναι:", number)
    except Exception:
        traceback.print_exc()   # εμφανίζει πλήρες traceback
        print("Το σφάλμα χειρίστηκε σε ανώτερο επίπεδο του προγράμματος.")


main()
```
Κ. 204 – Πρόκληση της raise μέσα σε μπλοκ κώδικα except.

Ένα παράδειγμα εξόδου, για την περίπτωση που δεν εντοπίζεται το αρχείο `data.txt`, παρουσιάζεται στην έξοδο Ε. 53.

```
Σφάλμα κατά την ανάγνωση αρχείου: [Errno 2] No such file or directory: 'data.txt'
Traceback (most recent call last):
  File "c:\Users\user\PYTHON-A\ch6\raise2.py", line 16, in main
    number = read_number_from_file("data.txt")
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\user\PYTHON-A\ch6\raise2.py", line 5, in read_number_from_file
    with open(filename, "r") as f:
         ^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
Το σφάλμα χειρίστηκε σε ανώτερο επίπεδο του προγράμματος.
```

Ε. 53 – Έξοδος κώδικα Κ. 204 για την περίπτωση που το αρχείο data.txt δεν υπάρχει στην αναμενόμενη θέση.

6.3. Αμυντικός προγραμματισμός

Ο αμυντικός προγραμματισμός (defensive programming) είναι μια πρακτική ανάπτυξης λογισμικού που στοχεύει στη συγγραφή κώδικα που προβλέπει και αντιμετωπίζει λανθασμένες ή απρόβλεπτες καταστάσεις, ακόμη και όταν αυτές θεωρητικά δεν θα έπρεπε να συμβούν. Κεντρικό εργαλείο αυτής της φιλοσοφίας είναι η εντολή `assert`, η οποία χρησιμοποιείται για να ελέγχει υποθέσεις, τα λεγόμενα invariants (προϋποθέσεις), που ο προγραμματιστής θεωρεί δεδομένες κατά την εκτέλεση του προγράμματος, όπως ότι μια τιμή είναι θετική ή ότι μια δομή δεδομένων δεν είναι κενή. Αν μια τέτοια υπόθεση αποτύχει, το πρόγραμμα διακόπτεται άμεσα, αποκαλύπτοντας έγκαιρα ένα λογικό σφάλμα κατά την ανάπτυξη ή τον έλεγχο.

Πέρα από το `assert`, ο αμυντικός προγραμματισμός περιλαμβάνει ελέγχους εγκυρότητας εισόδων, κατάλληλο χειρισμό εξαιρέσεων, σαφή μηνύματα σφάλματος και αποφυγή «σιωπηλών» αποτυχιών. Στόχος δεν είναι απλώς να μην καταρρέει το πρόγραμμα, αλλά να αποτυγχάνει με ελεγχόμενο και κατανοητό τρόπο, διευκολύνοντας τον εντοπισμό σφαλμάτων και αυξάνοντας τη συνολική αξιοπιστία και συντηρησιμότητα του λογισμικού.

6.3.1. Η εντολή `assert`

Η εντολή assert, όπως αναφέρθηκε, χρησιμοποιείται για τον έλεγχο προϋποθέσεων κατά την εκτέλεση ενός προγράμματος. Ελέγχει αν μια λογική συνθήκη είναι αληθής και, αν δεν είναι, προκαλεί εξαίρεση τύπου `AssertionError`, διακόπτοντας άμεσα την εκτέλεση εμφανίζοντας ένα σφάλμα. Η γενική σύνταξη της `assert` είναι `assert συνθήκη` ή `assert συνθήκη, μήνυμα`, όπου το προαιρετικό μήνυμα εμφανίζεται όταν αποτύχει ο έλεγχος. Για παράδειγμα, στον κώδικα Κ. 205 οι εντολές `assert` χρησιμοποιούνται για να ελεγχθούν δύο βασικές προϋποθέσεις ορθής

λειτουργίας της συνάρτησης `withdraw()`. Η πρώτη είναι ότι το ποσό της ανάληψης θα πρέπει να μην είναι αρνητικό και η δεύτερη ότι το ποσό της ανάληψης δεν θα πρέπει να υπερβαίνει το διαθέσιμο υπόλοιπο. Αν κάποια από αυτές τις συνθήκες αποτύχει, το πρόγραμμα διακόπτεται με `AssertionError`, υποδεικνύοντας ξεκάθαρα ένα λογικό σφάλμα στον τρόπο κλήσης της συνάρτησης.

```
def withdraw(balance, amount):
    assert amount >= 0, "Το ποσό ανάληψης δεν μπορεί να είναι αρνητικό"
    assert amount <= balance, "Ανεπαρκές υπόλοιπο"
    return balance - amount

balance = 100
balance = withdraw(balance, 30)   # σωστή χρήση
balance = withdraw(balance, 150)  # προκαλεί AssertionError
```

Κ. 205 – Παράδειγμα χρήσης της εντολής assert για τη διασφάλιση ορθών τιμών ορισμάτων στην κλήση της withdraw().

6.3.2. Παράδειγμα αμυντικού προγραμματισμού με εξαιρέσεις και επανάληψη

Στην παρούσα παράγραφο θα παρουσιαστεί ένα παράδειγμα αμυντικού προγραμματισμού (κώδικας Κ. 206) που συνδυάζει χειρισμό εξαιρέσεων με `try-except`, πρόκληση εξαιρέσεων με `raise` και επανάληψη με `while` έτσι ώστε σε περίπτωση που η είσοδος δεν είναι αποδεκτή, να επιχειρείται ξανά. Ειδικότερα, ο κώδικας ζητά επαναληπτικά έναν θετικό ακέραιο μέχρι να δοθεί έγκυρη τιμή. Σε κάθε επανάληψη αυξάνεται ο μετρητής `attempts`, γίνεται προσπάθεια μετατροπής της εισόδου σε ακέραιο (`int`), και αν η μετατροπή αποτύχει ή αν ο αριθμός δεν είναι θετικός, προκαλείται ή συλλαμβάνεται `ValueError`, εμφανίζεται μήνυμα σφάλματος και ο αριθμός προσπαθειών, και ο βρόχος συνεχίζεται. Όταν δοθεί έγκυρη είσοδος, εκτελείται `break`, ο βρόχος τερματίζει και στο τέλος εμφανίζονται τόσο ο έγκυρος πλέον αριθμός που δόθηκε από τον χρήστη όσο και το πλήθος των προσπαθειών που έγιναν.

```
attempts = 0

while True:
    attempts += 1
    user_input = input("Δώσε έναν θετικό ακέραιο αριθμό: ")
    try:
        n = int(user_input)
        if n <= 0:
            raise ValueError("Ο αριθμός πρέπει να είναι θετικός")
        break
    except ValueError:
        print("Λάθος είσοδος: πρέπει να δώσεις θετικό ακέραιο αριθμό.")
        print("Αριθμός προσπαθειών:", attempts)
```

```
print("Έδωσες τον αριθμό:", n)
print("Συνολικός αριθμός προσπαθειών:", attempts)
```
Κ. 206 – Επαναληπτική εισαγωγή τιμών μέχρι να δοθεί ένας θετικός ακέραιος.

Στην έξοδο Ε. 54 παρουσιάζεται ένα αποτέλεσμα εκτέλεσης του παραπάνω κώδικα, όπου ο χρήστης τελικά εισάγει έναν θετικό ακέραιο στην 3η προσπάθεια εισαγωγής προκειμένου να τερματίσει το πρόγραμμα.

```
Δώσε έναν θετικό ακέραιο αριθμό: -1
Λάθος είσοδος: πρέπει να δώσεις θετικό ακέραιο αριθμό.
Αριθμός προσπαθειών: 1
Δώσε έναν θετικό ακέραιο αριθμό: ένα
Λάθος είσοδος: πρέπει να δώσεις θετικό ακέραιο αριθμό.
Αριθμός προσπαθειών: 2
Δώσε έναν θετικό ακέραιο αριθμό: 42
Έδωσες τον αριθμό: 42
Συνολικός αριθμός προσπαθειών: 3
```
Ε. 54 – Έξοδος για 2 λανθασμένες εισαγωγές τιμών από τον χρήστη που ακολουθούνται από εισαγωγή 1 ορθής τιμής.

6.3.3. Άσκηση 1, με πρόκληση και σύλληψη εξαίρεσης

Να υλοποιήσετε πρόγραμμα που να ζητά από τον χρήστη να εισάγει 5 πραγματικές αριθμητικές τιμές, μία-μία. Κάθε νέα τιμή που δίνεται πρέπει να είναι αυστηρά μεγαλύτερη από την αμέσως προηγούμενη, ενώ σε διαφορετική περίπτωση, το πρόγραμμα να προκαλεί εξαίρεση `ValueError` με κατάλληλο μήνυμα σφάλματος. Το πρόγραμμα να τερματίζει μόλις προκύψει μη έγκυρη αύξουσα είσοδος, διαφορετικά, αν και οι 5 τιμές δοθούν σωστά, να εμφανίζει μήνυμα επιτυχούς ολοκλήρωσης.

Η λύση της άσκησης βρίσκεται στο Παράρτημα Α'.

6.4. Τυχαιότητα

Η δυνατότητα παραγωγής τυχαίων αριθμών είναι θεμελιώδης στις γλώσσες προγραμματισμού, επειδή επιτρέπει τη μοντελοποίηση και προσομοίωση μη προβλέψιμων φαινομένων, την υλοποίηση πιθανοκρατικών αλγορίθμων και τη δημιουργία ρεαλιστικών σεναρίων δοκιμών. Τυχαίοι αριθμοί χρησιμοποιούνται εκτενώς σε προσομοιώσεις (π.χ. Monte Carlo), σε παιχνίδια, σε αλγορίθμους βελτιστοποίησης και μηχανικής μάθησης, στη στατιστική δειγματοληψία, στην ασφάλεια πληροφοριών και αλλού.

Η Python διαθέτει στην τυπική βιβλιοθήκη της το module `random` που περιέχει διάφορες συναρτήσεις παραγωγής ψευδο-τυχαίων τιμών και επιλογής με τυχαίο τρόπο από πεπερασμένες ακολουθίες τιμών. Οι σημαντικότερες από τις συναρτήσεις της `random` είναι οι ακόλουθες:

- `random()` που επιστρέφει έναν τυχαίο πραγματικό αριθμό στο διάστημα `[0.0, 1.0)`.
- `randint(a, b)` που επιστρέφει έναν τυχαίο ακέραιο στο κλειστό διάστημα `[a, b]`.
- `randrange(start, stop[, step])` που επιλέγει τυχαία έναν ακέραιο από μια αριθμητική ακολουθία, αντίστοιχη της `range()`.
- `choice(seq)` που επιλέγει τυχαία ένα στοιχείο από μια ακολουθία.
- `shuffle(seq)` που αναδιατάσσει τυχαία τα στοιχεία μιας λίστας, επιτόπου (in place).
- `sample(population, k)` που επιστρέφει k διαφορετικά τυχαία στοιχεία από ένα οποιοδήποτε iterable (π.χ., λίστα, πλειάδα, σύνολο, συμβολοσειρά, αντικείμενο `range`).

Στο απόσπασμα κώδικα Α. 27 παρουσιάζονται παραδείγματα χρήσης των παραπάνω συναρτήσεων με σχόλια για την λειτουργία τους.

```
>>> import random
>>> random.random() # Επιστρέφει έναν ψευδοτυχαίο πραγματικό αριθμό στο διάστημα [0.0,1.0).
0.374921603847219
>>> random.randint(1, 6) # Επιστρέφει έναν τυχαίο ακέραιο από το κλειστό διάστημα [1,6].
4
>>> random.randrange(0, 10, 2) # Επιλέγει τυχαία ένα στοιχείο της ακολουθίας [0, 2, 4, 6,
8].
6
>>> random.choice(["red", "green", "blue"]) # Επιλέγει τυχαία ένα στοιχείο από μια
ακολουθία.
'green'
>>> lst = [1, 2, 3, 4]
>>> random.shuffle(lst) # Αναδιατάσσει τυχαία τα στοιχεία της λίστας lst τροποποιώντας την
ίδια.
>>> lst
[3, 1, 4, 2]
>>> random.sample(range(1, 11), 3) # Επιστρέφει 3 διαφορετικά τυχαία στοιχεία από έναν
iterable πληθυσμό.
[2, 7, 9]
```
Α. 27 – Παραδείγματα με κλήσεις συναρτήσεων του module random.

Για εφαρμογές που απαιτούν κρυπτογραφικά ισχυρή τυχαιότητα, μπορεί να χρησιμοποιηθεί το module `secrets`, ενώ για αριθμητικούς και επιστημονικούς υπολογισμούς μεγάλης κλίμακας, η εξωτερική βιβλιοθήκη numpy και το module της `random` που είναι αποδοτικότερο έναντι του module `random` της τυπικής βιβλιοθήκης.

6.4.1. Αναπαραγωγιμότητα αποτελεσμάτων με seed(s)

Η αναπαραγωγιμότητα (reproducibility) είναι ιδιαίτερα σημαντική στην πληροφορική και γενικά στην επιστήμη, επειδή επιτρέπει την επαλήθευση αποτελεσμάτων, τη σύγκριση πειραμάτων και τον αξιόπιστο εντοπισμό σφαλμάτων. Με τη χρήση του λεγόμενου seed (σπόρος) καθορίζεται η αρχική κατάσταση της γεννήτριας ψευδοτυχαίων αριθμών. Η βασική αρχή είναι ότι για το ίδιο seed, η ακολουθία «τυχαίων» αριθμών που παράγεται είναι ακριβώς η ίδια σε κάθε εκτέλεση. Αυτό είναι ιδιαίτερα χρήσιμο σε πειράματα, προσομοιώσεις και δοκιμές λογισμικού, όπου θέλουμε τα

αποτελέσματα να είναι ελέγξιμα και επαναλήψιμα, χωρίς να χάνεται η τυχαιότητα. Στην Python, αν δεν οριστεί seed, η γεννήτρια ψευδοτυχαίων αριθμών αρχικοποιείται με βάση την τρέχουσα χρονική στιγμή, οδηγώντας σε διαφορετικές τιμές σε κάθε εκτέλεση.

Στον κώδικα Κ. 207 παρουσιάζεται ένα παράδειγμα όπου ορίζεται seed και κάθε εκτέλεση του θα επιστρέφει τις ίδιες τιμές. Αν δεν υπήρχε η εντολή random.seed(42) τότε κάθε εκτέλεση θα επέστρεφε και διαφορετικό αποτέλεσμα.

```
import random

random.seed(42)
print(random.random()) # 0.6394267984578837
print(random.random()) # 0.025010755222666936
```
Κ. 207 – Εκτύπωση δύο τυχαίων τιμών, για συγκεκριμένο seed.

6.4.2. Άσκηση 2, με τυχαίες τιμές

Έστω ότι ζητείται η προσομοίωση ενός κλασικού ζαριού έξι πλευρών. Να γράψετε πρόγραμμα που υλοποιεί συνάρτηση με όνομα roll_die(n) που δέχεται ως όρισμα έναν θετικό ακέραιο αριθμό n και πραγματοποιεί n τυχαίες ρίψεις ζαριού με χρήση της συνάρτησης random.randint(1, 6). Το πρόγραμμα να υπολογίζει πόσες φορές εμφανίζεται κάθε τιμή από το 1 έως και το 6 και να εμφανίζει τις ποσοστιαίες συχνότητες εμφάνισης. Η προσομοίωση να εκτελείται για τις τιμές του n: 100, 1000 και 10000, ώστε να μελετηθεί η επίδραση του πλήθους των ρίψεων στα αποτελέσματα.

Η λύση της άσκησης βρίσκεται στο Παράρτημα Α'.

6.5. Χειρισμός ημερομηνιών, ωρών και χρονικών σημάνσεων

Η τυπική βιβλιοθήκη της Python διαθέτει το module `datetime`, το module `calendar` και το module `time` για τον χειρισμό ημερομηνιών, ωρών, χρονικών διαστημάτων και χρονικών σημάνσεων (timestamps). Το module `datetime` περιλαμβάνει κλάσεις όπως οι `date`, `time`, `datetime` και `timedelta` που επιτρέπουν την αναπαράσταση ημερομηνιών, ωρών, συνδυασμών ημερομηνιών-ωρών και διαφορών χρόνου αντίστοιχα. Επιπλέον, υπάρχουν εξωτερικές βιβλιοθήκες που προσφέρουν πιο προηγμένες δυνατότητες, όπως η `dateutil`, η οποία διευκολύνει τον χειρισμό χρονικών ζωνών και την ανάλυση (parsing) ημερομηνιών.

6.5.1. Το module `datetime`

Το module `datetime` υποστηρίζει την επεξεργασία δεδομένων ημερομηνίας και ώρας. Καθιστά δυνατή την πρόσθεση και την αφαίρεση χρονικών διαστημάτων σε ημερομηνίες, τη σύγκριση ημερομηνιών και ωρών, καθώς και τη μορφοποίηση και εξαγωγή συγκεκριμένων πληροφοριών

(π.χ., αριθμός μήνα, αριθμός ημέρας μήνα κ.λπ.) από ημερομηνίες και ώρες με χρήση προτύπων (patterns). Παρέχει υποστήριξη για χρονικές ζώνες μέσω του `tzinfo` και δίνει τη δυνατότητα ανάπτυξης πρακτικά χρήσιμων εφαρμογών που απαιτούν αξιόπιστη διαχείριση χρόνου. Στα αποσπάσματα κώδικα Α. 28, Α. 29, Α. 30, Α. 31, Α. 32 παρουσιάζονται παραδείγματα χρήσης του module `datetime` που επιδεικνύουν δυνατότητες του που αναφέρθηκαν παραπάνω.

```
>>> from datetime import datetime, date, time
>>> dt = datetime(2026, 2, 3, 10, 30, 0)
>>> d = date(2026, 2, 3)
>>> t = time(10, 30)
>>> dt, d, t
(datetime.datetime(2026, 2, 3, 10, 30), datetime.date(2026, 2, 3), datetime.time(10, 30))
```

Α. 28 - Δημιουργία αντικειμένων α) ημερομηνίας και ώρας β) ημερομηνίας γ) ώρας.

```
>>> from datetime import datetime, timedelta
>>> now = datetime.now()
>>> future = now + timedelta(days=7, hours=3)
>>> now, future
(datetime.datetime(2026, 2, 3, 1, 38, 5, 808884), datetime.datetime(2026, 2, 10, 4, 38, 5,
808884))
>>> future - now
datetime.timedelta(days=7, seconds=10800)
```

Α. 29 - Πρόσθεση και αφαίρεση χρονικών διαστημάτων, αφαίρεση ημερομηνιών και ωρών.

```
>>> from datetime import datetime
>>> d1 = datetime(2026, 2, 1, 9, 0)
>>> d2 = datetime(2026, 2, 3, 10, 30)
>>> if d1 < d2:
...     print(f"Η {d1} προηγείται της {d2}")
...
Η 2026-02-01 09:00:00 προηγείται της 2026-02-03 10:30:00
```

Α. 30 – Σύγκριση ημερομηνιών.

```
>>> from datetime import datetime
>>> text = "03-02-2026 10:30"
>>> dt = datetime.strptime(text, "%d-%m-%Y %H:%M")
>>> dt
>>> dt.strftime("%Y-%m-%d %H:%M")
'2026-02-03 10:30'
```

Α. 31 – Μετατροπή συμβολοσειράς σε αντικείμενο datetime και αντίστροφα.

```
>>> from datetime import datetime, timezone, timedelta
>>> meeting_utc = datetime(2026, 2, 3, 14, 0, tzinfo=timezone.utc)
>>> athens_tz = timezone(timedelta(hours=2))
>>> meeting_athens = meeting_utc.astimezone(athens_tz)
>>> meeting_athens
datetime.datetime(2026,2,3,16,0,tzinfo=datetime.timezone(datetime.timedelta(seconds=7200)))
```

Α. 32 – Αλλαγή timezone χρησιμοποιώντας διαφορά σε ώρες από το UTC (Coordinated Universal Time).

Από την έκδοση 3.9 της Python υπάρχει το module `zoneinfo` για ευκολότερο χειρισμό ημερομηνιών και ωρών σε διάφορες χρονικές ζώνες (timezones) με βάση ονομασίες, όπως για παράδειγμα "Europe/Athens". Ωστόσο, για να λειτουργήσει στα Windows θα πρέπει να

εγκατασταθεί πρώτα η εξωτερική βιβλιοθήκη `tzdata`. Μια εναλλακτική λύση χειρισμού χρονικών ζωνών θα παρουσιαστεί στην επόμενη παράγραφο με την εξωτερική βιβλιοθήκη `dateutil`.

6.5.2. Η εξωτερική βιβλιοθήκη `dateutil`

Η εξωτερική βιβλιοθήκη `dateutil` επεκτείνει και συμπληρώνει τις δυνατότητες του module `datetime`, παρέχοντας πιο ευέλικτη και πρακτική διαχείριση ημερομηνιών και ωρών. Υποστηρίζει αυτόματη αναγνώριση ημερομηνιών από συμβολοσειρές χωρίς να απαιτείται η εισαγωγή προκαθορισμένων προτύπων, όπως γίνεται με την `datetime.strptime()`, προηγμένες πράξεις με σχετικές χρονικές μετατοπίσεις (π.χ. μήνες, έτη), καθώς και πλήρη υποστήριξη χρονικών ζωνών.

Η εγκατάσταση της βιβλιοθήκης `dateutil` μπορεί να γίνει με το `pip`. Η εντολή που χρειάζεται να δοθεί είναι η ακόλουθη:

```
pip install python-dateutil
```

Στα αποσπάσματα κώδικα Α. 33, Α. 34 και Α. 35 παρουσιάζονται παραδείγματα των δυνατοτήτων της `dateutil`.

```
>>> from dateutil import parser
>>> parser.parse("3 Feb 2026 10:30")
datetime.datetime(2026, 2, 3, 10, 30)
>>> parser.parse("2026/02/03 10:30 am")
datetime.datetime(2026, 2, 3, 10, 30)
```

Α. 33 – Μετατροπή ημερομηνιών και ωρών από λεκτικά με διάφορες μορφές σε αντικείμενα datetime.

```
>>> from datetime import datetime
>>> from dateutil.relativedelta import relativedelta
>>> dt = datetime(2026, 1, 31)
>>> dt + relativedelta(months=1)
datetime.datetime(2026, 2, 28, 0, 0)
```

Α. 34 – Μετατόπιση ημερομηνίας και ώρας προσθέτοντας χρονικό διάστημα ενός μήνα.

```
>>> from dateutil import tz
>>> from datetime import datetime
>>> from dateutil import tz
>>> utc = tz.UTC
>>> athens = tz.gettz("Europe/Athens")
>>> dt_utc = datetime(2026, 2, 3, 9, 0, tzinfo=utc)
>>> dt_athens = dt_utc.astimezone(athens)
>>> dt_utc
datetime.datetime(2026, 2, 3, 9, 0, tzinfo=tzutc())
>>> dt_athens
datetime.datetime(2026, 2, 3, 11, 0, tzinfo=tzfile('Europe/Athens'))
```

Α. 35 – Μετατροπή ώρας από τη χρονική ζώνη UTC στη χρονική ζώνη Europe/Athens.

6.5.3. To module `calendar`

Το module `calendar` επιτρέπει την εμφάνιση μηνιαίων και ετήσιων ημερολογίων σε μορφή κειμένου, τον προσδιορισμό της ημέρας της εβδομάδας για συγκεκριμένη ημερομηνία, τον υπολογισμό του πλήθους ημερών ενός μήνα καθώς και τον έλεγχο αν ένα έτος είναι δίσεκτο.

Γενικότερα διευκολύνει την ανάπτυξη προγραμμάτων που πρέπει να ακολουθούν τους κανόνες που ισχύουν στο συνηθισμένο (Γρηγοριανό) ημερολόγιο. Στα αποσπάσματα οθόνης Α. 36, Α. 37, Α. 38 και Α. 39 παρουσιάζονται παραδείγματα με το module `calendar`.

```
>>> print(calendar.month(2026, 2))
   February 2026
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28
```

Α. 36 – Εκτύπωση ημερολογίου για τον Φεβρουάριο του 2026.

```
>>> calendar.monthrange(2026, 2)
(6, 28)
```

Α. 37 – Επιστροφή αριθμού ημέρας εβδομάδας (0=Δευτέρα, ..., 6=Κυριακή) και αριθμού ημερών για τον Φεβ. του 2026.

```
>>> calendar.weekday(2026, 2, 3)
1
```

Α. 38 – Επιστροφή ημέρας εβδομάδας για την ημερομηνία 3 Φεβ. 2026

```
>>> calendar.isleap(2024), calendar.isleap(2026), calendar.isleap(2028)
(True, False, True)
```

Α. 39 – Έλεγχος αν τα έτη 2024, 2026, 2028 είναι δίσεκτα ή όχι.

6.5.4. Το module `time`

Το module `time` παρέχει χαμηλού επιπέδου λειτουργίες και χρησιμοποιείται για πρόσβαση στην τρέχουσα χρονική στιγμή ως χρονική σήμανση (timestamp), την παύση εκτέλεσης και τη μετατροπή μεταξύ timestamp και δομών χρόνου εύκολων στην ανάγνωση. Είναι δε ιδιαίτερα χρήσιμο σε προγράμματα που είναι επιθυμητή η μέτρηση χρόνου εκτέλεσης τμημάτων κώδικα. Στα αποσπάσματα κώδικα Α. 40, Α. 41 παρουσιάζονται παραδείγματα με συναρτήσεις του module `time`. Αξίζει να σημειωθεί ότι η κλήση `time.time()` επιστρέφει τη σήμανση χρόνου της τρέχουσας χρονικής στιγμής σε δευτερόλεπτα που έχουν περάσει από το λεγόμενο UNIX EPOCH TIME που είναι η 1 Ιανουαρίου 1970, 00:00:00 UTC.

```
>>> import time
>>> ts = time.time()
>>> ts
1770161875.6923254
>>> time.ctime(ts)
'Wed Feb  4 01:37:55 2026'
>>> time.localtime(ts)
time.struct_time(tm_year=2026, tm_mon=2, tm_mday=4, tm_hour=1, tm_min=37, tm_sec=55,
tm_wday=2, tm_yday=35, tm_isdst=0)
```

Α. 40 – Λήψη timestamp τρέχουσας χρονικής στιγμής, αναγνώσιμη εμφάνιση, μετατροπή σε δομή με πλήρη στοιχεία.

Το παρακάτω απόσπασμα κώδικα Α. 41 έχει τοποθετηθεί στο μπλοκ μιας εντολής `if`, έτσι ώστε να μπορεί να εισαχθεί και να εκτελεστεί ως μπλοκ εντολών στο REPL.

```
>>> if True:
...     start = time.time()
...     time.sleep(1)
...     tot = sum(range(1_000_000))
...     print(time.time() - start)
...
1.0235493183135986
```

Α. 41 – Χρονομέτρηση κώδικα που κάνει παύση για 1 δευτερόλεπτο και υπολογίζει το άθροισμα 1.000.000 ακεραίων.

6.5.5. Άσκηση 3, με χειρισμό ημερομηνιών

Να γράψετε πρόγραμμα που να δέχεται μια ημερομηνία από τον χρήστη και να εμφανίζει το όνομα της ημέρας μέσα στην εβδομάδα για την συγκεκριμένη ημερομηνία (π.χ. για τις 4/2/2026 θα πρέπει να εμφανίζει Τετάρτη). Επίσης, να εμφανίζει πότε ξανά ο ίδιος αριθμός ημέρας και μήνα θα έχει το ίδιο όνομα ημέρας.

Η λύση της άσκησης βρίσκεται στο Παράρτημα Α'.

6.6. Jupyter Notebooks

Το Jupyter Notebook είναι ένα διαδραστικό περιβάλλον εργασίας που χρησιμοποιείται ευρέως στην εκπαίδευση, στην έρευνα και στην ανάλυση δεδομένων. Επιτρέπει τη συγγραφή και εκτέλεση κώδικα μέσα από ένα πρόγραμμα περιήγησης (web browser), συνδυάζοντας στον ίδιο χώρο κώδικα, μορφοποιημένο κείμενο, εικόνες, μαθηματικούς τύπους, πίνακες και γραφήματα. Με τον τρόπο αυτό, το Jupyter Notebook λειτουργεί όχι μόνο ως εργαλείο προγραμματισμού, αλλά και ως μέσο τεκμηρίωσης και παρουσίασης υπολογιστικών πειραμάτων.

Η βασική δομική μονάδα ενός notebook είναι το κελί (cell). Υπάρχουν κυρίως δύο τύποι κελιών: τα κελιά κώδικα και τα κελιά Markdown. Στα κελιά κώδικα ο χρήστης γράφει εντολές σε μια γλώσσα προγραμματισμού (συνήθως Python) και τις εκτελεί, βλέποντας αμέσως το αποτέλεσμα όταν «εκτελεί» το κελί. Στα κελιά Markdown μπορεί να προσθέσει μορφοποιημένο κείμενο, επικεφαλίδες, λίστες, συνδέσμους ή μαθηματικές εκφράσεις σε σύνταξη LaTeX. Ο συνδυασμός αυτός επιτρέπει την ανάπτυξη εγγράφων, όπου ο κώδικας και τα αποτελέσματα συνυπάρχουν. Τα notebooks μπορούν να αποθηκευτούν, να κοινοποιηθούν και να εκτελεστούν ξανά από άλλους χρήστες, ενισχύοντας τη συνεργασία και την αναπαραγωγιμότητα (reproducibility).

Αξίζει να επισημανθεί ότι το Jupyter Notebook δεν περιορίζεται σε μία μόνο γλώσσα προγραμματισμού. Μέσω διαφορετικών kernels μπορεί να υποστηρίξει, εκτός από Python, και άλλες γλώσσες όπως η R και η Julia.

6.6.1. Εγκατάσταση και εκκίνηση Jupyter Notebook

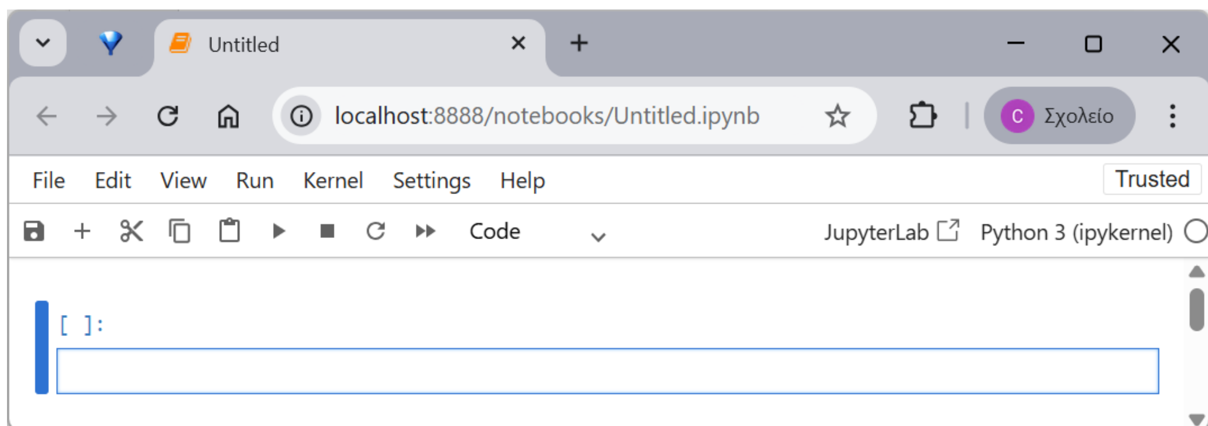
Το Jupyter Notebook μπορεί να εγκατασταθεί με το pip δίνοντας την ακόλουθη εντολή:

```
pip install notebook
```

Μετά την εγκατάσταση, η εφαρμογή Jupyter Notebook εκκινείται με την εντολή:

```
jupyter notebook
```

Η εντολή αυτή ανοίγει αυτόματα έναν web browser, όπου εμφανίζεται το περιβάλλον του Jupyter Notebook όπως φαίνεται στην εικόνα Εικόνα 15, μετά την επιλογή από το μενού: File → New → Notebook. Το όνομα που δίνεται αυτόματα στο σημειωματάριο είναι `Untitled.ipynb` ή παρόμοιο με κατάληξη `.ipynb` (IPython notebook). Το όνομα αυτό μπορεί να αλλάξει από το μενού με την επιλογή File → Rename.



Εικόνα 15 – Η αρχική οθόνη που εμφανίζεται κατά τη δημιουργία ενός νέου Jupyter Notebook.

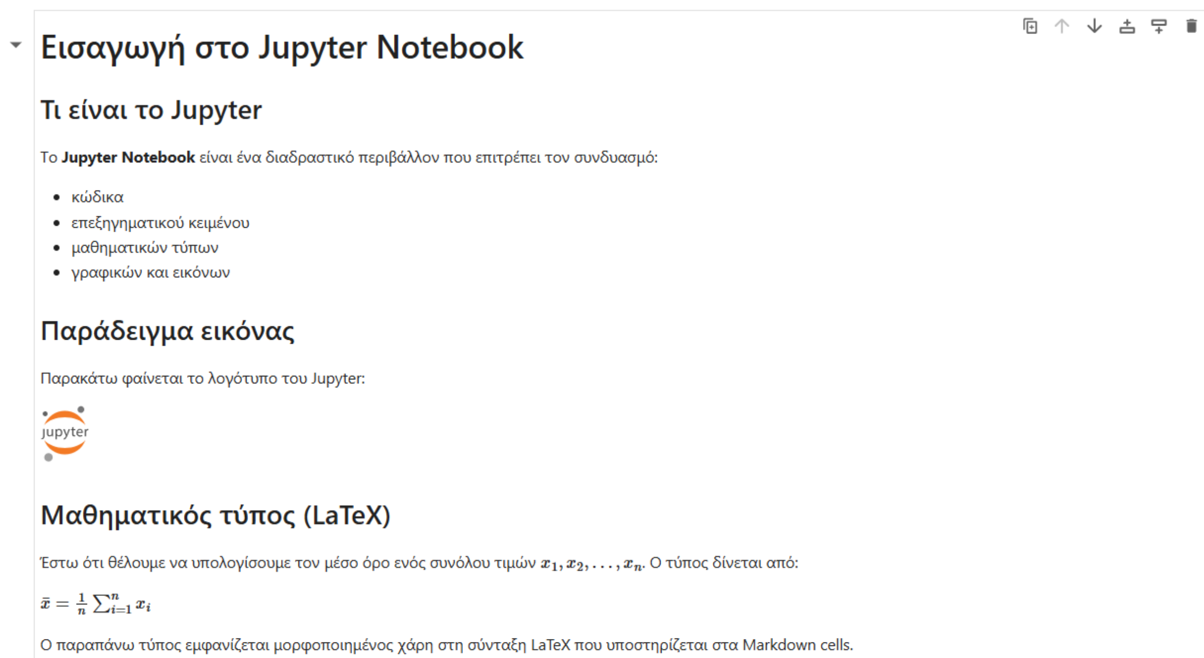
6.6.2. Βασικά παραδείγματα χρήσης Jupyter Notebook

Εδώ θα παρουσιαστούν ορισμένα παραδείγματα χρήσης του Jupyter Notebook, ξεκινώντας από τα κελιά markdown. Στο κελί που φαίνεται στην Εικόνα 16 παρουσιάζεται η κωδικοποίηση ενός κειμένου που περιέχει κείμενα με επικεφαλίδες επιπέδου 1 και επιπέδου 2, μια λίστα, μια εικόνα και δύο μαθηματικούς τύπους.

```
# Εισαγωγή στο Jupyter Notebook

## Τι είναι το Jupyter
Το **Jupyter Notebook** είναι ένα διαδραστικό περιβάλλον που επιτρέπει τον συνδυασμό:
- κώδικα
- επεξηγηματικού κειμένου
- μαθηματικών τύπων
- γραφικών και εικόνων

## Παράδειγμα εικόνας
Παρακάτω φαίνεται το λογότυπο του Jupyter:

![Jupyter Logo](https://jupyter.org/assets/homepage/main-logo.svg)

## Μαθηματικός τύπος (LaTeX)
Έστω ότι θέλουμε να υπολογίσουμε τον μέσο όρο ενός συνόλου τιμών $x_1, x_2, \dots, x_n$.
Ο τύπος δίνεται από:

$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$

Ο παραπάνω τύπος εμφανίζεται μορφοποιημένος χάρη στη σύνταξη LaTeX που υποστηρίζεται στα Markdown cells.
```

Εικόνα 16 – Ο κώδικας ενός κελιού markdown.

Η μορφοποιημένη εμφάνιση του παραπάνω κελιού παρουσιάζεται στην Εικόνα 17.



Εικόνα 17 – Μορφοποιημένη εμφάνιση περιεχομένου με markdown.

Στα κελιά που εμφανίζονται στην Εικόνα 18, παρουσιάζονται δύο παραδείγματα κελιών που περιέχουν κώδικα Python. Γενικά, για να εκτελεστεί ο κώδικας σε ένα κελί αρκεί να επιλεγεί το μενού Run → Run Selected Cell ή να πατηθεί το εικονίδιο με το σχήμα “play” ή να πατηθεί ο συνδυασμός πλήκτρων Shift+Enter. Σε κάθε περίπτωση ο κέρσορας θα πρέπει να βρίσκεται μέσα στο κελί που περιέχει τον κώδικα, σε οποιοδήποτε σημείο του.

Η εκτέλεση του κώδικα στο πρώτο κελί απλά θα εμφανίσει το άθροισμα δύο μεταβλητών. Η έκφραση που βρίσκεται στην τελευταία γραμμή κώδικα του κελιού που εκτελείται είναι αυτή για την οποία η τιμή της θα εμφανιστεί κάτω από κελί κώδικα. Η εκτέλεση του κώδικα στο δεύτερο κελί περιέχει μια επανάληψη και θα εμφανίσει το άθροισμα των στοιχείων της λίστας numbers του παραδείγματος.

```
[1]: # Εκτέλεση κώδικα σε κελί
     x = 10
     y = 20
     x + y

[1]: 30

[2]: # Απλό παράδειγμα με λίστα και επανάληψη
     numbers = [1, 2, 3, 4, 5]
     s = 0
     for n in numbers:
         s += n
     s

[2]: 15
```

Εικόνα 18 – Παραδείγματα κελιών κώδικα.

Στην Εικόνα 19 παρουσιάζεται ένα ακόμα παράδειγμα όπου στο κελί [3] δίνεται η εντολή εγκατάστασης της εξωτερικής βιβλιοθήκης σχεδίασης γραφημάτων matplotlib με την εντολή: `!pip install matplotlib`, ενώ στο κελί [4] παρατίθεται σύντομος κώδικας που εμφανίζει ένα γράφημα γραμμής.

```
[3]: !pip install matplotlib

     Requirement already satisfied: matplotlib in c:\users\cgogos\appdata\local\programs\python\python314\lib\site-package
     s (3.10.8)

[4]: import matplotlib.pyplot as plt

     values = [2, 4, 6, 8, 10]

     plt.figure(figsize=(2, 1.5))  # πλάτος x ύψος σε ίντσες
     plt.plot(values)
     plt.show()
```

Εικόνα 19 – Παράδειγμα σχεδίασης ενός γραφήματος με την εξωτερική βιβλιοθήκη matplotlib.

6.7. Ερωτήσεις κλειστού τύπου

1. Ποια από τις παρακάτω είναι built-in κλάση εξαίρεσης στην Python;
   A. `ReadError`
   B. `InputMismatchError`
   C. `ValueError`
   D. `FileMissingError`

2. Τι θα εμφανίσει ο παρακάτω κώδικας αν ο χρήστης δώσει ως είσοδο το κείμενο abc;

```
try:
    x = int(input())
    print("OK")
except ValueError:
    print("ERROR")
```

   A. Θα εμφανίσει OK
   B. Θα εμφανίσει ERROR
   C. Δεν θα εμφανίσει τίποτα
   D. Θα τερματίσει με μήνυμα σφάλματος χωρίς να εμφανίσει κείμενο

3. Πότε εκτελείται το μπλοκ `else` σε μια δομή `try/except/else`;
   A. Πάντα, ανεξάρτητα από το αν προέκυψε εξαίρεση ή όχι
   B. Μόνο αν συμβεί εξαίρεση
   C. Μόνο αν ΔΕΝ συμβεί εξαίρεση μέσα στο `try`
   D. Μόνο αν συμβεί εξαίρεση και δεν υπάρξει αντίστοιχο `except`

4. Ποια από τις παρακάτω είναι έγκυρη χρήση της assert με προσαρμοσμένο μήνυμα;
   A. `assert x > 0, "x must be positive"`
   B. `assert(x > 0: "x must be positive")`
   C. `assert(x > 0) "x must be positive"`
   D. `assert: x > 0, "x must be positive"`

5. Ποια είναι η λειτουργία της `strftime()`;
   A. Μετατρέπει `string` σε `datetime`
   B. Υπολογίζει διαφορά μεταξύ δύο ημερομηνιών
   C. Επιστρέφει `timestamp` σε δευτερόλεπτα
   D. Μετατρέπει αντικείμενο ημερομηνίας/ώρας σε μορφοποιημένο `string`

6.8. Ασκήσεις προς επίλυση

1. Γράψτε συνάρτηση `average(a, b)` που επιστρέφει τον μέσο όρο δύο αριθμών. Χρησιμοποιήστε την εντολή `assert` ώστε να διασφαλίζεται ότι και τα δύο ορίσματα της συνάρτησης είναι αριθμοί (`int` ή `float`), χρησιμοποιώντας τη συνάρτηση `isinstance()`.
2. Γράψτε πρόγραμμα που να ζητά το πλήθος βαθμών n που πρόκειται να καταχωρηθούν στη συνέχεια. Αν το n είναι μη θετικός ή αν δεν είναι ακέραιος αριθμός να εμφανίζει μήνυμα λάθους και να ζητά από τον χρήστη να καταχωρήσει εκ νέου την τιμή n. Στη συνέχεια για κάθε βαθμό να τον διαβάζει από το πληκτρολόγιο και να ελέγχει με `try/except` αν είναι πραγματικός αριθμός και να πραγματοποιεί `raise ValueError` αν ο βαθμός δεν ανήκει στο διάστημα [0, 10]. Αν δοθεί λάθος βαθμός να εμφανίζεται κατάλληλο μήνυμα

σφάλματος και ο χρήστης να ξαναδίνει βαθμό. Μετά την καταχώρηση όλων των βαθμών να εμφανίζει το μέσο όρο με δύο δεκαδικά ψηφία.

3. Για όλο το έτος 2026 εμφανίστε τις ημερομηνίες που είναι Κυριακή και βρίσκονται στις ημέρες από 10 μέχρι και 20 του μήνα. Ομοίως, υπολογίστε το πλήθος αυτών των ημερομηνιών για κάθε έτος από το 2000 μέχρι το 2100, χωρίς να εμφανίζετε τις ημερομηνίες. Εμφανίστε τα έτη με το μεγαλύτερο πλήθος τέτοιων ημερομηνιών.
4. Γράψτε πρόγραμμα που να λαμβάνει την τρέχουσα ημερομηνία και ώρα και να εμφανίζει κάθε 5 δευτερόλεπτα την ημερομηνία και την ώρα σε UTC, Europe/Athens, America/New_York, Asia/Tokyo και Australia/Sydney.
5. Γράψτε πρόγραμμα που να χρησιμοποιεί το module calendar και να ζητά από τον χρήστη να εισάγει ένα έτος και έναν μήνα (1–12), ελέγχοντας την εγκυρότητα της εισόδου, και να εμφανίζει το ημερολόγιο του συγκεκριμένου μήνα σε μορφή πίνακα, να υπολογίζει πόσες ημέρες έχει ο μήνας, ποια ημέρα της εβδομάδας αντιστοιχεί στην πρώτη ημέρα του μήνα και αν το έτος είναι δίσεκτο. Τέλος, να εμφανίζει ένα συνοπτικό μήνυμα με τα βασικά χαρακτηριστικά του επιλεγμένου μήνα που υπολογίστηκαν.
6. Να γράψετε πρόγραμμα που ο Η/Υ να επιλέγει τυχαία δύο ημερομηνίες και να τις εμφανίζει στον χρήστη, ζητώντας του να υπολογίσει πόσες ημέρες απέχουν μεταξύ τους. Ο χρήστης να έχει στη διάθεσή του χρονικό περιθώριο 5 δευτερολέπτων για να δώσει την απάντησή του, και αν απαντήσει εντός του χρόνου το πρόγραμμα να εμφανίζει πόσο απέχει η τιμή που εισήγαγε ο χρήστης από την πραγματική τιμή διαφοράς ημερομηνιών σε ημέρες. Αν ο χρόνος λήξει και ο χρήστης δεν έχει απαντήσει, να εμφανίζει μήνυμα «Δεν δόθηκε απάντηση εντός του χρόνου» και επίσης να εμφανίζει τη διαφορά ημερομηνιών σε ημέρες.

ΚΕΦΑΛΑΙΟ 7: ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

Ο αντικειμενοστραφής προγραμματισμός (OOP = Object Oriented Programming) είναι ένας τρόπος δόμησης λογισμικού που στοχεύει στη διαχείριση της πολυπλοκότητας που συνεπάγεται η συγγραφή μεγάλων και σύνθετων εφαρμογών. Ενώ στο διαδικασιακό προγραμματισμό (procedural programming) οι συναρτήσεις και οι δομές δεδομένων στις οποίες επενεργούν οι συναρτήσεις είναι ξεχωριστές οντότητες, στον OOP τα δεδομένα και οι συναρτήσεις που επενεργούν σε αυτά βρίσκονται μαζί, οπότε η πρόσβαση σε κάθε ομάδα δεδομένων επιτρέπεται μόνο στις συναρτήσεις που έχει σχεδιαστεί να έχουν πρόσβαση. Η μετάβαση από το διαδικασιακό προγραμματισμό στον OOP συνήθως βελτιώνει την επαναχρησιμοποίηση (reusability) του κώδικα καθιστώντας ευκολότερη τη συντήρηση και επέκταση του κώδικα.

Ο OOP είναι το κυρίαρχο προγραμματιστικό υπόδειγμα (programming paradigm) σήμερα. Πολλές σημαντικές γλώσσες προγραμματισμού όπως η Python, η C++, η Java και άλλες υποστηρίζουν (ή και επιβάλουν) τον αντικειμενοστραφή προγραμματισμό.

7.1. Κλάσεις και αντικείμενα

Οι κλάσεις αποτελούν βασική έννοια του OOP σε όλες τις αντικειμενοστραφείς γλώσσες προγραμματισμού. Μια κλάση είναι ένα σχέδιο (συχνά αναφέρεται ως blueprint ή template) που ορίζει:

- Δεδομένα με τη μορφή ιδιοτήτων (properties) ή αλλιώς χαρακτηριστικών (attributes).
- Συμπεριφορά με τη μορφή μεθόδων (methods).

Συνεπώς, μια κλάση καθορίζει τις ιδιότητες που θα έχει ένα αντικείμενο και ορίζουν την κατάσταση (state) του αντικειμένου και τις λειτουργίες που θα μπορεί να εκτελέσει, δηλαδή την συμπεριφορά του, αλλά η ίδια η κλάση δεν συνιστά μια συγκεκριμένη οντότητα από αυτό που περιγράφει, είναι απλά το σχέδιό της.

Στην Python μια κλάση ορίζεται με τη δεσμευμένη λέξη `class`. Για παράδειγμα, στον κώδικα Κ. 208 ορίζεται η κλάση `Student` με χαρακτηριστικά `name` και `grade` και συμπεριφορά τη μέθοδο με όνομα `passed` που όταν θα καλείται για ένα αντικείμενο `Student` θα επιστρέφει `True` αν ο βαθμός του σπουδαστή είναι μεγαλύτερος ή ίσος του 5, αλλιώς θα επιστρέφει `False`. Από την άλλη πλευρά, ένα αντικείμενο (object) είναι ένα στιγμιότυπο (instance) μιας κλάσης που

αναπαριστά μια συγκεκριμένη οντότητα με τα δικά της δεδομένα για την οποία μπορούν να κληθούν οι μέθοδοι που ορίζονται στην κλάση. Στον κώδικα Κ. 208, πέρα από τον ορισμό της κλάσης, δημιουργούνται δύο αντικείμενα και καλείται για κάθε αντικείμενο η μέθοδος `passed()`.

```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def passed(self):
        return self.grade >= 5


s1 = Student("Μαρία", 7)
s2 = Student("Νίκος", 4)

s1.passed()  # True
s2.passed()  # False

print(
    f"Όνομα: {s1.name}, βαθμός: {s1.grade}, προβιβάζεται: {s1.passed()}"
)  # Όνομα: Μαρία, βαθμός: 7, προβιβάζεται: True
print(
    f"Όνομα: {s2.name}, βαθμός: {s2.grade}, προβιβάζεται: {s2.passed()}"
)  # Όνομα: Νίκος, βαθμός: 4, προβιβάζεται: False
```

Κ. 208 – Ορισμός της κλάσης Student, δημιουργία 2 αντικειμένων Student και κλήση της μεθόδου passed().

Αξίζει να σημειωθεί ότι στην Python τα πάντα είναι αντικείμενα (π.χ. ακέραιοι, λεκτικά, λίστες, λεξικά, συναρτήσεις) και ότι οι μέθοδοι που ορίζονται σε μια κλάση λαμβάνουν το αντικείμενο για το οποίο καλούνται με την πρώτη παράμετρό τους που η σύμβαση είναι να ονομάζεται `self`.

7.1.1. Σταδιακή δημιουργία μιας κλάσης και αντικειμένων της στο REPL

Η Python λόγω της δυναμικής της φύσης επιτρέπει την προσθήκη και αφαίρεση κατά το χρόνο εκτέλεσης ιδιοτήτων και μεθόδων στις κλάσεις. Στη συνέχεια θα παρουσιαστεί ένα παράδειγμα σταδιακής δημιουργίας μια κλάσης και αντικειμένων της στο REPL.

Αρχικά ενεργοποιείται το REPL και δημιουργείται σε αυτό μια κλάση με όνομα Rectangle χωρίς καθόλου περιεχόμενο.

```
Python 3.14.2 (main, Dec  5 2025, 16:49:16) [Clang 17.0.0 (clang-1700.4.4.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> class Rectangle:
...     pass
...
>>>
```

Στη συνέχεια, δημιουργείται ένα στιγμιότυπο της κλάσης Rectangle, το αντικείμενο r1.

```
>>> r1 = Rectangle()
```

Μετά, προστίθενται οι ιδιότητες width και height στο αντικείμενο r1.

```
>>> r1.width, r1.height = 4, 5
```

Μετά, προστίθεται στην κλάση μια μέθοδος area() που υπολογίζει και επιστρέφει το εμβαδόν ορθογωνίου για τα στιγμιότυπά της.

```
>>> def area(self):
...     return self.width * self.height
...
>>> Rectangle.area = area
```

Τέλος, εκτυπώνονται οι τιμές των ιδιοτήτων του αντικειμένου r1 και καλείται σε αυτό η μέθοδος area(). Το ίδιο συμβαίνει και για ένα ακόμη αντικείμενο, το r2.

```
>>> r1.width
4
>>> r1.height
5
>>> r1.area()
20
>>> r2 = Rectangle()
>>> r2.width, r2.height = 2, 7
>>> r2.area()
14
```

Θα πρέπει να σημειωθεί ότι ο παραπάνω δεν είναι ο τυπικός τρόπος δημιουργίας κλάσεων και αντικειμένων αλλά επιδεικνύει τη δυναμική φύση προγραμματισμού που υποστηρίζει η Python. Στη επόμενη παράγραφο θα παρουσιαστεί η τυπική δημιουργία της ίδιας κλάσης μέσα σε ένα αρχείο πηγαίου κώδικα Python (.py), κατ' αναλογία με το παράδειγμα που παρουσιάστηκε στην παράγραφο 7.1.1.

7.1.2. Δημιουργία κλάσης και αντικειμένων σε πρόγραμμα

Στον κώδικα Κ. 209 ορίζεται η κλάση Rectangle και δημιουργούνται δύο αντικείμενά της. Ο κώδικας αυτός τυπικά βρίσκεται σε ένα αρχείο πηγαίου κώδικα Python, π.χ., rectangle.py. Η μέθοδος __init__ είναι μια μέθοδος αρχικοποίησης που εκτελείται αυτόματα όταν δημιουργείται ένα νέο στιγμιότυπο μιας κλάσης όπως για παράδειγμα συμβαίνει με τις εντολές r1 = Rectangle(4, 5) και r2 = Rectangle(2, 7) για τη δημιουργία των αντικειμένων r1 και r2 αντίστοιχα. Γενικά, η μέθοδος __init__ δέχεται τουλάχιστον μια παράμετρο, την self που αναφέρεται στο νέο αντικείμενο που δημιουργείται (τα r1 και r2 στο παράδειγμα) και δεν επιστρέφει το αντικείμενο που δημιουργείται, καθώς επιστρέφει None, αλλά απλά το αρχικοποιεί.

Η δε μέθοδος `area()` της κλάσης `Rectangle` όπως και κάθε άλλη μέθοδος οποιασδήποτε άλλης κλάσης έχει ως πρώτη παράμετρο την παράμετρο `self` που αντιστοιχεί στο αντικείμενο για το οποίο γίνεται η κλήση της μεθόδου.

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height


r1 = Rectangle(4, 5)
print(
    f"Ορθογώνιο 1: {r1.width}x{r1.height}, εμβαδόν = {r1.area()}"
)  # Ορθογώνιο 1: 4x5, εμβαδόν = 20
r2 = Rectangle(2, 7)
print(
    f"Rectangle 2: {r2.width}x{r2.height}, εμβαδόν = {r2.area()}"
)  # Ορθογώνιο 2: 2x7, εμβαδόν = 14
```

Κ. 209 – Η κλάση Rectangle.

7.1.3. Ένα ακόμα παράδειγμα δημιουργίας κλάσης και στιγμιοτύπων της

Στην παράγραφο αυτή παρουσιάζεται ένα ακόμα παράδειγμα δημιουργίας μιας κλάσης και αντικειμένων της. Η κλάση αφορά έναν ψηφιακό μετρητή (`DigitalCounter`) που ξεκινά από το μηδέν και επιστρέφει στο μηδέν όταν ξεπεράσει μια μέγιστη τιμή. Ο κώδικας έχει διαμεριστεί σε δύο αρχεία που βρίσκονται στον ίδιο φάκελο, το αρχείο `dc.py` που περιέχει τον ορισμό της κλάσης και το αρχείο `main_dc.py` όπου γίνεται import η κλάση του ψηφιακού μετρητή, δημιουργούνται αντικείμενα ψηφιακού μετρητή και καλούνται μέθοδοί τους.

Στον κώδικα Κ. 210 δημιουργείται η κλάση `DigitalCounter` με χαρακτηριστικά την τιμή του μετρητή (`count`) και τη μέγιστη τιμή μετρητή (`max_value`), καθώς και τις ακόλουθες μεθόδους:

- Μέθοδο `set_max()` που θέτει μέγιστη τιμή στο μετρητή.
- Μέθοδο `increment()` που αυξάνει τον μετρητή κατά ένα.
- Μέθοδο `clear()` που μηδενίζει τον μετρητή.

```
class DigitalCounter:
    def __init__(self, max_value=10):
        self.count = 0
        self.max_value = max_value
```

```python
    def set_max(self, value):
        self.max_value = value

    def increment(self):
        self.count += 1
        if self.count > self.max_value:
            self.count = 0

    def clear(self):
        self.count = 0
```
Κ. 210 – Ορισμός της κλάσης DigitalCounter (αρχείο: dc.py).

Στον κώδικα Κ. 211 δημιουργούνται δύο αντικείμενα μετρητές και καλούνται οι μέθοδοί τους. Αξίζει να σημειωθεί ότι στις εντολές επανάληψης for του συγκεκριμένου κώδικα χρησιμοποιείται η κάτω παύλα _ αντί για κανονικό όνομα μεταβλητής. Η σύμβαση αυτή δηλώνει ότι η μεταβλητή του βρόχου δεν αξιοποιείται στο σώμα της επανάληψης· πρόκειται δηλαδή για «αδιάφορη» (throwaway) μεταβλητή, της οποίας η τιμή δεν έχει σημασία και χρησιμοποιείται μόνο για τον έλεγχο του πλήθους επαναλήψεων.

```python
from dc import DigitalCounter

counter1 = DigitalCounter()
counter1.set_max(5)
print(f"{counter1.count=}")  # 0
for _ in range(7):
    counter1.increment()
    print(f"{counter1.count=}")  # 1, 2, 3, 4, 5, 0, 1
counter1.clear()
print(f"{counter1.count=}")  # 0

print("#" * 20)

counter2 = DigitalCounter()
counter2.set_max(2)
print(f"{counter2.count=}")  # 0
for _ in range(5):
    counter2.increment()
    print(f"{counter2.count=}")  # 1, 2, 0, 1, 2, 0
counter2.clear()
print(f"{counter2.count=}")  # 0
```
Κ. 211 – Κύριο αρχείο εκτέλεσης για τη δημιουργία αντικειμένων DigitalCounter (αρχείο: main_dc.py).

Η έξοδος της εκτέλεσης του κώδικα (εκτέλεση του αρχείου main_dc.py) παρουσιάζεται στο Ε. 55.

```
counter1.count=0
counter1.count=1
```

```
counter1.count=2
counter1.count=3
counter1.count=4
counter1.count=5
counter1.count=0
counter1.count=1
counter1.count=0
###################
counter2.count=0
counter2.count=1
counter2.count=2
counter2.count=0
counter2.count=1
counter2.count=2
counter2.count=0
```

Ε. 55 – Αποτέλεσμα εκτέλεσης του αρχείου main_dc.py.

7.2. Dunder μέθοδοι αντικειμένων

Οι μέθοδοι που το όνομά τους ξεκινά και τελειώνει με 2 κάτω παύλες ονομάζονται dunder (double underscore) μέθοδοι ή αλλιώς magic μέθοδοι. Οι dunder μέθοδοι επιτρέπουν να προσδιοριστεί κατάλληλα η συμπεριφορά ενσωματωμένων (built-in) λειτουργιών, όπως η δημιουργία αντικειμένων, η εμφάνιση αντικειμένων όταν εκτυπώνονται κ.α. Μια dunder μέθοδος που ήδη αναφέρθηκε ήδη είναι η `__init__` αλλά υπάρχουν και άλλες όπως οι ακόλουθες:

- Η μέθοδος `__str__` που επιστρέφει μια συμβολοσειρά που εμφανίζεται όταν εκτυπώνεται το αντικείμενο.
- Η μέθοδος `__repr__` που είναι παρόμοια με την `__str__`, αλλά χρησιμοποιείται για λόγους αποσφαλμάτωσης (debugging), και επιστρέφει ένα λεκτικό που μπορεί να χρησιμοποιηθεί για τη δημιουργία του αντικειμένου.
- Η μέθοδος `__del__` που καλείται είτε ρητά (explicitly) είτε αυτόματα ή αλλιώς υπονοούμενα (implicitly) που συμβαίνει όταν καταστρέφεται το αντικείμενο, δηλαδή όταν δεν υπάρχουν πλέον μεταβλητές που να αποτελούν αναφορές σε αυτό.
- Η μέθοδος `__add__` που καθορίζει τη συμπεριφορά του τελεστή + για αντικείμενα μιας κλάσης, επιτρέπει δηλαδή να καθορίσουμε τι σημαίνει πρόσθεση για τα δικά μας αντικείμενα (αντίστοιχα υπάρχουν οι `__sub__`, `__mul__`, `__truediv__` για αφαίρεση, πολλαπλασιασμό και διαίρεση καθώς και πολλές άλλες).
- Η μέθοδος `__call__` που επιτρέπει σε ένα αντικείμενο να κληθεί όπως θα καλούσαμε μια συνάρτηση, δηλαδή όταν χρησιμοποιούμε παρενθέσεις ( ) σε ένα αντικείμενο η Python καλεί την `__call__` μέθοδο της κλάσης του αντικειμένου.

- Η μέθοδος __getitem__ που επιτρέπει στα αντικείμενα μιας κλάσης του χρήστη να χρησιμοποιούν τις αγκύλες, όπως χρησιμοποιούνται για παράδειγμα στις λίστες και στα λεξικά.

7.2.1. Παράδειγμα με __str__ και __repr__

Οι μέθοδοι __str__ και __repr__ έχουν παρόμοια χαρακτηριστικά. Από τη μια μεριά η __str__ στοχεύει στους χρήστες του προγράμματος και παρουσιάζει μια μορφή εκτύπωσης αντικειμένου που είναι εύκολα αναγνώσιμη και έχει ωραία μορφοποίηση. Από την άλλη μεριά η __repr__ στοχεύει στους προγραμματιστές και παρουσιάζει μια μορφή εκτύπωσης αντικειμένου που είναι χρήσιμη στην αποσφαλμάτωση του κώδικα και ιδανικά μπορεί να χρησιμοποιηθεί για να δημιουργηθεί από αυτή το αντικείμενο.

Στο παράδειγμα του κώδικα Κ. 212 ορίζεται η κλάση Book και με τις δύο μεθόδους, τόσο την __str__, όσο και την __repr__. Αν δεν υπήρχε ορισμός της __str__ τότε η εκτύπωση του αντικειμένου με την print θα γινόταν χρησιμοποιώντας την __repr__. Αν δεν υπήρχαν και οι δύο μέθοδοι τότε θα εμφανίζονταν μια τυπική μορφή απεικόνισης του αντικειμένου της μορφής: <__main__.Book object at 0x101597650>.

Στην περίπτωση της __repr__ μεταβλητές κειμένου όπως το title και το author συμπεριλαμβάνονται στο f-string με τον συντελεστή μετατροπής !r που τοποθετεί εισαγωγικά στην αρχή και στο τέλος του κειμένου.

```python
class Book:
    def __init__(self, title, author, price):
        self.title = title
        self.author = author
        self.price = price

    def __str__(self):
        return f"'{self.title}' από τον {self.author}"

    def __repr__(self):
        return (
            f"Book(title={self.title!r}, author={self.author!r}, price={self.price!r})"
        )


b = Book(title="1984", author="George Orwell", price=9.99)

print(b)  # '1984' από George Orwell
print(str(b))  # '1984' από George Orwell
```

```
print(repr(b))  # Book(title='1984', author='George Orwell', price=9.99)
```
Κ. 212 – Η κλάση Book με υλοποιήσεις των μεθόδων __str__ και __repr__.

7.2.2. Παράδειγμα με την μέθοδο `__del__`

Η μέθοδος `__del__` είναι ένας καταστροφέας (destructor), δηλαδή καλείται αυτόματα για ένα αντικείμενο όταν το αντικείμενο πρόκειται να καταστραφεί, δηλαδή όταν δεν υπάρχουν πλέον αναφορές σε αυτό. Στον κώδικα Κ. 213 υπάρχει ο ορισμός της κλάσης `Connection` με κάθε στιγμιότυπο της κλάσης να λαμβάνει έναν μοναδικό κωδικό σύνδεσης, μέσω του module `uuid`. Το uuid είναι ένα module της τυπικής βιβλιοθήκης της Python που χρησιμοποιείται για τη δημιουργία καθολικά μοναδικών αναγνωριστικών (Universally Unique Identifiers – UUIDs). Στη συνάρτηση `demo()`, που βρίσκεται εκτός της κλάσης `Connection` δημιουργούνται τρεις συνδέσεις με κάποια υποθετική υπηρεσία με κάθε σύνδεση να λαμβάνει έναν μοναδικό κωδικό. Παρατηρήστε ότι η `__del__` καλείται είτε ρητά με την εντολή `del c2`, είτε αυτόματα για τις δύο άλλες συνδέσεις όταν οι μεταβλητές `c1` και `c2` βγαίνουν εκτός εμβέλειας, δηλαδή όταν ολοκληρωθεί η εκτέλεση της συνάρτησης `demo()`.

```
import uuid


class Connection:
    def __init__(self):
        self.code = str(uuid.uuid4()).split("-")[0]
        print(f"Η σύνδεση {self.code} άνοιξε.")

    def __del__(self):
        print(f"Η σύνδεση {self.code} έκλεισε.")

    def __str__(self):
        return self.code


def demo():
    c1 = Connection()
    c2 = Connection()
    c3 = Connection()
    print(f"Εκτελείται εργασία στις συνδέσεις {c1}, {c2}, {c3}")
    tmp = c2.code
    del c2
    print(f"Η σύνδεση {tmp} διαγράφηκε.")


demo()
print("Η λειτουργία demo ολοκληρώθηκε.")
```

Κ. 213 – Παράδειγμα ρητής κλήσης και αυτόματων κλήσεων της μεθόδου __del__.

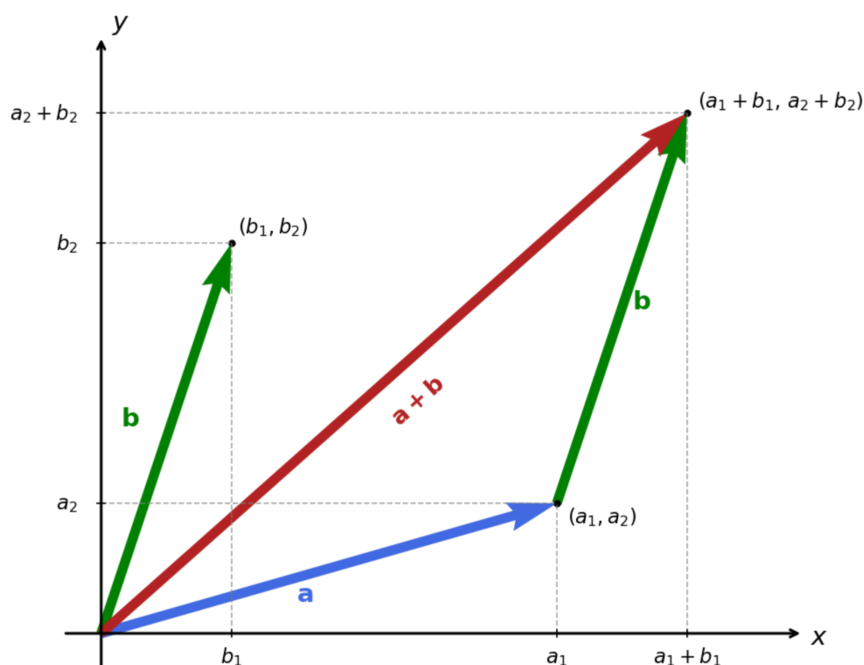
Η έξοδος από μια εκτέλεση του κώδικα Κ. 213 φαίνεται στη συνέχεια (Ε. 56).

```
Η σύνδεση 4a63193b άνοιξε.
Η σύνδεση 9035fff6 άνοιξε.
Η σύνδεση 987a15b9 άνοιξε.
Εκτελείται εργασία στις συνδέσεις 4a63193b, 9035fff6, 987a15b9
Η σύνδεση 9035fff6 έκλεισε.
Η σύνδεση 9035fff6 διαγράφηκε.
Η σύνδεση 987a15b9 έκλεισε.
Η σύνδεση 4a63193b έκλεισε.
Η λειτουργία demo ολοκληρώθηκε.
```

Ε. 56 – Ένα παράδειγμα εξόδου από την εκτέλεση του κώδικα Κ. 213.

7.2.3. Παράδειγμα με την μέθοδο __add__

Η μέθοδος __add__ επιτρέπει την υπερφόρτωση του τελεστή + για αντικείμενα νέων κλάσεων έτσι ώστε η χρήση των αντικειμένων των κλάσεων αυτών να είναι διαισθητικά ευκολότερη. Στο παράδειγμα που ακολουθεί, στον κώδικα Κ. 214, ορίζεται μια κλάση με όνομα `Vector` που αναπαριστά ένα διάνυσμα. Τα διανύσματα είναι βασικά μαθηματικά αντικείμενα και η πρόσθεση διανυσμάτων ορίζεται όπως φαίνεται στην Εικόνα 20. Η μέθοδος __add__ επιτρέπει η πρόσθεση αντικειμένων διανυσμάτων να γίνεται με τον τελεστή +.



Εικόνα 20 – Γεωμετρική αναπαράσταση πρόσθεσης διανυσμάτων.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        if isinstance(other, Vector):
            return Vector(self.x + other.x, self.y + other.y)
        return NotImplemented

    def __repr__(self):
        return f"Vector({self.x}, {self.y})"


v1 = Vector(3, 4)
v2 = Vector(1, 2)
v3 = v1 + v2
print(v3)  # Vector(4, 6)
```
Κ. 214 – Υπερφόρτωση του τελεστή + για αντικείμενα της κλάσης Vector.

Παρατηρήστε τη χρήση της μεθόδου `isinstance` και της επιστρεφόμενης τιμής `NotImplemented` στην `__add__`. Ο συνδυασμός αυτός επιτρέπει την ορθή και επεκτάσιμη υλοποίηση του τελεστή +, καθώς δηλώνει ρητά ότι η κλάση μπορεί να χειριστεί την πρόσθεση μόνο με αντικείμενα συμβατού τύπου, ενώ για κάθε άλλο τύπο αναθέτει την επίλυση της πράξης στον μηχανισμό επίλυσης τελεστών της Python. Αυτός ο σχεδιασμός διασφαλίζει σωστή σημασιολογία τελεστών, υποστηρίζει τη συνεργασία μεταξύ διαφορετικών κλάσεων και αποτρέπει λανθασμένη συμπεριφορά του τελεστή +.

7.2.4. Παράδειγμα με την μέθοδο `__call__`

Η μέθοδος `__call__` επιτρέπει σε ένα αντικείμενο κλάσης να καλείται σαν συνάρτηση. Ειδικότερα, όταν ένα αντικείμενο `obj` υλοποιεί την `__call__` τότε η έκφραση `obj(…)` μεταφράζεται εσωτερικά σε `obj.__call__(...)`. Στον κώδικα Κ. 215 – Παράδειγμα υλοποίησης της μεθόδου __call__. παρουσιάζεται ένα παράδειγμα υλοποίησης της μεθόδου `__call__` στην κλάση `Multiplier` που αποθηκεύει έναν συντελεστή πολλαπλασιασμού `factor` σε κάθε αντικείμενό της. Τα δύο αντικείμενα που δημιουργούνται στη συνέχεια, το `double` και το `triple` με συντελεστές 2 και 3 αντίστοιχα, μπορούν να καλούνται ως συναρτήσεις που δέχεται ένα όρισμα και να πραγματοποιούν πολλαπλασιασμούς του ορίσματος που δέχονται με το 2 και το 3 αντίστοιχα.

```
class Multiplier:
    def __init__(self, factor):
        self.factor = factor
```

```
    def __call__(self, value):
        return value * self.factor


double = Multiplier(2)
triple = Multiplier(3)
print(double(5))  # 10
print(double(100))  # 200
print(triple(5))  # 15
print(triple(100))  # 300
```
Κ. 215 – Παράδειγμα υλοποίησης της μεθόδου __call__.

7.2.5. Παράδειγμα με την μέθοδο `__getitem__`

Η μέθοδος `__getitem__` επιτρέπει σε ένα αντικείμενο `obj` να χρησιμοποιεί το συντακτικό πρόσβασης τύπου δείκτη, δηλαδή `obj[key]`, όπου `key` είναι ένας ακέραιος, λεκτικό, slice ή οποιοδήποτε αντικείμενο ανάλογα με το σχεδιασμό της κλάσης. Στην πράξη η `__getitem__` χρησιμοποιείται για να προσομοιωθεί η συμπεριφορά ενός container όπως τα λεξικά και οι λίστες από τα αντικείμενα μιας νέας κλάσης. Για παράδειγμα στο κώδικα Κ. 216, ορίζεται η κλάση `GreekDictionary` που αποθηκεύει ένα λεξικό με αγγλικές λέξεις και τι ελληνικές μεταφράσεις τους. Η `__getitem__` επιτρέπει την πρόσβαση στο αποθηκευμένο λεξικό χρησιμοποιώντας απευθείας πάνω σε ένα αντικείμενο της κλάσης `GreekDictionary` αγκύλες. Έτσι, αντί για `dictionary.words['hello']` μπορούμε να γράψουμε `dictionary['hello']` και να λάβουμε το ίδιο αποτέλεσμα.

```
class GreekDictionary:
    def __init__(self):
        self.words = {
            "hello": "γεια σου",
            "thank you": "ευχαριστώ",
            "goodbye": "αντίο"
        }

    def __getitem__(self, english_word):
        return self.words.get(english_word, "Δεν βρέθηκε")


dictionary = GreekDictionary()
print(dictionary["hello"])  # γεια σου
print(dictionary["thank you"])  # ευχαριστώ
print(dictionary["water"])  # Δεν βρέθηκε
```
Κ. 216 – Παράδειγμα υλοποίησης της μεθόδου __getitem__.

7.2.6. Άσκηση 1

Να υλοποιήσετε μια κλάση Book, η οποία θα περιγράφει ένα βιβλίο με πεδία `title`, `isbn` και `ratings` (λίστα με βαθμολογίες, αρχικά κενή). Η κλάση θα πρέπει να περιλαμβάνει μέθοδο `add_rating(score)` που προσθέτει μια βαθμολογία από 1 έως 5, μέθοδο `average_rating()` που υπολογίζει και επιστρέφει το μέσο όρο των βαθμολογιών (ή μήνυμα αν δεν υπάρχουν βαθμολογίες), καθώς και μέθοδο `__str__` που επιστρέφει σε μορφή κειμένου τον τίτλο, το ISBN και τη μέση βαθμολογία του βιβλίου.

Η λύση της άσκησης βρίσκεται στο Παράρτημα Α'.

7.3. Επίλυση προβλήματος με διαδικασιακό προγραμματισμό και με αντικειμενοστραφή προγραμματισμό

Σε αυτό το σημείο, που έχουν ήδη αναφερθεί κάποιες βασικές έννοιες περί αντικειμενοστραφούς προγραμματισμού, θα παρουσιαστεί η υλοποίηση της λύσης ενός προβλήματος πρώτα με διαδικασιακό προγραμματισμό και στη συνέχεια με αντικειμενοστραφή προγραμματισμό έτσι ώστε να γίνει αποκτηθεί καλύτερη αντίληψη για τη διαφορά ανάμεσα στις δύο προσεγγίσεις που ωστόσο στο συγκεκριμένο παράδειγμα θα παράγουν το ίδιο αποτέλεσμα. Το πρόβλημα αφορά ορθογώνια παραλληλόγραμμα για τα οποία δίνονται οι διαστάσεις τους (πλάτος και ύψος) και ζητείται ο υπολογισμός μεγεθών όπως το εμβαδόν και η περίμετρος τους καθώς και το αν το ορθογώνιο είναι τετράγωνο ή όχι.

Στον κώδικα Κ. 217 παρουσιάζεται η διαδικασιακή προσέγγιση όπου ορίζονται τρεις ανεξάρτητες συναρτήσεις για τον υπολογισμό εμβαδού, περιμέτρου και έλεγχο αν το ορθογώνιο είναι τετράγωνο. Οι δε μεταβλητές που αποθηκεύουν τις διαστάσεις των ορθογωνίων είναι απλές μεταβλητές.

```
def area(width, height):
    return width * height


def perimeter(width, height):
    return 2 * width + 2 * height


def is_square(width, height):
    return width == height


w1, h1 = 4, 5
```

```
a1 = area(w1, h1)
p1 = perimeter(w1, h1)
s1 = is_square(w1, h1)
print(f"Ορθογώνιο με πλάτος {w1} και ύψος {h1}")  # Ορθογώνιο με πλάτος 4 και ύψος 5
print(
    f"Εμβαδόν:{a1} περίμετρος:{p1} τετράγωνο:{s1}"
)  # Εμβαδόν:20 περίμετρος:18 τετράγωνο:False

w2, h2 = 10, 10
a2 = area(w2, h2)
p2 = perimeter(w2, h2)
s2 = is_square(w2, h2)
print(f"Ορθογώνιο με πλάτος {w2} και ύψος {h2}")  # Ορθογώνιο με πλάτος 10 και ύψος 10
print(
    f"Εμβαδόν:{a2} περίμετρος:{p2} τετράγωνο:{s2}"
)  # Εμβαδόν:100 περίμετρος:40 τετράγωνο:True
```

Κ. 217 – Προσέγγιση διαδικασιακού προγραμματισμού για επίλυση προβλήματος με ορθογώνια παραλληλόγραμμα

Στον κώδικα Κ. 218 παρουσιάζεται η αντικειμενοστραφής προσέγγιση για το ίδιο πρόβλημα. Στην προσέγγιση αυτή ορίζεται η κλάση `Rectangle` με χαρακτηριστικά `width` και `height` και μεθόδους που υπολογίζουν το εμβαδόν, και την περίμετρο και εξετάζουν αν το ορθογώνιο είναι τετράγωνο. Επίσης, ορίζεται η μέθοδος `__str__` που εμφανίζει ένα κείμενο με τις τιμές πλάτους και ύψους για το αντικείμενο ορθογώνιο για εύκολη εμφάνιση αντικειμένων ορθογωνίων.

```
class Rectangle:
    def __init__(self, w, h):
        self.width = w
        self.height = h

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * self.width + 2 * self.height

    def is_square(self):
        return self.width == self.height:

    def __str__(self):
        return f"Ορθογώνιο με πλάτος {self.width} και ύψος {self.height}"


r1 = Rectangle(4, 5)
print(r1)  # Ορθογώνιο με πλάτος 4 και ύψος 5
print(
    f"Εμβαδόν:{r1.area()} περίμετρος:{r1.perimeter()} τετράγωνο:{r1.is_square()}"
```

```
)  # Εμβαδόν:20 περίμετρος:18 τετράγωνο:False


r2 = Rectangle(10, 10)
print(r2)  # Ορθογώνιο με πλάτος 10 και ύψος 10
print(
    f"Εμβαδόν:{r2.area()} περίμετρος:{r2.perimeter()} τετράγωνο:{r2.is_square()}"
)  # Εμβαδόν:100 περίμετρος:40 τετράγωνο:True
```
Κ. 218 – Προσέγγιση αντικειμενοστραφούς προγραμματισμού για επίλυση προβλήματος με ορθογώνια παραλληλόγραμμα.

Και στις δύο περιπτώσεις η έξοδος που παράγεται κατά την εκτέλεση του κώδικα είναι η ίδια όπως φαίνεται στα σχόλια που έχουν συμπεριληφθεί στους κώδικες. Ωστόσο, στην περίπτωση της αντικειμενοστραφούς υλοποίησης ο κώδικας θεωρείται ότι είναι πιο αναγνώσιμος και συντηρήσιμος καθώς κάθε μέθοδος εκφράζει ξεκάθαρα τι κάνει σε ποιο αντικείμενο, ενώ στην διαδικασιακή υλοποίηση η λογική είναι διάσπαρτη και η κατανόηση της συνολικής συμπεριφοράς προϋποθέτει την ιχνηλάτηση της ροής εκτέλεσης διαμέσου των συναρτήσεων.

7.4. Σύγκριση αντικειμένων

Η σύγκριση δύο αντικειμένων για ισότητα (με τον τελεστή ==) ελέγχει αν πρόκειται για το ίδιο αντικείμενο καθώς η προκαθορισμένη υλοποίηση είναι η υλοποίηση του τελεστή is. Ωστόσο, οι κλάσεις μπορούν να ορίσουν τη μέθοδο __eq__ για να καθορίσουν τι σημαίνει η ισότητα. Αντίστοιχα για τους τελεστές !=, <, <=, >, >= υπάρχουν οι μέθοδοι __ne__, __lt__, __le__, __gt__ και __ge__.

Στο παράδειγμα του κώδικα Κ. 219 ορίζεται η κλάση Point που αφορά ένα σημείο στο καρτεσιανό επίπεδο (κάθε σημείο διαθέτει συντεταγμένες x και y) και προστίθεται μέσω της μεθόδου __eq__ η δυνατότητα ελέγχου ισότητας δύο σημείων με τον τελεστή ==, όπου τα σημεία θα θεωρούνται ίσα αν έχουν τις ίδιες συντεταγμένες x και y. Αν δεν υπήρχε η υλοποίηση της μεθόδου __eq__ στην κλάση Point τότε η σύγκριση p1 == p2 θα επέστρεφε False.

```
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __eq__(self, other):
        if not isinstance(other, Point):
            return NotImplemented
        return self.x == other.x and self.y == other.y

    def __repr__(self):
```

```
        return f"Point({self.x}, {self.y})"


p1 = Point(2, 3)
p2 = Point(2, 3)
print(p1)  # Point(2, 3)
print(p1 == p2)  # True
```
Κ. 219 – Σύγκριση αντικειμένων Point με τον τελεστή ==.

7.4.1. Εισαγωγή αντικειμένων σε σύνολα και χρήση αντικειμένων ως κλειδιά λεξικών

Ορισμένα αντικείμενα είναι hashable που σημαίνει ότι υπάρχει υλοποίηση της μεθόδου __hash__ που επιστρέφει έναν ακέραιο αριθμό που παραμένει σταθερός καθόλη τη διάρκεια ζωής του αντικειμένου και είναι συνεπής με την __eq__. Η συνέπεια σε αυτή την περίπτωση σημαίνει ότι αν δύο αντικείμενα θεωρούνται ίσα σύμφωνα με τη μέθοδο __eq__, τότε πρέπει υποχρεωτικά να έχουν την ίδια τιμή __hash__. Δηλαδή, αν για δύο αντικείμενα a και b ισχύει `a == b`, τότε πρέπει να ισχύει και `hash(a) == hash(b)`. Η απαίτηση αυτή είναι κρίσιμη για τη σωστή λειτουργία δομών όπως τα dict και set, που πρώτα χρησιμοποιούν το hash για να εντοπίσουν το κατάλληλο "bucket" και στη συνέχεια την __eq__ για να ελέγξουν την πραγματική ισότητα.

Το παράδειγμα που ακολουθεί στον κώδικα Κ. 220 είναι μια επέκταση του κώδικα Κ. 219, όπου έχει προστεθεί η μέθοδος __hash__ και στη συνέχεια δημιουργούνται δύο αντικείμενα `Point` με τις ίδιες συντεταγμένες (2, 3) και τα σημεία αυτά εισάγονται σε ένα σύνολο. Καθώς τα αντικείμενα αυτά είναι ίσα, μετά την εισαγωγή τους στο σύνολο, στο σύνολο υπάρχει μόνο ένα αντικείμενο.

```
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __eq__(self, other):
        if not isinstance(other, Point):
            return NotImplemented
        return self.x == other.x and self.y == other.y

    def __hash__(self):
        return hash((self.x, self.y))

    def __repr__(self):
        return f"Point({self.x}, {self.y})"


points = set()
points.add(Point(2, 3))
points.add(Point(2, 3))
```

```
print(points) # {Point(2, 3)}
```
Κ. 220 – Εισαγωγή αντικειμένων Point σε σύνολο.

Αν δεν υπήρχε στην κλάση `Point` υλοποίηση της μεθόδου __hash__ τότε η απόπειρα εισαγωγής αντικειμένου `Point` στο σύνολο `points` θα αποτύγχανε και θα επέστρεφε ένα μήνυμα σφάλματος (`TypeError`) της μορφής:

```
Traceback (most recent call last):
  File "xxx/point2.py", line 18, in <module>
    points.add(Point(2, 3))
TypeError: unhashable type: 'Point'
```

7.5. Αντιγραφή αντικειμένων (ρηχή και βαθιά αντιγραφή)

Η αντιγραφή αντικειμένων είναι ένα θέμα που απαιτεί προσοχή. Η απλή ανάθεση μιας μεταβλητής που είναι αναφορά σε ένα αντικείμενο σε μια άλλη μεταβλητή, δεν πραγματοποιεί αντιγραφή, αλλά δημιουργεί δύο αναφορές προς το ίδιο αντικείμενο. Στο παράδειγμα που ακολουθεί και το οποίο εκτελείται στο REPL δημιουργείται αρχικά ένα λεξικό που ανατίθεται στην μεταβλητή `st1`. Η ανάθεση της `st1` στη νέα μεταβλητή `st2` με τον τελεστή = δεν δημιουργεί ένα νέο αντίγραφο του λεξικού, αλλά μια νέα αναφορά προς το ήδη υπάρχον αντικείμενο. Συνεπώς, αν επιχειρηθεί αλλαγή ενός στοιχείου του λεξικού όπως του αποθηκευμένου ονόματος, οι αλλαγές θα είναι ορατές και μέσω της μεταβλητής `st1` και μέσω της μεταβλητής `st2`.

```
>>> st1 = {'name': 'Nikos', 'grades': [7.0, 8.5, 6.5]}
>>> st2 = st1
>>> st1['name'] = 'Nikolaos'
>>> st1
{'name': 'Nikolaos', 'grades': [8.5, 6.5, 7.0]}
>>> st2
{'name': 'Nikolaos', 'grades': [8.5, 6.5, 7.0]}
```

Για να γίνει πραγματική αντιγραφή πρέπει να χρησιμοποιηθεί το module `copy`. Η συνάρτηση `copy.copy()` δημιουργεί ένα νέο αντικείμενο και πραγματοποιεί αντιγραφή για απλά (μη σύνθετα) αντικείμενα. Έτσι, στο προηγούμενο παράδειγμα η συμπεριφορά μπορεί να είναι διαφορετική, να γίνεται δηλαδή αντιγραφή, αν αντί για `st2 = st1` γίνει πρώτα import το module `copy` και στη συνέχεια δοθεί η εντολή `st2 = copy.copy(st1)` ή αν χρησιμοποιηθεί η εντολή `st3 = st1.copy()`, όπως φαίνεται στη συνέχεια.

```
>>> st1 = {'name': 'Nikos', 'grades': [7.0, 8.5, 6.5]}
>>> import copy
>>> st2 = copy.copy(st1)
>>> st3 = st1.copy()
>>> st1['name'] = 'Nikolaos'
>>> st1
{'name': 'Nikolaos', 'grades': [8.5, 6.5, 7.0]}
>>> st2
{'name': 'Nikos', 'grades': [8.5, 6.5, 7.0]}
>>> st3
```

```
{'name': 'Nikos', 'grades': [8.5, 6.5, 7.0]}
```

Στο προηγούμενο παράδειγμα, πραγματοποιήθηκε αντιγραφή του λεξικού και κάθε μεταβλητή από τις st1, st2, st3 αποτελεί πλέον αναφορά προς ξεχωριστό αντίγραφο του λεξικού. Ωστόσο, αν επιχειρήσουμε να αλλάξουμε τη λίστα βαθμών (grades) προσθέτοντας για παράδειγμα έναν ακόμα βαθμό στην αναφορά st1 θα διαπιστώσουμε ότι ο βαθμός θα έχει «προστεθεί» και στις αναφορές st2 και st3. Αυτό συμβαίνει διότι η αντιγραφή που θα συμβεί με την copy θα είναι «ρηχή», δηλαδή δεν θα προχωρήσει σε αντιγραφή των αντικειμένων που περιέχονται εντός του αντικειμένου που αντιγράφεται. Η συμπεριφορά αυτή φαίνεται στον ακόλουθο κώδικα που εκτελείται πάλι στο REPL.

```
>>> st1 = {'name':'Nikos', 'grades':[7.0, 8.5, 6.5]}
>>> st2 = copy.copy(st1)
>>> st3 = st1.copy()
>>> st1['name'] = 'Nikolaos'
>>> st1['grades'].append(10.0)
>>> st1
{'name': 'Nikolaos', 'grades': [7.0, 8.5, 6.5, 10.0]}
>>> st2
{'name': 'Nikos', 'grades': [7.0, 8.5, 6.5, 10.0]}
>>> st3
{'name': 'Nikos', 'grades': [7.0, 8.5, 6.5, 10.0]}
```

Για να γίνει η λεγόμενη βαθιά αντιγραφή χρησιμοποιείται η συνάρτηση copy.deepcopy() που δημιουργεί ένα νέο αντικείμενο και αναδρομικά αντιγράφει όλα τα εμφωλευμένα αντικείμενα που υπάρχουν από το παλιό στο νέο αντικείμενο. Η συμπεριφορά αυτή φαίνεται στον ακόλουθο κώδικα, όπου πλέον αλλαγές στο st1 που αφορούν το εμφωλευμένο αντικείμενο grades δεν επηρεάζουν το βαθύ αντίγραφο που έχει δημιουργηθεί και για το οποίο υπάρχει αναφορά από τη μεταβλητή st2. Σε ότι αφορά το αντίγραφο που έχει δημιουργηθεί από το st1 με τη μέθοδο copy() και για το οποίο η st3 είναι μια αναφορά, αυτό εξακολουθεί να είναι ένα ρηχό αντίγραφο.

```
>>> st1 = {'name':'Nikos', 'grades':[7.0, 8.5, 6.5]}
>>> import copy
>>> st2 = copy.deepcopy(st1)
>>> st3 = st1.copy()
>>> st1['name'] = 'Nikolaos'
>>> st1['grades'].append(10.0)
>>> st1
{'name': 'Nikolaos', 'grades': [7.0, 8.5, 6.5, 10.0]}
>>> st2
{'name': 'Nikos', 'grades': [7.0, 8.5, 6.5]}
>>> st3
{'name': 'Nikos', 'grades': [7.0, 8.5, 6.5, 10.0]}
```

Μια οπτικοποίηση του προηγούμενου παραδείγματος φαίνεται στην Εικόνα 21 και στην Εικόνα 22 που δημιουργήθηκαν με το pythontutor.com και όπου φαίνεται ότι η μεν μεταβλητή st2 είναι αναφορά προς ένα βαθύ αντίγραφο του αρχικού αντικειμένου ενώ η δε μεταβλητή st3 είναι αναφορά προς ένα ρηχό αντίγραφο.

Python 3.11
known limitations

```
1  st1 = {'name':'Nikos', 'grades':[7.0, 8.5, 6.5]}
2  import copy
3  st2 = copy.deepcopy(st1)
4  st3 = st1.copy()
5  st1['name'] = 'Nikolaos'
6  st1['grades'].append(10.0)
```

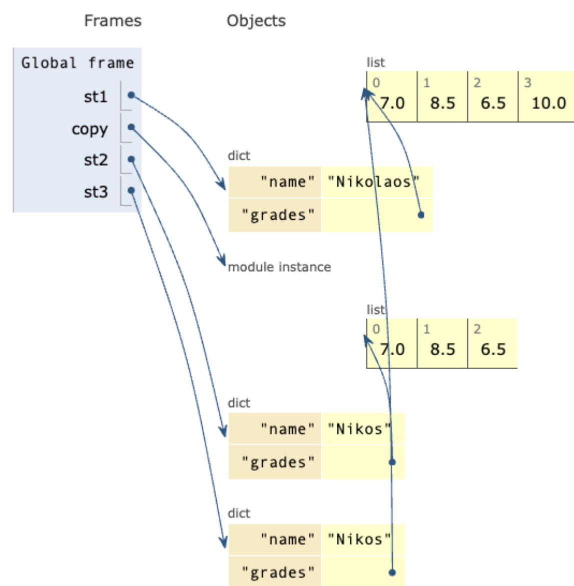
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Done running (6 steps)

Εικόνα 21 – Οπτικοποίηση ρηχής και βαθιάς αντιγραφής από το pythontutor.com (1/2) – https://tinyurl.com/ysdrahe2.



Εικόνα 22 – Οπτικοποίηση ρηχής και βαθιάς αντιγραφής από το pythontutor.com (2/2).

7.6. Κληρονομικότητα

Η κληρονομικότητα (inheritance) είναι ένα βασικό χαρακτηριστικό του OOP που επιτρέπει σε μια κλάση να οριστεί ως υποκλάση (subclass) μιας άλλης κλάσης που αναφέρεται ως υπερκλάση (superclass) της, κληρονομώντας τα πεδία δεδομένων και τις μεθόδους της. Η κληρονομικότητα διευκολύνει την επαναχρησιμοποίηση κώδικα (code reuse) καθώς οι υποκλάσεις δεν γράφονται από το μηδέν, αλλά χρησιμοποιούν ήδη υπάρχοντα κώδικα που απλά τον εξειδικεύουν.

Για να κληρονομήσει μια κλάση B από μια κλάση A, θα πρέπει κατά τον ορισμό της κλάσης B, η κλάση A να τοποθετηθεί σε παρενθέσεις δίπλα από το όνομα της κλάσης B όπως στη συνέχεια.

```
>>> class A:
...     pass
...
>>> class B(A):
...     pass
...
>>> obj1, obj2 = A(), B()
>>> type(obj1), type(obj2)
(<class '__main__.A'>, <class '__main__.B'>)
```

Στον κώδικα Κ. 221 παρουσιάζεται ένα παράδειγμα κληρονομικότητας με την κλάση `Employee` να είναι η υπερκλάση ή αλλιώς βασική κλάση και την κλάση `Manager` να είναι υποκλάση της `Employee` ή αλλιώς παραγόμενη κλάση της `Employee`. Στην υπερκλάση `Employee` ορίζονται τα

χαρακτηριστικά name και salary, η μέθοδος work ενώ ορίζεται και η μέθοδος __repr__. Η υποκλάση Manager κληρονομεί από την Employee τα χαρακτηριστικά name και salary. Επαναορίζει, υπερκαλύπτει (overrides) τη μέθοδο work και κληρονομεί την υλοποίηση της __repr__ από την υποκλάση, εφόσον δεν την ορίζει εκ νέου. Επίσης, διαθέτει τη δική της __init__ που ορίζει το επιπλέον χαρακτηριστικό team_size, ενώ καλεί με τη δεσμευμένη λέξη super() την __init__ της υπερκλάσης έτσι ώστε να αρχικοποιεί τα χαρακτηριστικά που κληρονομεί από την υπερκλάση. Στη συνέχεια δημιουργούνται δύο αντικείμενα, ένα της υπερκλάσης Employee και ένα της υποκλάσης Manager, εκτυπώνονται τα αντικείμενα και καλείται η μέθοδος work και για τα δύο αντικείμενα.

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def work(self):
        return f"O {self.name} εργάζεται."

    def __repr__(self):
        return f"Employee(name={self.name!r}, salary={self.salary})"


class Manager(Employee):
    def __init__(self, name, salary, team_size):
        super().__init__(name, salary)  # κλήση κατασκευαστή υπερκλάσης
        self.team_size = team_size

    def work(self):
        return f"O {self.name} διευθύνει {self.team_size} άτομα."


e1 = Employee("Πέτρος", 2000)
e2 = Manager("Μαρία", 4000, team_size=5)
print(e1)  # Employee(name='Πέτρος', salary=2000)
print(e1.work())  # O Πέτρος εργάζεται.
print(e2)  # Employee(name='Μαρία', salary=4000)
print(e2.work())  # O Μαρία διευθύνει 5 άτομα.
```

Κ. 221 – Παράδειγμα κληρονομικότητας με υπερκλάση την Employee και υποκλάση την Manager.

7.6.1.1 Άσκηση 2, κληρονομικότητα

Να γράψετε ένα πρόγραμμα που να επιδεικνύει την κληρονομικότητα (inheritance) μέσα από ένα παράδειγμα που αφορά ηλεκτρονικές συσκευές. Δημιουργήστε μια βασική κλάση Device με χαρακτηριστικά brand και power (ισχύς σε Watt), καθώς και μια μέθοδο turn_on που να

εμφανίζει μήνυμα ότι η συσκευή ενεργοποιήθηκε. Στη συνέχεια, δημιουργήστε δύο υποκλάσεις: την κλάση Laptop, που να προσθέτει το χαρακτηριστικό battery_life και να υπερκαλύπτει τη μέθοδο turn_on ώστε να εμφανίζει μήνυμα ότι ο φορητός υπολογιστής ενεργοποιήθηκε με μπαταρία, και την κλάση TV, που να προσθέτει το χαρακτηριστικό screen_size και να υπερκαλύπτει τη μέθοδο turn_on ώστε να εμφανίζει μήνυμα ότι η τηλεόραση ενεργοποιήθηκε και προβάλλει εικόνα. Δημιουργήστε αντικείμενα και των δύο υποκλάσεων και καλέστε τη μέθοδο turn_on για να φανεί η διαφορετική συμπεριφορά κάθε συσκευής.

Η λύση της άσκησης βρίσκεται στο Παράρτημα Α'.

7.6.2. Πολλαπλή κληρονομικότητα

Η Python υποστηρίζει πολλαπλή κληρονομικότητα (multiple inheritance) που είναι η δυνατότητα μια κλάσης να κληρονομεί από περισσότερες από μια υπερκλάσεις. Με αυτό τον τρόπο δίνεται η δυνατότητα συνδυασμού συμπεριφοράς και χαρακτηριστικών από δύο ή περισσότερες υπερκλάσεις. Ένα απλοϊκό αλλά διδακτικό παράδειγμα πολλαπλής κληρονομικότητας παρουσιάζεται στον κώδικα Κ. 222 όπου ορίζονται οι υπερκλάσεις Car και Aircraft και η υποκλάση FlyingCar που κληρονομεί από αυτές.

```python
class Car:
    def drive(self):
        print("Οδήγηση στο δρόμο")


class Aircraft:
    def fly(self):
        print("Πτήση στον αέρα")


class FlyingCar(Car, Aircraft):
    pass


fc = FlyingCar()
fc.drive()  # Οδήγηση στο δρόμο
fc.fly()  # Πτήση στον αέρα
```
Κ. 222 – Παράδειγμα πολλαπλής κληρονομικότητας.

Αν οι υπερκλάσεις διαθέτουν μεθόδους με το ίδιο όνομα, τότε επιλέγεται για την υποκλάση η μέθοδος από εκείνη την υπερκλάση που εμφανίζεται πρώτη εντός των παρενθέσεων στον ορισμό της υποκλάσης όπως φαίνεται στο κώδικα Κ. 223. Αυτή η σειρά επιλογής της κλάσης που θα

προηγηθεί για τη μέθοδο που θα κληρονομήσει η υποκλάση ονομάζεται MRO (Method Resolution Order).

```
class A:
    def foo(self):
        print("Μήνυμα από την κλάση A")


class B:
    def foo(self):
        print("Μήνυμα από την κλάση B")


class C(A, B):
    pass


C().foo()  # Μήνυμα από την κλάση A
```

Κ. 223 – Εφαρμογή της MRO για την επιλογή της μεθόδου που κληρονομείται.

7.7. Μεταβλητές κλάσης και μέθοδοι κλάσης

Για τη διευκόλυνση της σχεδίασης λύσεων σε προβλήματα μπορεί να είναι χρήσιμο να οριστούν μεταβλητές που να είναι κοινές για όλα τα αντικείμενα μιας κλάσης και μέθοδοι που να επιδρούν πάνω σε αυτές τις μεταβλητές και να ανήκουν στην κλάση συνολικά. Οι μεταβλητές αυτές ονομάζονται μεταβλητές κλάσης (class variables) και οι αντίστοιχες μέθοδοι ονομάζονται μέθοδοι κλάσης (class methods). Οι μεταβλητές κλάσης ορίζονται εντός της κλάσης αλλά εκτός των μεθόδων της κλάσης. Από την άλλη μεριά, οι μέθοδοι κλάσης ορίζονται με την επισημείωση (annotation) `@classmethod` και λαμβάνουν ως πρώτο όρισμα το `cls` που αντιστοιχεί στην ίδια την κλάση. Η πρόσβαση στις μεταβλητές κλάσης και στις μεθόδους κλάσης γίνεται μέσω του ονόματος της κλάσης (π.χ. `Student.total_students`) ή μέσω ενός αντικειμένου της κλάσης (π.χ. `s1.total_students`), όπως φαίνεται στον κώδικα Κ. 224.

```
class Student:
    total_students = 0

    def __init__(self, name):
        self.name = name
        Student.total_students += 1

    @classmethod
    def how_many(cls):
        return cls.total_students
```

```
s1 = Student("Νίκος")
s2 = Student("Μαρία")
print("s1:", s1.name)  # s1: Νίκος
print("s2:", s2.name)  # s2: Μαρία
print("Πλήθος σπουδαστών:", Student.how_many())  # Πλήθος σπουδαστών: 2
print("Πλήθος σπουδαστών:", Student.total_students)  # Πλήθος σπουδαστών: 2
print("Πλήθος σπουδαστών:", s1.total_students)  # Πλήθος σπουδαστών: 2
```
Κ. 224 – Μεταβλητή κλάσης και μέθοδος κλάσης.

7.7.1. Χρήση μεθόδου κλάσης ως εναλλακτικού κατασκευαστή

Μια συνηθισμένη χρήση των μεθόδων κλάσης είναι για τη δημιουργία των λεγόμενων factory μεθόδων που λειτουργούν ως εναλλακτικός τρόπος δημιουργίας αντικειμένων. Αυτό συμβαίνει στο παράδειγμα του κώδικα Κ. 225 όπου η κλάση `Circle` έχει έναν τυπικό κατασκευαστή που δέχεται ως όρισμα την ακτίνα του κύκλου και έναν επιπλέον δευτερεύοντα κατασκευαστή μέσω μεθόδου κλάσης που δέχεται ως όρισμα το εμβαδόν του κύκλου.

```
import math


class Circle:
    def __init__(self, radius):
        self.radius = radius

    @classmethod
    def from_area(cls, area):
        radius = math.sqrt(area / math.pi)
        return cls(radius)

    def area(self):
        return math.pi * self.radius**2


c1 = Circle(5)
print(f"c1: {c1.radius:.2f}, area: {c1.area():.2f}")  # c1: 5.00, area: 78.54
c2 = Circle.from_area(78.54)
print(f"c2: {c2.radius:.2f}, area: {c2.area():.2f}")  # c1: 5.00, area: 78.54
```
Κ. 225 – Μέθοδος κλάσης για δημιουργία αντικειμένων με εναλλακτικό τρόπο.

7.8. Στατικές μεταβλητές και στατικές μέθοδοι

Οι μεταβλητές κλάσης που αναφέρθηκαν ήδη ονομάζονται αλλιώς και στατικές μεταβλητές ή στατικά δεδομένα της κλάσης. Προσοχή όμως, οι μέθοδοι κλάσης και οι στατικές μέθοδοι είναι διαφορετικές έννοιες. Οι στατικές μέθοδοι ορίζονται εντός της κλάσης με την επισημείωση

@staticmethod αλλά δεν λαμβάνουν ως πρώτο όρισμα το self ή το cls. Ωστόσο, οι στατικές μέθοδοι δεν συνδέονται ούτε με την κλάση ούτε με τα στιγμιότυπα της κλάσης αλλά πρόκειται για συναρτήσεις που απλά γράφονται μέσα σε μια κλάση για καλύτερη οργάνωση κώδικα, ενώ δεν έχουν πρόσβαση στις μεταβλητές της κλάσης ούτε στις μεταβλητές των στιγμιοτύπων της κλάσης.

Για παράδειγμα, στον κώδικα Κ. 226 ορίζεται η κλάση FileHelper που λειτουργεί απλά ως ένα namespace για τη συνάρτηση get_extension, δηλαδή η συνάρτηση δεν καλείται απευθείας με το όνομά της, αλλά πρέπει να προηγείται το όνομα της κλάσης για την ορθή κλήση της.

```
class FileHelper:
    @staticmethod
    def get_extension(filename):
        """Επιστρέφει την επέκταση αρχείου"""
        return filename.split(".")[-1] if "." in filename else None


print(FileHelper.get_extension("document.pdf"))  # pdf
print(FileHelper.get_extension("archive.tar.gz"))  # gz
print(FileHelper.get_extension("no_extension"))  # None
```
Κ. 226 – Ορισμός μιας στατικής μεθόδου.

7.9. Σύνθεση

Η σύνθεση (composition) είναι ένας τρόπος συσχέτισης κλάσεων που αναφέρεται ως συσχέτιση τύπου "has-a" (έχει ένα). Χρησιμοποιείται όταν ένα αντικείμενο συντίθεται από άλλα αντικείμενα τα οποία διατηρεί ως επιμέρους χαρακτηριστικά. Για παράδειγμα, στον κώδικα Κ. 227 η κλάση Car ορίζει το χαρακτηριστικό engine που είναι αντικείμενο της κλάσης Engine, οπότε σε αυτή την περίπτωση λέμε ότι το αντικείμενο Car έχει ένα αντικείμενο Engine.

```
class Engine:
    def __init__(self, horsepower, engine_type):
        self.horsepower = horsepower
        self.engine_type = engine_type

    def start(self):
        print(f"Η μηχανή {self.engine_type} με {self.horsepower} ίππους ξεκίνησε.")


class Car:
    def __init__(self, make, model, engine):
        self.make = make
        self.model = model
        self.engine = engine  # Σύνθεση
```

```python
    def start_car(self):
        print(f"Εκκίνηση {self.make} {self.model}...")
        self.engine.start()


v6_engine = Engine(300, "V6")
my_car = Car("Toyota", "Supra", v6_engine)
my_car.start_car() # Εκκίνηση Toyota Supra...
                   # Η μηχανή V6 με 300 ίππους ξεκίνησε.
```
Κ. 227 – Παράδειγμα σύνθεσης αντικειμένων.

7.9.1.1 Άσκηση 3

Γράψτε ένα πρόγραμμα που να αναπαριστά τη σχέση σύνθεσης (composition) μέσα από την αλληλεπίδραση μεταξύ ενός υπολογιστή και του επεξεργαστή του. Δημιουργήστε μια κλάση CPU με χαρακτηριστικά `model` και `speed`, καθώς και μια μέθοδο `process()` που να εμφανίζει μήνυμα επεξεργασίας δεδομένων. Δημιουργήστε μια κλάση `Computer` που να περιέχει ένα αντικείμενο τύπου CPU και να διαθέτει τα χαρακτηριστικά `brand` και `ram`, καθώς και μια μέθοδο `start()` που να εμφανίζει μήνυμα εκκίνησης του υπολογιστή και να καλεί τη μέθοδο `process()` της CPU. Δημιουργήστε αντικείμενα των δύο κλάσεων και δείξτε την αλληλεπίδραση τους μέσω της μεθόδου `start()`.

Η λύση της άσκησης βρίσκεται στο Παράρτημα Α'.

7.10. Ενθυλάκωση

Μια ακόμη βασική αρχή του αντικειμενοστραφούς προγραμματισμού είναι η ενθυλάκωση (encapsulation) σύμφωνα με την οποία μπορούν να οριστούν επίπεδα προστασίας έτσι ώστε να περιορίζεται η πρόσβαση διαφόρων συναρτήσεων στα χαρακτηριστικά των αντικειμένων. Στην Python δεν υπάρχουν οι λέξεις κλειδιά private (ιδιωτικό), protected (προστατευμένο) και public (δημόσιο) που συναντώνται σε άλλες γλώσσες προγραμματισμού όπως η C++ και η Java, αλλά υπάρχουν κάποιες συμβάσεις που προτείνονται από την γλώσσα και ακολουθούνται από τους προγραμματιστές. Ωστόσο, είναι σημαντικό να τονιστεί ότι αυτές οι συμβάσεις δεν επιβάλλονται από την ίδια την γλώσσα και συνεπώς μπορούν να παρακαμφθούν. Οι συμβάσεις αυτές είναι:

- Όταν το όνομα μιας μεταβλητής ή μια μεθόδου μιας κλάσης ξεκινά με δύο κάτω παύλες τότε η μεταβλητή ή η μέθοδος θεωρείται ότι είναι private (επιτρέπεται η πρόσβαση μόνο από μεθόδους της ίδιας κλάσης).

- Όταν το όνομα μιας μεταβλητής ή μιας μεθόδου μιας κλάσης ξεκινά με μια κάτω παύλα τότε η μεταβλητή ή η μέθοδος θεωρείται ως protected (επιτρέπεται η πρόσβαση μόνο από μεθόδους της ίδιας κλάσης και των υποκλάσεών της).
- Όταν το όνομα μιας μεταβλητής ή μιας μεθόδου μιας κλάσης δεν ξεκινά με κάτω παύλα τότε η μεταβλητή ή η μέθοδος θεωρείται ως public (επιτρέπεται η πρόσβαση από οποιαδήποτε συνάρτηση ή μέθοδο)

Στον κώδικα Κ. 228 ορίζεται η κλάση `BankAccount` με τα χαρακτηριστικά `owner`, `_balance` και `__pin` που αντιστοιχούν στα επίπεδα προστασίας public, protected και private. Ομοίως ορίζονται οι public μέθοδοι `deposit` και `withdraw` και η private μέθοδος `__verify`. Ο όρος "name mangling", που υπάρχει σε σχόλιο του κώδικα, αναφέρεται στο μηχανισμό της Python που μετασχηματίζει ονόματα χαρακτηριστικών με διπλή κάτω παύλα (π.χ. __x) σε _ClassName__x ώστε να αποφεύγονται συγκρούσεις ονομάτων.

```
class BankAccount:
    def __init__(self, owner, balance):
        self.owner = owner  # public
        self._balance = balance  # protected
        self.__pin = 1234  # private

    def deposit(self, amount):
        self._balance += amount
        print(f"Κατάθεση ποσού {amount}")

    def __verify_pin(self, pin):  # private
        return pin == self.__pin

    def withdraw(self, amount, pin):
        if self.__verify_pin(pin):
            if amount <= self._balance:
                self._balance -= amount
                print(f"Ανάληψη ποσού {amount}")
            else:
                print("Μη επαρκές υπόλοιπο.")
        else:
            print("Λανθασμένος PIN!")


account = BankAccount("Χρήστος", 1000)
print("Ιδιοκτήτης:", account.owner)  # επιτρέπεται
print("Υπόλοιπο:", account._balance)  # επιτρέπεται αλλά ΔΕΝ συνιστάται
account.deposit(200)
account.withdraw(100, pin=1234)
```

```
# print(account.__pin)  # AttributeError αν επιχειρηθεί
print("PIN =", account._BankAccount__pin)  # πρόσβαση στο __pin με name mangling
# print(account.__verify_pin(1234)) # AttributeError αν επιχειρηθεί
print(account._BankAccount__verify_pin(1234))  # name mangling
```
Κ. 228 – Παράδειγμα με διαφορετικά επίπεδα προστασίας.

Η έξοδος του κώδικα Κ. 228 είναι η ακόλουθη (έξοδος Ε. 57).

```
Ιδιοκτήτης: Χρήστος
Υπόλοιπο: 1000
Κατάθεση ποσού 200
Ανάληψη ποσού 100
PIN = 1234
True
```
Ε. 57 – Έξοδος του κώδικα Κ. 228

7.11. Ιδιότητες (properties)

Στον αντικειμενοστραφή προγραμματισμό υπάρχουν μέθοδοι που είναι γνωστοί ως getters και setters. Ο ρόλος τους είναι να δίνουν πρόσβαση με ελεγχόμενο τρόπο στα μέλη δεδομένων της κλάσης. Στο παράδειγμα του κώδικα Κ. 229, setter είναι η μέθοδος `set_celsious()` η οποία δεν επιτρέπει την ανάθεση θερμοκρασίας μικρότερης από -273.15 βαθμούς Κελσίου στα αντικείμενα της κλάσης `Temperature`.

```
class Temperature:
    def __init__(self, celsius):
        self._celsius = celsius

    def get_celsius(self):
        return self._celsius

    def set_celsius(self, value):
        if value < -273.15:
            raise ValueError(
                "Θερμοκρασία κάτω από το απόλυτο μηδέν δεν μπορεί να υπάρξει."
            )
        self._celsius = value


t = Temperature(25)
print(t.get_celsius())  # 25
t.set_celsius(30)
print(t.get_celsius())  # 30
t.set_celsius(-300) # ValueError
```
Κ. 229 – Η κλάση Temperature με getter και setter για το χαρακτηριστικό _celsius.

Αν και οι getters και setters είναι χρήσιμοι, συχνά είναι κουραστικό να γράφονται και να καλούνται, δημιουργώντας τον λεγόμενο boilerplate κώδικα. Συνίσταται δε να αντικαθίστανται από properties για καθαρότερο και ιδιωματικό Python κώδικα. Συνεπώς, ο κώδικας Κ. 229 θα ήταν καλύτερο να γραφεί όπως παρουσιάζεται στον κώδικα Κ. 230 όπου πλέον χρησιμοποιούνται οι επισημειώσεις `@property` και `@celsious.setter`, κάτι που καθιστά την μετέπειτα χρήση των αντικειμένων της κλάσης ευχερέστερη.

```
class Temperature:
    def __init__(self, celsius):
        self._celsius = celsius

    @property
    def celsius(self):
        return self._celsius

    @celsius.setter
    def celsius(self, value):
        if value < -273.15:
            raise ValueError(
                "Θερμοκρασία κάτω από το απόλυτο μηδέν δεν μπορεί να υπάρξει."
            )
        self._celsius = value


t = Temperature(25)
print(t.celsius)  # 25
t.celsius = 30
print(t.celsius)  # 30
t.celsius = -300  # ValueError
```
Κ. 230 – Η κλάση Temperature με το χαρακτηριστικό _celsius ως property.

7.12. Ερωτήσεις κλειστού τύπου

1. Με ποιον τρόπο ορίζεται ότι μια κλάση B είναι υποκλάση μιας κλάσης A;
   A. `class A(B):`
   B. `class B(A):`
   C. `class B -> A:`
   D. `class B extends A:`

2. Ποιος είναι ο ρόλος της μεθόδου `__init__`;
   A. Διαγράφει το αντικείμενο από τη μνήμη
   B. Αρχικοποιεί τα χαρακτηριστικά (attributes) ενός αντικειμένου
   C. Ορίζει στατικές μεθόδους
   D. Καλείται μόνο μία φορά σε όλο το πρόγραμμα

3. Ποια από τις παρακάτω μεθόδους καλείται όταν χρησιμοποιούμε την print(obj) για ένα αντικείμενο obj;

A. __repr__ μόνο
B. __print__
C. __str__ (αν υπάρχει, αλλιώς __repr__ αν υπάρχει)
D. __call__

4. Ποια δήλωση για τα class attributes είναι σωστή;

A. Ανήκουν στην κλάση και είναι κοινά για όλα τα αντικείμενα
B. Ανήκουν σε κάθε αντικείμενο ξεχωριστά
C. Ορίζονται μόνο μέσα στη __init__
D. Δεν μπορούν να τροποποιηθούν

5. Ποια σχέση περιγράφει καλύτερα τη σύνθεση (composition);

A. is-a
B. kind-of
C. same-as
D. has-a

7.13. Ασκήσεις προς επίλυση

1. Δημιουργήστε μια κλάση Car που να έχει ως ιδιότητες τα make (μάρκα), model (μοντέλο), year (έτος κυκλοφορίας) και odometer (χιλιόμετρα που έχει διανύσει το αυτοκίνητο). Οι ιδιότητες να αρχικοποιούνται μέσω της __init__ της κλάσης. Συμπληρώστε μια μέθοδο με όνομα print_info που να εκτυπώνει ένα μήνυμα με τη μάρκα, το μοντέλο, το έτος του οχήματος και τα χιλιόμετρα που έχει διανύσει. Επίσης, συμπληρώστε μια μέθοδο με όνομα update_odometer που να ενημερώνει το odometer στην τιμή new_km που θα δέχεται ως παράμετρο, μόνο αν η νέα τιμή χιλιομέτρων είναι μεγαλύτερη από την υπάρχουσα τιμή, αλλιώς να εμφανίζει ένα μήνυμα.
2. Δημιουργήστε μια κλάση Dog με ιδιότητες name και age που να λαμβάνουν τιμές στη συνάρτηση αρχικοποίησης __init__(self, name, age) από τα ορίσματά της. Ορίστε τη μέθοδο bark(self) που να εμφανίζει μήνυμα της μορφής "Ο σκύλος [name] γαβγίζει!". Ορίστε τη μέθοδο get_info(self) που να επιστρέφει ένα μήνυμα της μορφής "Ο σκύλος [name] είναι [age] ετών". Δημιουργήστε 2 αντικείμενα της κλάσης Dog και καλέστε τις μεθόδους.
3. Δημιουργήστε μια κλάση Book με ιδιότητες title, author και year που να λαμβάνουν τιμές στη συνάρτηση αρχικοποίησης __init__(self, title, author, year). Ορίστε τη μέθοδο __str__(self) που να επιστρέφει μια φιλική προς τον χρήστη αναπαράσταση της

μορφής "Τίτλος: <title>, Συγγραφέας: <author>" και τη μέθοδο `__repr__(self)` που να επιστρέφει μια αναπαράσταση αντικειμένου Book της μορφής Book('<title>', '<author>', <year>). Δοκιμάστε την κλάση δημιουργώντας ένα αντικείμενο και εκτυπώνοντας το με `print(book)` και `print(repr(book))`.

4. Δημιουργήστε μια κλάση `BankAccount` με ιδιότητες `owner` (ιδιοκτήτης) και `balance` (υπόλοιπο). Η κλάση να έχει μέθοδο `__init__(self, owner, balance=0)` που αρχικοποιεί τον λογαριασμό με προεπιλεγμένο υπόλοιπο 0 ευρώ αν δεν δοθεί άλλη τιμή. Δημιουργήστε μια υποκλάση της `BankAccount` με όνομα `SavingsAccount` που προσθέτει την ιδιότητα `interest_rate` (επιτόκιο). Η `SavingsAccount` θα πρέπει να έχει μέθοδο `apply_interest(self)` που προσθέτει τόκους στο υπόλοιπο βάσει του επιτοκίου (π.χ. για `interest_rate=0.05`, προσθέτει 5% του υπολοίπου). Δημιουργήστε ένα `SavingsAccount` με επιτόκιο 5% και υπόλοιπο 3000 ευρώ καλέστε 10 φορές την `apply_interest()` και εμφανίστε το υπόλοιπο του λογαριασμού.
5. Δημιουργήστε μια κλάση `Student` με μεταβλητή κλάσης `total_students` που να αρχικοποιείται στο 0. Η κλάση θα πρέπει να έχει ιδιότητες στιγμιότυπου `name` (όνομα) και `grade` (τάξη). Στη μέθοδο `__init__(self, name, grade)` να αυξάνεται αυτόματα το `total_students` κατά 1 κάθε φορά που δημιουργείται νέος μαθητής. Δοκιμάστε την κλάση δημιουργώντας 3 μαθητές και εμφανίζοντας το συνολικό πλήθος μαθητών με `print(Student.total_students)`.
6. Δημιουργήστε μια κλάση `Address` με ιδιότητες `street` (οδός), `city` (πόλη) και `postal_code` (ταχυδρομικός κώδικας). Στη συνέχεια, δημιουργήστε μια κλάση `Person` με ιδιότητες `name` (όνομα) και `address` (διεύθυνση). Η ιδιότητα `address` θα πρέπει να είναι ένα αντικείμενο της κλάσης `Address`. Δημιουργήστε μια μέθοδο `get_full_info(self)` στην κλάση `Person` που να επιστρέφει ένα μήνυμα της μορφής "Όνομα: <name>, Διεύθυνση: <street>, <city>, <postal_code>". Δοκιμάστε τις κλάσεις δημιουργώντας ένα αντικείμενο `Address` και ένα αντικείμενο `Person` που να περιέχει αυτή τη διεύθυνση.

ΠΑΡΑΡΤΗΜΑ Α' - ΛΥΣΕΙΣ ΑΣΚΗΣΕΩΝ

Λύσεις ασκήσεων κεφαλαίου 1

Άσκηση 1

Εντολές στην Python: αριθμητική πράξη, αντιστροφή λεκτικού, ταξινόμηση λίστας τιμών.

```
>>> 2 + 3 * 4
14
>>> "Python"[::-1]
"nohtyP"
>>> sorted([7, 2, 3, 1, 6])
[1, 2, 3, 6, 7]
```

Τα παραδείγματα με τη random που ακολουθούν δίνουν διαφορετικά αποτελέσματα σε κάθε εκτέλεση.

```
>>> [x**2 for x in range(5)]
[0, 1, 4, 9, 16]
>>> import random; [random.randint(1,6) for _ in range(5)]
[6, 1, 1, 4, 2]
>>> random.choice(["πέτρα", "ψαλίδι", "χαρτί"])
"χαρτί"
```

Εμφάνιση του Zen της Python από τον Tim Peters (PEP20[4]).

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Ένας γρήγορος τρόπος να γραφεί το εισαγωγικό πρόγραμμα «Hello World!».

[4] https://peps.python.org/pep-0020/

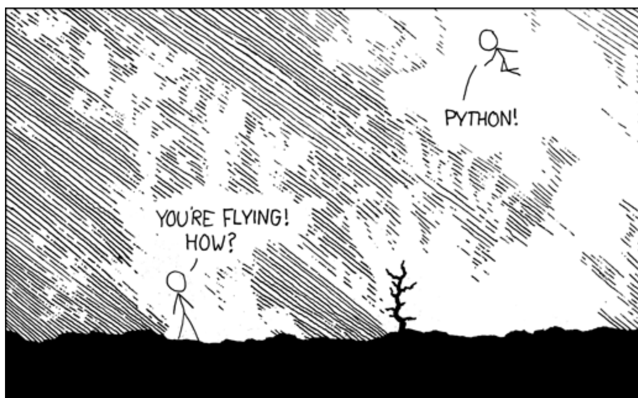
```
>>> import __hello__; __hello__.main()
Hello world!
```

Απάντηση στην πιθανότητα να χρησιμοποιηθούν αγκύλες αντί για εσοχές για τον ορισμό ενός μπλοκ κώδικα, κάποια στιγμή μελλοντικά στην Python.

```
>>> from __future__ import braces
  File "<python-input-3>", line 1
    from __future__ import braces
                           ^^^^^^
SyntaxError: not a chance
```

Ανοίγει τη σελίδα https://xkcd.com/353/ και εμφανίζει το κόμικ της Εικόνα 23.

```
>>> import antigravity
```



Εικόνα 23 – https://xkcd.com/353/.

Άσκηση 2

```
num = int(input("Δώσε έναν ακέραιο αριθμό: "))

if num == 0:
    print("Ο αριθμός είναι μηδέν")
elif num > 0 and num % 2 == 0:
    print("Ο αριθμός είναι άρτιος και θετικός")
elif num < 0 and num % 2 == 0:
    print("Ο αριθμός είναι άρτιος και αρνητικός")
elif num > 0 and num % 2 != 0:
    print("Ο αριθμός είναι περιττός και θετικός")
else:
```

```
    print("Ο αριθμός είναι περιττός και αρνητικός")
```
Κ. 231 – Λύση 2ης άσκησης κεφαλαίου 1.

Άσκηση 3

Η άσκηση 3 ζητά την υλοποίηση του παιχνιδιού «μάντεψε τον αριθμό». Για τη δημιουργία του τυχαίου αριθμού στο διάστημα [1, 100] χρησιμοποιείται η συνάρτηση `randint()` του module `random`. Μια υλοποίηση της λύσης είναι η ακόλουθη:

```
import random

lucky_number = random.randint(1, 100)
print("Ο τυχερός αριθμός δημιουργήθηκε!")

winner = False
c = 0
for i in range(6):
    c += 1
    x = int(input(f"Μαντέψτε τον αριθμό (προσπάθεια {c}): "))
    if x == lucky_number:
        winner = True
        break
    elif x < lucky_number:
        print("Πολύ χαμηλά")
    else:
        print("Πολύ υψηλά")

if winner:
    print(f"Κέρδισες με την προσπάθεια {c}!")
else:
    print(f"Έχασες, ο τυχερός αριθμός ήταν ο {lucky_number}!")
```
Κ. 232 – Πρόγραμμα για το «μάντεψε τον αριθμό».

Ένα παράδειγμα εκτέλεσης του προγράμματος παρουσιάζεται στην ακόλουθη έξοδο:

```
Ο τυχερός αριθμός δημιουργήθηκε!
Μαντέψτε τον αριθμό (προσπάθεια 1): 50
Πολύ υψηλά
Μαντέψτε τον αριθμό (προσπάθεια 2): 25
Πολύ υψηλά
Μαντέψτε τον αριθμό (προσπάθεια 3): 12
Πολύ υψηλά
Μαντέψτε τον αριθμό (προσπάθεια 4): 6
Πολύ χαμηλά
Μαντέψτε τον αριθμό (προσπάθεια 5): 9
Κέρδισες με την προσπάθεια 5!
```
Ε. 58 – Παράδειγμα εκτέλεσης για το «μάντεψε τον αριθμό».

Λύσεις ασκήσεων κεφαλαίου 2

Άσκηση 1

Μετατροπή των βαθμών Κελσίου σε βαθμούς Φαρενάιτ, με έλεγχο έγκυρης εισόδου και ερώτηση για νέα μετατροπή.

```
def celcius_to_fahrenheit(c):
    """
    Δέχεται σαν όρισμα έναν δεκαδικό αριθμό που θεωρεί ότι είναι θερμοκρασία
    σε βαθμούς Κελσίου και το μετατρέπει σε βαθμούς Φαρενάιτ.
    """
    return (c * 9 / 5) + 32

def is_decimal(c):
    """
    Έλεγχος εκγυρότητας εισόδου για δεκαδικούς αριθμούς.
    """
    c = c.strip()
    if c[0] in "-+":   # αγνόηση πιθανού προσήμου
        offset = 1
    else:
        offset = 0

    # Έλεγχος εισόδου αν έχει υποδιαστολή
    if '.' in c[offset:]:
        a, b = c[offset:].split('.')
        if a.isdecimal() and b.isdecimal():
            return True
    # Έλεγχος εισόδου αν είναι ακέραιος
    if c[offset:].isdecimal():
        return True
    else:
        return False

def check_dp(c):
    """
    Αν έχει δοθεί κόμμα σαν σημείο υποδιαστολής, το αλλάζει σε τελεία.
    """
    if ',' in c:
        c = c.replace(',', '.')
    return c


input_ok = False
while not input_ok:
    c = input("Δώστε τη θερμοκρασία σε βαθμούς Κελσίου, ή <Enter> για έξοδο: ")
```

```
    if c == "":
        break

    c = check_dp(c)
    input_ok = is_decimal(c)
    if input_ok:
        c = float(c)
        print(f"Οι {c}ºC είναι {celcius_to_fahrenheit(c):.2f}ºF.")
        input_ok = False
    else:
        print(f"Το {c} δεν είναι αποδεκτή είσοδος, παρακαλώ δώστε νέα είσοδο.")
```

Κ. 233 - Μετατροπή από βαθμούς Κελσίου σε βαθμούς Φαρανάιτ, με έλεγχο έγκυρης εισόδου.

Άσκηση 2

```
def perimeter(a, b, c, *d):
    """
    Υπολογίζει την περίμετρο τυχαίου πολυγώνου. Χρειάζονται τουλάχιστον τρεις
    πλευρές, οπότε οι τρεις πρώτες παράμετροι είναι υποχρεωτικές.
    """
    length = a + b + c
    for side in d:
        length = length + side
    return length
```

Κ. 234 - Συνάρτηση που υπολογίζει την περίμετρο τυχαίου πολυγώνου.

Άσκηση 3

Αναδρομική συνάρτηση υπολογισμού της ακολουθίας Fibonacci,με ενδεικτικές κλήσεις της για τους αριθμούς 5, 10, 20 και 40.

```
def fib(n):
    """
    Συνάρτηση που υπολογίζει την ακολουθία Fibonacci αναδρομικά.
    """
    #print(f"fib({n})")
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

for m in (5, 10, 20, 40):
    print(f"fib({m}) = {fib(m)}.")
```

Κ. 235 - Συνάρτηση που υπολογίζει αναδρομικά την ακολουθία Fibonacci.

```
fib(5) = 5.
fib(10) = 55.
fib(20) = 6765.
fib(40) = 102334155.
```

Ε. 59 - Ενδεικτικά αποτελέσματα για τις τιμές 5, 10, 20 και 40.

Λύσεις ασκήσεων κεφαλαίου 3

Άσκηση 1

```
1.  days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
2.  log_data = list()
3.  chars = list("ALGORITHM")
```

Άσκηση 2

```
1.  buffer = [0] * 50
2.  status_codes = ["Pending"] * 10
```

Άσκηση 3

```
1.  multiples_of_five = list(range(0, 101, 5))
2.  cubes = [x**3 for x in range(1, 11)]
3.  even_squares = [x**2 for x in range(1, 21) if x % 2 == 0]
```

Άσκηση 4

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
# Αρχική πλειάδα
coords = (10.5, 20.8, 30.1)

# Μετατροπή σε λίστα
coords_list = list(coords)
print(coords_list)       # [10.5, 20.8, 30.1]
```

Άσκηση 5

Η λύση της άσκησης είναι:

1. web_history[-1]

2. web_history[-3]

3. web_history[-2:]

4. Επιστρέφει ολόκληρη τη λίστα. Στο slicing, αν ο δείκτης έναρξης είναι μικρότερος από το όριο, η Python ξεκινά από το πρώτο διαθέσιμο στοιχείο χωρίς να διακόψει την εκτέλεση.

Άσκηση 6

Η λύση της άσκησης είναι:

1. numbers[3:8] (Ο δείκτης 8 δεν συμπεριλαμβάνεται, άρα σταματά στο 70).

2. numbers[:4]

3. numbers[::2]

4. numbers[::-1]

Άσκηση 7

Η λύση της άσκησης είναι:

1. `prices[1] = 30.00`
2. `prices[-1] = prices[-1] - 2.00` (ή `prices[-1] -= 2.00`)
3. `print(prices)`

Άσκηση 8

Η λύση της άσκησης είναι:

1. `schedule[3:6] = ["Άδεια", "Άδεια", "Άδεια"]`
2. `schedule[:2] = ["Τηλεργασία", "Τηλεργασία"]`
3. Η Python θα "σπρώξει" τα υπόλοιπα στοιχεία και η λίστα θα μεγαλώσει κατά μία θέση. Το slicing assignment επιτρέπει την αντικατάσταση ενός τμήματος με λίστα διαφορετικού μήκους.

```
schedule = ["Εργασία", "Εργασία", "Ρεπό", "Εργασία", "Εργασία",
            "Εργασία", "Εργασία", "Ρεπό", "Εργασία", "Εργασία"]

# Ανάθεση 3 στοιχείων σε τμήμα 2 θέσεων
schedule[0:2] = ["Α", "Β", "Γ"]

print(f"Μετά: {schedule}")
# Εξοδος: ['Α', 'Β', 'Γ', 'Ρεπό', 'Εργασία', 'Εργασία', 'Εργασία', 'Εργασία', 'Ρεπό',
'Εργασία', 'Εργασία']
print(f"Μήκος μετά: {len(schedule)}")  #Εξοδος: Μήκος μετά: 11
```

Άσκηση 9

Η λύση της άσκησης είναι:

```
products = ["Γάλα", "Ψωμί", "Αυγά", "Τυρί"]

products.append("Γιαούρτι")       # Προσθήκη στο τέλος
products.insert(1, "Μέλι")        # Εισαγωγή στη θέση 1
products.remove("Αυγά")           # Αφαίρεση βάσει τιμής
print(len(products))              # Εκτύπωση πλήθους (4)
```

Άσκηση 10

Η λύση της άσκησης είναι:

```
temps = [12, 15, 10, 18, 15, 22, 15]

print(temps.count(15))      # Καταμέτρηση του 15 (3)
print(temps.index(22))      # Θέση του 22 (5)
temps.sort(reverse=True)    # Φθίνουσα ταξινόμηση [22, 18, 15, 15, 15, 12, 10]
temps.reverse()             # Αντιστροφή σειράς [10, 12, 15, 15, 15, 18, 22]
```

Άσκηση 11

Η λύση της άσκησης είναι:

```
users = ["Νίκος", "Ελένη", "Ανδρέας", "Μαρία", "Κώστας"]

if "Γιώργος" not in users:        # Έλεγχος ύπαρξης
    print("Ο χρήστης δεν βρέθηκε")

removed_user = users.pop()        # Αφαίρεση τελευταίου ("Κώστας")
del users[0]                      # Διαγραφή πρώτου στοιχείου ("Νίκος")
```

Άσκηση 12

Η λύση της άσκησης είναι:

```
basket_a = ["Μήλο", "Αχλάδι"]
basket_b = ["Φράουλα", "Ακτινίδιο"]
```

```
basket_a.extend(basket_b)          # Επέκταση: ["Μήλο", "Αχλάδι", "Φράουλα", "Ακτινίδιο"]
print("Μπανάνα" not in basket_a)  # Έλεγχος απουσίας (True)
basket_a.clear()                   # Καθαρισμός λίστας ([])
```

Άσκηση 13

Η λύση της άσκησης είναι:

```
prices = [10.5, 45.0, 12.8, 33.4, 60.1, 22.0]
max_price = prices[0]  # Αρχικοποίηση με το πρώτο στοιχείο

for price in prices:
    if price > max_price:
        max_price = price

print(f"Η μέγιστη τιμή είναι: {max_price}")
```

Άσκηση 14

Η λύση της άσκησης είναι:

```
basket = ['Μήλο', 'Μπανάνα', 'Κεράσι', 'Μήλο', 'Πορτοκάλι', 'Μήλο']

for i in range(len(basket)):
    if basket[i] == "Μήλο":
        basket[i] = "Εξαντλήθηκε"

print(basket)
```

Άσκηση 15

Η λύση της άσκησης είναι:

```
runners = ["Δημήτρης", "Κατερίνα", "Κώστας", "Σοφία"]
for i, name in enumerate(runners, start=1):
    print(f"• Μετάλλιο {i}: {name}")
```

Άσκηση 16

Η λύση της άσκησης είναι:

```python
grades = [
    [75, 82, 90, 68],  # Μαθητής 1
    [40, 52, 35, 45],  # Μαθητής 2
    [95, 90, 92, 88],  # Μαθητής 3
    [62, 58, 70, 65],  # Μαθητής 4
    [48, 50, 52, 47]   # Μαθητής 5
]

# Λίστα για να αποθηκεύσουμε τα συγκεντρωτικά στοιχεία (θα είναι λίστα από λίστες)
# Μορφή κάθε στοιχείου: [Μέσος Όρος, Αριθμός Μαθητή]
summary = []

print("--- ΑΠΟΤΕΛΕΣΜΑΤΑ ΑΝΑ ΜΑΘΗΤΗ ---")
for i in range(len(grades)):
    student_grades = grades[i]

    # 1. Μέσος Όρος
    avg = sum(student_grades) / len(student_grades)

    # 2. Καλύτερο Μάθημα
    best = max(student_grades)

    # 3. Έλεγχος αν πέρασε
    if avg >= 50:
        status = "Πέρασε"
    else:
        status = "Απέτυχε"

    print(f"Μαθητής {i+1}: Μ.Ο. = {avg:.2f}, Καλύτερο = {best}, Κατάσταση = {status}")

    # Αποθηκεύουμε τα στοιχεία για την ταξινόμηση
    # Βάζουμε πρώτο τον Μέσο Όρο για να γίνει εύκολα η ταξινόμηση μετά
    summary.append([avg, i + 1])

# 4. Ταξινόμηση κατά επίδοση
# Η sort() σε μια λίστα από λίστες, ταξινομεί με βάση το πρώτο στοιχείο (εδώ τον Μ.Ο.)
summary.sort(reverse=True)

print("\n--- ΚΑΤΑΤΑΞΗ ---")
for rank in range(len(summary)):
    # Το summary[rank][0] είναι ο βαθμός, το summary[rank][1] είναι ο αριθμός μαθητή
    score = summary[rank][0]
    student_id = summary[rank][1]
    print(f"{rank+1}η θέση: Μαθητής {student_id} με Μ.Ο. {score:.2f}")
```

Άσκηση 17

Η λύση της άσκησης είναι:

```
# Αρχική λίστα τμήματος
tmima_A = ["Νίκος", "Άννα", "Γιώργος", "Μαρία", "Χρήστος"]

# 1. Δημιουργήστε ένα αντίγραφο της tmima_A με το όνομα tmima_B χρησιμοποιώντας την
.copy()
tmima_B = tmima_A.copy()

# 2. Προσθέστε το όνομα "Ελένη" στην tmima_B
tmima_B.append("Ελένη")

# 3. Εκτυπώστε και τις δύο λίστες
print("Αρχικό Τμήμα:", tmima_A)
print("Νέο Τμήμα:", tmima_B)
```

Άσκηση 18

Η λύση της άσκησης είναι:

```
import copy

# Πίνακας: [Βαθμός1, Βαθμός2] για κάθε μαθητή
grades = [[50, 60], [80, 70], [45, 55], [90, 85], [48, 52]]

# 1. Δημιουργήστε ένα "βαθύ αντίγραφο" με το όνομα grades_backup
grades_backup = copy.deepcopy(grades)

# 2. Αλλάξτε τον πρώτο βαθμό του πρώτου μαθητή σε 100 στο grades_backup
grades_backup[0][0]=100

# 3. Ελέγξτε αν άλλαξε ο αρχικός πίνακας
print("Αρχικοί Βαθμοί (πρώτος μαθητής):", grades[0])
print("Αντίγραφο Βαθμών (πρώτος μαθητής):", grades_backup[0])
```

Άσκηση 19

Η λύση της άσκησης είναι:

```
pip install emoji
```

Άσκηση 20

Η λύση της άσκησης είναι:

1. **Converter.py.**

```python
# Αρχείο: converter.py

def to_percentage(score, total):
    """
    Υπολογίζει το ποσοστό επί τοις εκατό.
    score: Ο βαθμός που επιτεύχθηκε
    total: Το μέγιστο δυνατό (άριστα)
    """
    if total == 0:
        return 0.0

    percentage = (score / total) * 100
    return percentage
```

2. **app.py**

```python
# Αρχείο: app.py
from converter import to_percentage

def main():
    try:
        # Λήψη δεδομένων από τον χρήστη
        # Χρησιμοποιούμε float για να δεχόμαστε και δεκαδικούς βαθμούς
        user_score = float(input("Εισάγετε τον βαθμό σας: "))
        max_score = float(input("Εισάγετε το άριστα (συνολικός βαθμός): "))

        # Κλήση της συνάρτησης από το άλλο αρχείο
        result = to_percentage(user_score, max_score)

        # Εμφάνιση αποτελέσματος με 1 δεκαδικό ψηφίο
        print(f"Το ποσοστό σας είναι: {result:.1f}%")

    except ValueError:
        print("Παρακαλώ εισάγετε έγκυρους αριθμούς.")

if __name__ == "__main__":
    main()
```

Άσκηση 21

Η λύση της άσκησης είναι:

Αρχείο: warehouse/finance/pricing.py

```
def apply_discount(price, discount_percentage):
   return price * (1 - discount_percentage / 100)
```

Αρχείο: warehouse/ finance/__init__.py :κενό

Αρχείο: warehouse/products/info.py

```
def get_product(product_id):
    """Επιστρέφει το όνομα του προϊόντος βάσει του ID του."""
    database = {
        101: "Laptop Pro",
        102: "Smartphone X",
        103: "Wireless Headphones"
    }
    # Επιστρέφει το όνομα ή ένα μήνυμα αν δεν βρεθεί το ID
    return database.get(product_id, "Προϊόν μη διαθέσιμο")
```

Αρχείο: warehouse/ products/__init__.py :κενό

Αρχείο: warehouse/__init__.py

```
from .products.info import get_product
from .finance.pricing import apply_discount
```

Αρχείο: main.py

```
item = warehouse.get_product(101)
final_price = warehouse.apply_discount(1200, 15)

print(f"Προϊόν: {item}")
print(f"Τελική Τιμή: {final_price}€")
```

Άσκηση 22

Η λύση της άσκησης είναι:

- **Το αρχείο greetings.py**

```
# Αρχείο: greetings.py

def say_hello():
```

```python
    print("Γεια σας από το module!")

# Αυτό το μπλοκ εκτελείται ΜΟΝΟ αν τρέξουμε το αρχείο απευθείας
if __name__ == "__main__":
    print("Το module εκτελείται αυτόνομα")
```

- **Το αρχείο index.py**

```python
# Αρχείο: index.py
import greetings

# Καλούμε τη συνάρτηση από το module
greetings.say_hello()
```

Λύσεις ασκήσεων κεφαλαίου 4

Άσκηση 1

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

Άσκηση 1.1:

```python
colors = ("κόκκινο", "πράσινο", "μπλε")
for color in colors:
    print(color)
```

Άσκηση 1.2:

```python
colors = ("κόκκινο", "πράσινο", "μπλε")
i = 0  # Αρχικοποίηση μετρητή
while i < len(colors):
    print(colors[i])
    i += 1  # Προσαύξηση μετρητή
```

Άσκηση 2

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```python
grades = (8, 9, 7, 10)
for i in range(len(grades)):
    print(f"Ο βαθμός στη θέση {i} είναι {grades[i]}")
```

Άσκηση 3

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
# Δεδομένα
lista1 = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]
lista2 = [2, 4, 6, 8, 10, 12, 14]
lista3 = [1, 3, 5, 7, 9, 11, 13]

# Μετατροπή σε σύνολα
set1 = set(lista1)
set2 = set(lista2)
set3 = set(lista3)

# 1. Μοναδικοί αριθμοί από την lista1
print("Μοναδικοί αριθμοί lista1:", set1)
# {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
# Το 5 εμφανιζόταν 2 φορές στη λίστα, στο set κρατήθηκε μία

# 2. Αριθμοί που υπάρχουν στις lista1 ΚΑΙ lista2 (Τομή)
common = set1 & set2  # ή set1.intersection(set2)
print("Κοινοί lista1 & lista2:", common)
# {2, 4, 6, 8, 10}

# 3. Αριθμοί που υπάρχουν στην lista1 αλλά ΟΧΙ στην lista3 (Διαφορά)
diff = set1 - set3  # ή set1.difference(set3)
print("Στη lista1 αλλά ΟΧΙ στη lista3:", diff)
# {2, 4, 6, 8, 10}
```

Άσκηση 4

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
products = ["μήλα", "ψωμί", "γάλα", "τυρί"]
quantities = [3, 1, 2, 1]
# Αναμενόμενο αποτέλεσμα και με τους 3 τρόπους:
# {'μήλα': 3, 'ψωμί': 1, 'γάλα': 2, 'τυρί': 1}

# Τρόπος 1: Με άγκιστρα {}
cart_1 = {"μήλα": 3, "ψωμί": 1, "γάλα": 2, "τυρί": 1}
print("Τρόπος 1:", cart_1)

# Τρόπος 2: Με dict()
```

```
cart_2 = dict(μήλα=3, ψωμί=1, γάλα=2, τυρί=1)
print("Τρόπος 2:", cart_2)
```

Άσκηση 5

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
temperatures = {
    "Δευτέρα": 18, "Τρίτη": 22, "Τετάρτη": 17,
    "Πέμπτη": 25, "Παρασκευή": 23, "Σάββατο": 20, "Κυριακή": 16
}

# 1. Μόνο οι μέρες (κλειδιά)
print("--- Μέρες ---")
for day in temperatures.keys():
    print(day)

# 2. Μόνο οι θερμοκρασίες (τιμές)
print("\n--- Θερμοκρασίες ---")
for temp in temperatures.values():
    print(temp)

# 3. Μέρα και θερμοκρασία μαζί
print("\n--- Εβδομάδα ---")
for day, temp in temperatures.items():
    print(f"{day} → {temp}°C")

# 4. Μόνο μέρες πάνω από 20°C
print("\n--- Ζεστές μέρες (>20°C) ---")
for day, temp in temperatures.items():
    if temp > 20:
        print(f"{day}: {temp}°C")
```

Άσκηση 6

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
grades = {"Μαθηματικά": 15, "Φυσική": 12, "Ιστορία": 18}

# 1. Προσθήκη Χημείας με update()
grades.update({"Χημεία": 14})
print("Μετά την update():", grades)
```

```
# 2. Ανάγνωση Γεωγραφίας με get()
geography_grade = grades.get("Γεωγραφία", "Δεν βρέθηκε")
print("Βαθμός Γεωγραφίας:", geography_grade)

# 3. Αφαίρεση Φυσικής με pop()
physics_grade = grades.pop("Φυσική")
print("Αφαιρέθηκε η Φυσική με βαθμό:", physics_grade)

# 4. Τελικό λεξικό
print("Τελικό λεξικό:", grades)
print("Αριθμός μαθημάτων:", len(grades))
```

Άσκηση 7

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
1.  cubes = [x**3 for x in range(1, 11)]
2.  even_squares = [x**2 for x in range(1, 21) if x % 2 == 0]
```

Άσκηση 8

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
products = {"laptop": 800, "ποντίκι": 25, "οθόνη": 350, "καλώδιο": 8, "κάμερα": 120}
affordable = {item: round(price * 0.9, 1) for item, price in products.items() if price <
100}
print("affordable:", affordable)
```

Άσκηση 9

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
name = "Ελένη"
age = 22
balance = 1534.678
products = ["laptop", "ποντίκι", "οθόνη"]

# 1. Βασικό μήνυμα
print(f"Καλωσόρισες, {name}! Είσαι {age} ετών.")
```

```
# 2. Υπόλοιπο με 2 δεκαδικά
print(f"Το υπόλοιπό σου είναι: {balance:.2f}€")

# 3. Εκτύπωση προϊόντων με for
for i, product in enumerate(products, 1):
    print(f"Προϊόν {i}: {product}")
```

Άσκηση 10

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
city = "Θεσσαλονίκη"
country = "Ελλάδα"
temperature = 23.456

# 1. Απλά {}
print("Η {} βρίσκεται στην {}.".format(city, country))

# 2. Αριθμητικοί δείκτες {0}, {1}
print("Η {0} είναι στην {1} και η {0} είναι όμορφη.".format(city, country))

# 3. Ονοματισμένες παράμετροι με 2 δεκαδικά
print("Πόλη: {city}, Χώρα: {country}, Θερμοκρασία: {temp:.2f}°C".format(
    city=city, country=country, temp=temperature
))
```

Άσκηση 11

Η λύση της άσκησης παρουσιάζεται στον παρακάτω κώδικα:

```
product = "Laptop"
quantity = 3
unit_price = 799.999
total = quantity * unit_price

# 1. Προϊόν με %s
print("Προϊόν: %s" % product)

# 2. Τεμάχια με %d
print("Τεμάχια: %d" % quantity)

# 3. Τιμή μονάδας με %.2f
print("Τιμή μονάδας: %.2f€" % unit_price)
```

```
# 4. Συνδυασμός %d, %s, %.2f
print("Αγόρασες %d τεμάχια %s συνολικής αξίας %.2f€" % (quantity, product, total))
```

Λύσεις ασκήσεων κεφαλαίου 5

Άσκηση 1

Ακολουθεί μια ενδεικτική λύση της άσκησης.

```
import csv

"""
Η πρώτη γραμμή είναι επικεφαλίδα που περιέχει τα ονόματα των στηλών.
Χρησιμοποιείται για να διαβαστούν οι υπόλοιπες γραμμές σαν λεξικά,
με κλειδιά τα ονόματα των στηλών.
"""
filepath = "compsci-books-file.csv"
with open(filepath, "r", encoding="utf_8", newline='') as book_file:
    reader = csv.DictReader(book_file, delimiter=";")
    books = []
    for row in reader:
        books.append(row)

    # Ανάγνωση των γραμμών και προσθήκη των νέων στηλών με τυχαίες τιμές
    for book in books:
        if "Knuth" in book["Συγγραφέας"]:
            book["Τιμή"] = 59
            book["Έκπτωση"] = 15
            book["Τιμή έκπτωσης"] = book["Τιμή"] * ( 100 - book["Έκπτωση"]) / 100
            book["Βαθμολογία"] = 5
        elif "Παπαθεοδώρου" in book["Συγγραφέας"]:
            book["Τιμή"] = 30
            book["Έκπτωση"] = 20
            book["Τιμή έκπτωσης"] = book["Τιμή"] * ( 100 - book["Έκπτωση"]) / 100
            book["Βαθμολογία"] = 4
        elif "Ritchie" in book["Συγγραφέας"]:
            book["Τιμή"] = 25
            book["Έκπτωση"] = 10
            book["Τιμή έκπτωσης"] = book["Τιμή"] * ( 100 - book["Έκπτωση"]) / 100
            book["Βαθμολογία"] = 4.5
        else:
            book["Τιμή"] = 35
            book["Έκπτωση"] = 18
            book["Τιμή έκπτωσης"] = book["Τιμή"] * ( 100 - book["Έκπτωση"]) / 100
```

```
            book["Βαθμολογία"] = 4.7

# Τα κλειδιά του λεξικού χρειάζονται για την εγγραφή του νέου αρχείου CSV
book_keys = list(books[0].keys())

new_file = "compsci-books-file-added-columns.csv"
with open(new_file, "w", encoding="utf_8", newline='') as book_file:
    writer = csv.DictWriter(book_file, book_keys, delimiter=";")

    # Εγγραφή της πρώτης γραμμής με τα ονόματα των στηλών
    writer.writeheader()
    # Εγγραφή ένα λεξικό (ένα για κάθε βιβλίο) τη φορά
    #for book in books:
    #    writer.writerow(book)

    # Εγγραφή ολων των λεξικών μαζί
    writer.writerows(books)
```

Αν εισαχθεί το νέο αρχείο στο excel, καλό είναι οι στήλες με το ISBN, τις τιμές και τη βαθμολογία να εισαχθούν αρχικά σαν κείμενο (text). Κατόπιν να αλλαχτεί η υποδιαστολή, όπου υπάρχει, σε κόμμα και μετά να αλλαχτεί ο τύπος των στηλών με τις τιμές και τη βαθμολογία σε νόμισμα (currency) ή δεκαδικό. Αυτό γιατί το excel, λόγω ελληνικών ρυθμίσεων, περιμένει κόμμα για υποδιαστολή και εισάγει λάθος τιμές. Θεωρεί ότι η τελεία είναι διαχωριστικό για τις χιλιάδες, εκατομμύρια, κλπ..

Άσκηση 2

Παρακάτω ακολουθεί μια ενδεικτική λύση.

```
import csv
from openpyxl import Workbook

"""
Η πρώτη γραμμή είναι επικεφαλίδα που περιέχει τα ονόματα των στηλών.
Χρησιμοποιείται για να διαβαστούν οι υπόλοιπες γραμμές σαν λεξικά,
με κλειδιά τα ονόματα των στηλών.
"""
filepath = "compsci-books-file-added-columns.csv"
with open(filepath, "r", encoding="utf_8", newline='') as book_file:
    reader = csv.DictReader(book_file, delimiter=";")
    books = []
    for row in reader:
        books.append(row)
    # print(books)

    # Δημιουργία workbook
```

```python
    wb = Workbook()
    # χρήση του ενεργού worksheet
    ws = wb.active
    col_names = list(books[0].keys())
    # απόδοση ονόματος στο worksheet
    ws.title = "compsci-books-added-columns"

    # Εγγραφή τίτλων  στηλών
    for col in range(1, len(col_names) + 1):
        ws.cell(1, col, value=col_names[col - 1])

    # Εγγραφή δεδομένων
    for i in range(len(books)):
        for j in range(len(col_names)):
            ws.cell(i + 2, j + 1, books[i][col_names[j]])

# Σώσιμο του αρχείου (workbook) excel
wb.save("compsci-books-file-added-columns.xlsx")
```

Άσκηση 3

```python
import os
import shutil
from pathlib import Path

def create_backup_dir(target_path, csubdirs):
    """
    Δημιουργεί τον φάκελλο των αντιγράφων ασφαλείας εκεί που ζητείται και
    από κάτω του δημιουργεί τους υποφακέλους.
    Αν υπάρχουν ήδη δεν προκαλείται λάθος.
    """
    if os.path.exists(target_path):
        os.makedirs(os.path.join(target_path, "backups"), exist_ok=True)
        for subd in subdirs:
            os.makedirs(os.path.join(target_path, "backups/" + subd), exist_ok=True)
        return True
    else:
        print(f"Το μονοπάτι {target_path} δεν υπάρχει.")
        return False

def find_files(dirpath, extension):
    """
    Αναζητάει αναδρομικά αρχεία που έχουν τη ζητούμενη κατάληξη και τα
    επιστρέφει σε μια λίστα.
```

```python
    """
    results = []
    for path, dirs, files in os.walk(dirpath):
        for f in files:
            if f[-len(extension):] == extension:
                results.append(os.path.join(path, f))
    return results

# οι υποφάκελοι που θα δημιουργηθούν είναι τα κλειδιά του λεξικού και τιμές
# τα είδη των αρχείων (κατάληξη) που θα αντιγραφούν ανά υποφάκελο, σε λεξικό
extensions = {
    "python": (".py", ".ipynb"),
    "excel": (".xlsx", ".csv"),
    "text": (".txt",),
}
subdirs = list(extensions.keys())


# Επιμονή μέχρι να δοθεί σωστό μονοπάτι ή κενή συμβολοσειρά
path_exists = False
while not path_exists:
    target_path = input("Πού να δημιουργηθεί ο φάκελος backup;")
    if target_path == "":
        exit(0)
    path_exists = create_backup_dir(target_path, subdirs)

# Πού βρισκόμαστε;
workdir = os.getcwd()
print(f"Αντιγραφή αρχείων από: {workdir}.")

# Αντιγραφή αρείων σε υποφακέλους ανά είδος αρχείου (κατάληξη)
for subdir in subdirs:
    for ext in extensions[subdir]:
        python_files = find_files(workdir, ext)
        for p_file in python_files:
            shutil.copy2(p_file, os.path.join(target_path, "backups/" + subdir))
```
Κ. 236 – Δημιουργία φακέλων αντιγράφων ασφαλείας και αντιγραφή αρχείων σε αυτούς.

Λύσεις ασκήσεων κεφαλαίου 6

Άσκηση 1

Η λύση της άσκησης παρουσιάζεται στον κώδικα Κ. 237. Στο πρόγραμμα αυτό καλείται ο χρήστης να εισάγει τιμές σε αυστηρά αύξουσα σειρά. Αν αυτό δεν συμβεί τότε ενημερώνεται με μήνυμα και το πρόγραμμα τερματίζει την εκτέλεσή του. Το ίδιο θα συμβεί και αν ο χρήστης εισάγει μια μη

αριθμητική τιμή, για παράδειγμα κάποιο κείμενο, οπότε θα εμφανιστεί μήνυμα σφάλματος και η εκτέλεση θα τερματιστεί. Στη δεύτερη περίπτωση θα έχει συλληφθεί με το `try-except` μια εξαίρεση που θα έχει δημιουργηθεί αυτόματα και όχι με την εντολή `raise` που υπάρχει στον κώδικα.

```
previous = None

try:
    for i in range(1, 6):
        value = float(input(f"Δώσε την τιμή {i}: "))
        if previous is not None and value <= previous:
            raise ValueError(
                f"Σφάλμα: η τιμή {value} δεν είναι μεγαλύτερη από την ({previous})."
            )
        previous = value

except ValueError as e:
    print(e)
else:
    print("Επιτυχής ολοκλήρωση: οι 5 τιμές δόθηκαν σε αυστηρά αύξουσα σειρά.")
```
Κ. 237 – Εισαγωγή 5 τιμών σε αυστηρά αύξουσα σειρά.

Στην έξοδο Ε. 60 παρουσιάζονται τρία παραδείγματα εκτέλεσης.

```
Δώσε την τιμή 1: 1.7
Δώσε την τιμή 2: 2.1
Δώσε την τιμή 3: 3.3
Δώσε την τιμή 4: 4.2
Δώσε την τιμή 5: 5.1
Επιτυχής ολοκλήρωση: οι 5 τιμές δόθηκαν σε αυστηρά αύξουσα σειρά.
```

```
Δώσε την τιμή 1: 1.6
Δώσε την τιμή 2: 2.1
Δώσε την τιμή 3: 1.9
Σφάλμα: η τιμή 1.9 δεν είναι μεγαλύτερη από την (2.1).
```

```
Δώσε την τιμή 1: 1.2
Δώσε την τιμή 2: δύο
could not convert string to float: 'δύο'
```
Ε. 60 – Παραδείγματα εξόδου για 3 διαφορετικά σενάρια: α) ορθή είσοδος τιμών β) λάθος σειρά γ) λάθος τιμή.

Άσκηση 2

Στον κώδικα Κ. 238 παρουσιάζεται η λύση της άσκησης 2 του κεφαλαίου 6, ενώ τα αποτελέσματα μιας εκτέλεσης του κώδικα φαίνονται στην έξοδο Ε. 61. Λόγω της τυχαιότητας, και λόγω του ότι δεν έχει οριστεί κάποιο seed κάθε εκτέλεση θα παρουσιάζει διαφορετικές τιμές. Αν η εντολή `random.seed(42)` πάψει να είναι «σχόλιο», τότε τα αποτελέσματα σε κάθε εκτέλεση θα είναι τα ίδια.

```
import random

# random.seed(42)

def roll_die(n):
```

```
    c = [0] * 6
    for _ in range(n):
        c[random.randint(1, 6) - 1] += 1
    return [x * 100 / n for x in c]

for n in (100, 1000, 10000):
    freqs = roll_die(n)
    print(f"n={n:<5} " + " ".join(f"{i+1}:{f:6.2f}%" for i, f in enumerate(freqs)))
```

Κ. 238 – Υπολογισμός ποσοστών εμφάνισης πλευρών ζαριού για διάφορα πλήθη ρίψεων.

```
n=100   1: 17.00% 2: 13.00% 3: 25.00% 4: 16.00% 5:  9.00% 6: 20.00%
n=1000  1: 18.10% 2: 15.80% 3: 17.50% 4: 16.10% 5: 15.80% 6: 16.70%
n=10000 1: 16.51% 2: 16.67% 3: 16.74% 4: 16.64% 5: 17.06% 6: 16.38%
```

Ε. 61 – Ποσοστά εμφάνισης κάθε πλευράς ενός «ηλεκτρονικού ζαριού» για διαφορετικά πλήθη ρίψεων.

Άσκηση 3

Ο κώδικας Κ. 239 εμφανίζει το όνομα της ημέρας για μια ημερομηνία που δίνει ο χρήστης. Επίσης, εμφανίζει την μελλοντική ημερομηνία με την ίδιο αριθμό ημέρας και μήνα που θα έχει και το ίδιο όνομα ημέρας.

```
from datetime import date

weekday_names = [
    "Δευτέρα",
    "Τρίτη",
    "Τετάρτη",
    "Πέμπτη",
    "Παρασκευή",
    "Σάββατο",
    "Κυριακή",
]

s = input("Δώσε ημερομηνία (ηη/μμ/εεεε): ")
day, month, year = map(int, s.split("/"))

d = date(year, month, day)
w = d.weekday()

print(f"Η {d.strftime('%d/%m/%Y')} είναι {weekday_names[w]}.")

y = year + 1
while True:
    try:
        d2 = date(y, month, day)
    except ValueError:
        y += 1
        continue
```

```
    if d2.weekday() == w:
        print(
            f"Η ίδια ημέρα και μήνας ({day}/{month}) θα ξαναπέσει {weekday_names[w]} στις
{d2.strftime('%d/%m/%Y')}."
        )
        break
    y += 1
```
Κ. 239 – Εντοπισμός ονόματος ημέρας ημερομηνίας.

Ένα παράδειγμα εκτέλεσης του κώδικα παρουσιάζεται στη συνέχεια. Ο χρήστης εισάγει την ημερομηνία 4/2/2026. Το πρόγραμμα εμφανίζει ότι η ημερομηνία 4 Φεβρουαρίου του 2026 είναι Τετάρτη και ότι η επόμενη Τετάρτη που θα είναι 4 Φεβρουαρίου θα είναι στο έτος 2032.

```
Δώσε ημερομηνία (ηη/μμ/εεεε): 4/2/2026
Η 04/02/2026 είναι Τετάρτη.
Η ίδια ημέρα και μήνας (4/2) θα ξαναπέσει Τετάρτη στις 04/02/2032.
```
Ε. 62 – Αποτέλεσμα εκτέλεσης για μια είσοδο του χρήστη.

Λύσεις ασκήσεων κεφαλαίου 7

Άσκηση 1

Ο κώδικας Κ. 240 ορίζει την κλάση Book, δημιουργεί δύο αντικείμενά της, προσθέτει υποθετικές βαθμολογίες αξιολογήσεων των βιβλίων και εκτυπώνει τα δύο βιβλία.

```
class Book:
    def __init__(self, title, isbn):
        self.title = title
        self.isbn = isbn
        self.ratings = []

    def add_rating(self, score):
        if 1 <= score <= 5:
            self.ratings.append(score)
        else:
            print("Η βαθμολογία πρέπει να είναι από 1 έως 5.")

    def average_rating(self):
        if not self.ratings:
            return "Δεν υπάρχουν βαθμολογίες"
        return sum(self.ratings) / len(self.ratings)

    def __str__(self):
        avg = self.average_rating()
        return f"Τίτλος: {self.title}, ISBN: {self.isbn}, Μέση βαθμολογία: {avg}"
```

```
book1 = Book("Ο Μικρός Πρίγκιπας", "978-960-453-083-7")
book2 = Book("Η Φόνισσα", "978-960-01-0547-7")
book1.add_rating(5)
book1.add_rating(4)
book2.add_rating(4)
book2.add_rating(4)
print(
    book1
)  # Τίτλος: Ο Μικρός Πρίγκιπας, ISBN: 978-960-453-083-7, Μέση βαθμολογία: 4.5
print(book2)  # Τίτλος: Η Φόνισσα, ISBN: 978-960-01-0547-7, Μέση βαθμολογία: 4.0
```
Κ. 240 – Ορισμός κλάσης Book και δημιουργία αντικειμένων της.

Άσκηση 2

Ο κώδικας Κ. 241 ορίζει τρεις κλάσεις, την υπερκλάση Device, και τις δύο υποκλάσεις της, Laptop και TV. Στη συνέχεια δημιουργούνται τρία αντικείμενα (ένα Device, ένα Laptop και ένα αντικείμενο TV) και ως παράδειγμα πολυμορφισμού καλείται η μέθοδος turn_on() για κάθε αντικείμενο.

```
class Device:
    def __init__(self, brand, power):
        self.brand = brand
        self.power = power

    def turn_on(self):
        return f"Η συσκευή {self.brand} ισχύος {self.power}W ενεργοποιήθηκε."

    def __repr__(self):
        return f"Device(brand={self.brand!r}, power={self.power})"


class Laptop(Device):
    def __init__(self, brand, power, battery_life):
        super().__init__(brand, power)
        self.battery_life = battery_life

    def turn_on(self):
        return f"Ο φορητός υπολογιστής {self.brand} ενεργοποιήθηκε."

    def __repr__(self):
        return f"Laptop(brand={self.brand!r},
power={self.power},battery_life={self.battery_life})"


class TV(Device):
    def __init__(self, brand, power, screen_size):
        super().__init__(brand, power)
```

```
        self.screen_size = screen_size

    def turn_on(self):
        return f"Η τηλεόραση {self.brand} {self.screen_size} ενεργοποιήθηκε."

    def __repr__(self):
        return f"TV(brand={self.brand!r}, power={self.power},
screen_size={self.screen_size})"


d1 = Device("Philips", 100)
l1 = Laptop("Dell", 65, 10)
t1 = TV("Samsung", 150, 55)

print(d1)  # Device(brand='Philips', power=100)
print(d1.turn_on())  # Η συσκευή Philips ισχύος 100W ενεργοποιήθηκε.
print(l1)  # Laptop(brand='Dell', power=65,battery_life=10)
print(l1.turn_on())  # Ο φορητός υπολογιστής Dell ενεργοποιήθηκε.
print(t1)  # TV(brand='Samsung', power=150, screen_size=55)
print(t1.turn_on())  # Η τηλεόραση Samsung 55 ενεργοποιήθηκε.
```
Κ. 241 – Άσκηση με κληρονομικότητα.

Άσκηση 3

Στον κώδικα Κ. 242 ορίζεται η κλάση CPU και η κλάση Computer που έχει ως χαρακτηριστικό ένα αντικείμενο της κλάσης CPU. Αυτό αποτελεί ένα παράδειγμα σύνθεσης (composition) καθώς το αντικείμενο Computer «έχει-ένα» αντικείμενο CPU.

```
class CPU:
    def __init__(self, model, speed):
        self.model = model
        self.speed = speed

    def process(self):
        print(f"Η CPU {self.model} στα {self.speed}GHz επεξεργάζεται δεδομένα...")


class Computer:
    def __init__(self, brand, ram, cpu):
        self.brand = brand
        self.ram = ram
        self.cpu = cpu

    def start(self):
        print(f"Εκκίνηση υπολογιστή {self.brand} με {self.ram}GB RAM...")
```

```
        self.cpu.process()


cpu = CPU("Intel i7", 3.4)
my_computer = Computer("Dell", 16, cpu)
my_computer.start() # Εκκίνηση υπολογιστή Dell με 16GB RAM...
                    # Η CPU Intel i7 στα 3.4GHz επεξεργάζεται δεδομένα...
```
Κ. 242 – Άσκηση με σύνθεση (composition).

ΠΑΡΑΡΤΗΜΑ Β' - ΑΠΑΝΤΗΣΕΙΣ ΕΡΩΤΗΣΕΩΝ ΚΛΕΙΣΤΟΥ ΤΥΠΟΥ

Απαντήσεις για τις ερωτήσεις του κεφαλαίου 1:

Ερώτηση 1	*Ερώτηση 2*	*Ερώτηση 3*	*Ερώτηση 4*	*Ερώτηση 5*
C	**C**	**B**	**D**	**A**

Απαντήσεις για τις ερωτήσεις του κεφαλαίου 2:

Ερώτηση 1	*Ερώτηση 2*	*Ερώτηση 3*	*Ερώτηση 4*	*Ερώτηση 5*
A	**C**	**A**	**A**	**D**

Απαντήσεις για τις ερωτήσεις του κεφαλαίου 3:

Ερώτηση 1	*Ερώτηση 2*	*Ερώτηση 3*	*Ερώτηση 4*	*Ερώτηση 5*
B	**A**	**B**	**B**	**C**

Απαντήσεις για τις ερωτήσεις του κεφαλαίου 4:

Ερώτηση 1	*Ερώτηση 2*	*Ερώτηση 3*	*Ερώτηση 4*	*Ερώτηση 5*
B	**B**	**B**	**D**	**A**

Απαντήσεις για τις ερωτήσεις του κεφαλαίου 5:

Ερώτηση 1	*Ερώτηση 2*	*Ερώτηση 3*	*Ερώτηση 4*	*Ερώτηση 5*
C	**B**	**D**	**A**	**B**

Απαντήσεις για τις ερωτήσεις του κεφαλαίου 6:

Ερώτηση 1	*Ερώτηση 2*	*Ερώτηση 3*	*Ερώτηση 4*	*Ερώτηση 5*
C	**B**	**C**	**A**	**D**

Απαντήσεις για τις ερωτήσεις του κεφαλαίου 7:

Ερώτηση 1	*Ερώτηση 2*	*Ερώτηση 3*	*Ερώτηση 4*	*Ερώτηση 5*
B	**B**	**C**	**A**	**D**

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. A byte of Python by Swaroop - https://python.swaroopch.com/
2. A Whirlwind Tour of Python by Jake VanderPlas - https://github.com/jakevdp/WhirlwindTourOfPython
3. Automate the boring stuff with Python by Al Sweigart - https://automatetheboringstuff.com/
4. Introduction to Python Programming by openstax - https://openstax.org/details/books/introduction-python-programming
5. Learn Python 3 - https://animator.github.io/learn-python/
6. The Hitchhiker's Guide to Python! - https://docs.python-guide.org/
7. Think Python: How to think like a computer scientist by Allen B. Downey - https://greenteapress.com/thinkpython2/html/index.html
8. Κανίδης Ε., Μακρυγιάννης Π., Χατζηπαπαδόπουλος Α. (2024). ΕΕΛΛΑΚ-Εισαγωγή στον προγραμματισμό και την αλγοριθμική με τη γλώσσα προγραμματισμού Python, https://nationaldigitalacademy.gov.gr/mathimata/episthmh-ypologistwn-5/eisagwgh-ston-programmatismo-kai-thn-algorithmikh-me-th-glwssa-programmatismou-python-387
9. Μαγκούτης, Κ., & Νικολάου, Χ. (2015). Εισαγωγή στον Αντικειμενοστραφή Προγραμματισμό με Python [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. https://dx.doi.org/10.57713/kallipos-829
10. Μανής, Γ. (2015). Εισαγωγή στον Προγραμματισμό με αρωγό τη γλώσσα Python [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. https://dx.doi.org/10.57713/kallipos-749
11. Περάκης, Κ., & Δασυγένης, Μ. (2024). Εργαλειοθήκη της Python [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. https://dx.doi.org/10.57713/kallipos-364
12. Παναγιώτου, Γ. (2022). Μια εισαγωγή στην Python για Μεταλλειολόγους & άλλους Μηχανικούς [Προπτυχιακό εγχειρίδιο]. Κάλλιπος, Ανοικτές Ακαδημαϊκές Εκδόσεις. https://dx.doi.org/10.57713/kallipos-99

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

εκδδα

ΕΘΝΙΚΟ ΚΕΝΤΡΟ ΔΗΜΟΣΙΑΣ
ΔΙΟΙΚΗΣΗΣ & ΑΥΤΟΔΙΟΙΚΗΣΗΣ