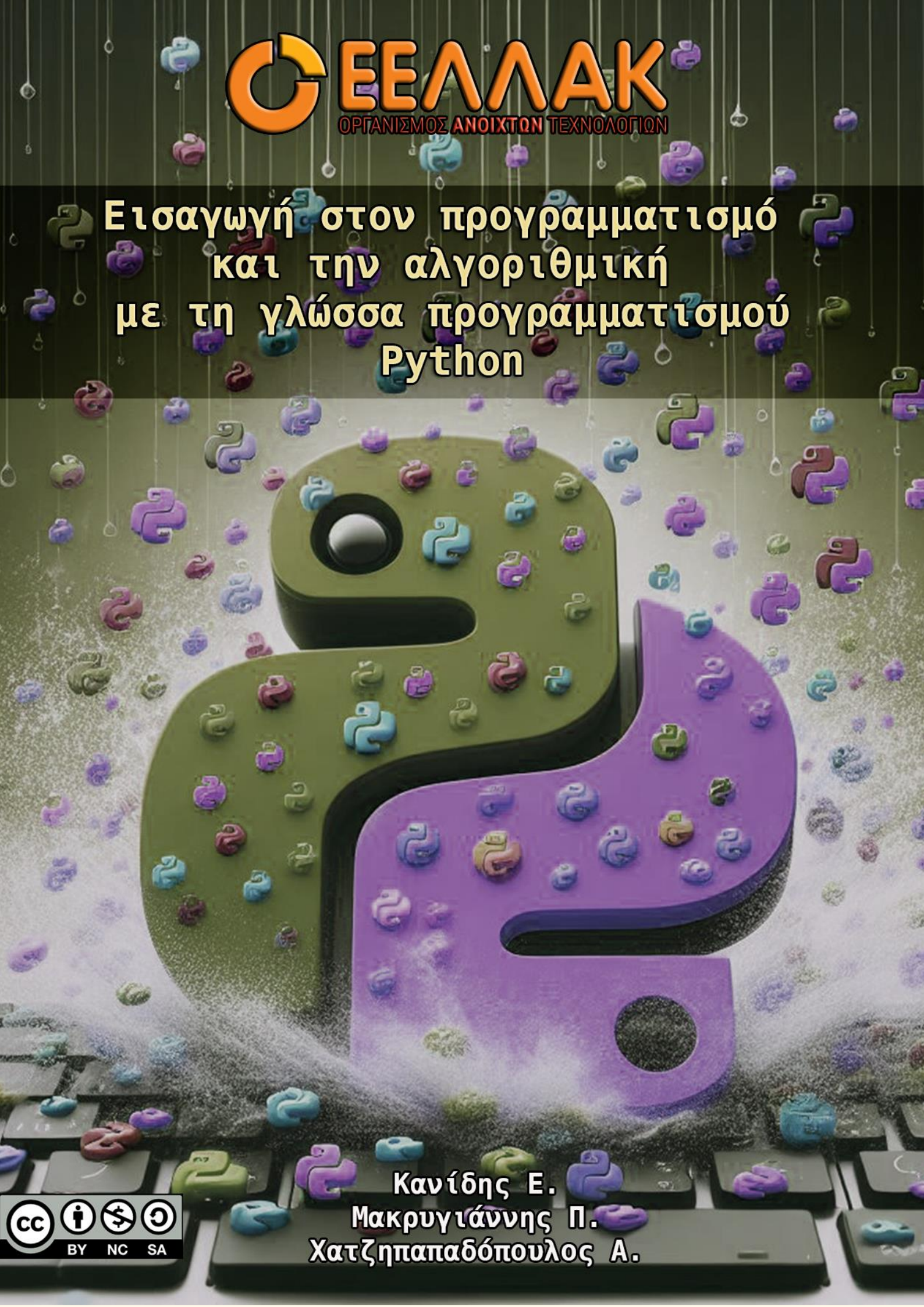
ΕΕΛΛΑΚ
ΟΡΓΑΝΙΣΜΟΣ ΑΝΟΙΧΤΩΝ ΤΕΧΝΟΛΟΓΙΩΝ

Εισαγωγή στον προγραμματισμό και την αλγοριθμική με τη γλώσσα προγραμματισμού Python



Κανίδης Ε.
Μακρυγιάννης Π.
Χατζηπαπαδόπουλος Α.

ΕΕΛΛΑΚ
ΟΡΓΑΝΙΣΜΟΣ ΑΝΟΙΧΤΩΝ ΤΕΧΝΟΛΟΓΙΩΝ

Εισαγωγή στον προγραμματισμό και την αλγοριθμική με τη γλώσσα προγραμματισμού Python

ΤΕΧΝΙΚΟΣ ΑΝΟΙΧΤΟΥ ΛΟΓΙΣΜΙΚΟΥ Δ. ΙΕΚ

ISBN 978-618-84221-7-9

Κανίδης Ε. - Μακρυγιάννης Π. - Χατζηπαπαδόπουλος Α.

ISBN 978-618-84221-7-9

ΟΜΑΔΑ ΣΥΓΓΡΑΦΗΣ

Κανίδης Ευάγγελος, PhD, MSc, π. Σχολικός Σύμβουλος Πληροφορικής

Μακρυγιάννης Παναγιώτης, MSc, Εκπαιδευτικός Πληροφορικής

Χατζηπαπαδόπουλος Αναστάσιος, MSc, Εκπαιδευτικός Πληροφορικής

ΠΕΡΙΕΧΟΜΕΝΑ

1

1. Από τα προβλήματα στις υπολογιστικές λύσεις 1

1.1 Από τα προβλήματα στις υπολογιστικές λύσεις 3

1.1.1 Επίλυση προβλήματος με Η/Υ 4

1.1.2 Μέθοδοι ανάπτυξης λογισμικού 6

1.1.3 Καλές πρακτικές ανάπτυξης λογισμικού και προγραμματισμού 9

1.2 Αλγόριθμοι & Γλώσσες Προγραμματισμού 12

1.2.1 Υποδείγματα προγραμματισμού και γλώσσες που τους ταιριάζουν 12

1.2.2 Τα χαρακτηριστικά της Python 13

1.3 Να θυμάμαι 15

1.4 Ερωτήσεις Κατανόησης 16

1.5 Ερωτήσεις Ανάπτυξης 17

19

2. Γνωριμία με τη γλώσσα Python 19

2.1 Εισαγωγή στη γλώσσα Python 21

2.1.1 Ιστορικά στοιχεία και χαρακτηριστικά 21

2.1.2 Λήψη και εγκατάσταση της γλώσσας 21

2.1.3 Εφαρμογές της γλώσσας Python 22

2.1.4 Ενσωματωμένα περιβάλλοντα συγγραφής κώδικα 22

2.2 Εντολή εκτύπωσης της γλώσσας Python 26

2.2.1 Δραστηριότητα 26

2.3 Ομάδες εντολών με την εντολή def της Python 29

2.3.1 Δραστηριότητα 31

2.3.2 Δραστηριότητα 31

2.4 Επίλυση ενός συνηθισμένου προβλήματος 33

2.4.1 Δραστηριότητα 33

2.4.2 Βελτιώσεις στη λύση του προβλήματος μισθοδοσίας 34

2.4.3 Δραστηριότητα 35

2.5 Χρήση εντολής εισόδου δεδομένων input της Python 37

2.5.1 Βελτίωση της λύσης του προβλήματος μισθοδοσίας με τη χρήση εντολών εισόδου 37

2.6 Ενδεικτική Ολοκληρωμένη Λύση στο πρόβλημα της μισθοδοσίας 40

2.7 Να θυμάμαι 43
2.8 Ερωτήσεις....47
2.8.1 Ερωτήσεις κατανόησης 47
2.8.2 Ερωτήσεις ανάπτυξης 49
2.9 Ασκήσεις....49
2.9.1 Λύσεις των Ασκήσεων....51
53
3. Τύποι δεδομένων και εκφράσεις....53
3.1 Μεταβλητές 55
3.2 Τύποι Δεδομένων 56
3.3 Αριθμητικές και λογικές εκφράσεις....58
3.3.1 Αριθμητικοί τελεστές 58
3.3.2 Συγκριτικοί η σχεσιακοί τελεστές....59
3.3.3 Τελεστές λογικών πράξεων (λογικοί τελεστές) 59
3.4 Ακρίβεια πράξεων με πραγματικούς αριθμούς (float) στην Python....61
3.5 Δημιουργία και αποτίμηση εκφράσεων....62
3.6 Σχόλια εντολών ή μιας γραμμής και σχόλια πολλαπλών γραμμών 63
3.7 Δραστηριότητες....65
3.7.1 Δραστηριότητα....65
3.7.2 Δραστηριότητα....65
3.7.3 Δραστηριότητα....65
3.7.4 Δραστηριότητα....65
3.8 Να θυμάμαι 66
67
4. Αρθρώματα Λογισμικού, Συναρτήσεις 67
4.1 Αντικείμενα στη γλώσσα Python....69
4.2 Συναρτήσεις στη γλώσσα Python....70
4.2.1 Η χρησιμότητα και τα πλεονεκτήματα των συναρτήσεων....72
4.3 Ενσωματωμένες συναρτήσεις στο περιβάλλον της γλώσσας....73
4.4 Αρθρώματα Λογισμικού....81
4.4.1 Φόρτωση ενσωματωμένων αρθρωμάτων 82
4.4.2 Δραστηριότητα....87

4.4.3 Δραστηριότητα 87
4.5 Δημιουργία, Δομή και λειτουργία συναρτήσεων χρήστη 88
4.5.1 Δημιουργία συναρτήσεων χωρίς επιστροφή και χωρίς παραμέτρους 89
4.5.2 Δημιουργία συναρτήσεων χωρίς επιστροφή με παραμέτρους 89
4.5.3 Δραστηριότητα 90
4.5.4 Δημιουργία συνάρτησης με παραμέτρους και επιστροφή τιμής 90
4.5.5 Κλήση συνάρτησης με ορίσματα εκφράσεις ή συναρτήσεις 93
4.5.6 Δραστηριότητα 93
4.5.7 Δραστηριότητα 94
4.5.8 Δραστηριότητα 95
4.5.9 Κλήση συναρτήσεων με προεπιλεγμένες παραμέτρους 95
4.5.10 Κλήση συνάρτησης από πρόγραμμα ή από άλλη συνάρτηση 96
4.6 Δημιουργία και φόρτωση αρθρώματος λογισμικού του χρήστη 100
4.6.1 Δραστηριότητα 108
4.6.2 Δραστηριότητα 109
4.6.3 Δραστηριότητα 109
4.6.4 Δραστηριότητα 110
4.7 Αφαιρετικότητα 111
4.7.1 Δραστηριότητα 113
4.8 Εμβέλεια μεταβλητών 114
4.8.1 Δραστηριότητες 117
4.9 Να θυμάμαι 119
4.10 Ερωτήσεις Κατανόησης 123
4.11 Ερωτήσεις Ανάπτυξης 126
4.12 Ασκήσεις 127
129
5. Υλοποίηση Βασικών προγραμματιστικών Δομών 129
5.1 Εισαγωγή 131
5.2 Λογικές εκφράσεις ή λογικές συνθήκες 132
5.3 Αλγοριθμικές δομές 132
5.4 Δομή ακολουθίας 133
5.5 Δομές επιλογής 134

5.5.1 Η απλή δομή if 134
5.5.2 Σύνθετη δομή if (if ...else) 135
5.5.3 Πολλαπλή if (if...elif...elif..else) 136
5.5.4 Εμφωλευμένα if (nested if) 137
5.5.5 Δραστηριότητα 138
5.6 Δομές επανάληψης (βρόχοι - loops) 139
5.6.1 Δομή επανάληψης ορισμένου πλήθους επαναλήψεων (for) 139
5.6.2 Δραστηριότητα 141
5.6.3 Δραστηριότητα 141
5.6.4 Δομή επανάληψης υπό προϋπόθεση (while) 142
5.6.5 Εμφώλευση δομών επανάληψης 145
5.7 Να θυμάμαι 148
5.8 Ερωτήσεις Κατανόησης 150
5.9 Ασκήσεις 151
153
6. Δομή Συμβολοσειράς, Αντικείμενα, Αναζήτηση 153
6.1 Οι συμβολοσειρές 155
6.1.1 Δομές Δεδομένων και συμβολοσειρές 155
6.1.2 Γενικά Χαρακτηριστικά των συμβολοσειρών 156
6.1.3 Εισαγωγή δεδομένων συμβολοσειρών 156
6.2 Αντικείμενα 159
6.2.1 Χαρακτηριστικά των Αντικειμένων 160
6.2.2 Δραστηριότητα, ταυτότητα & τύπος αντικειμένων 162
6.2.3 Αντικείμενα και συναρτήσεις μετατροπής τύπου str, int, float, bool 162
6.2.4 Δραστηριότητα, συναρτήσεις μετατροπής τύπων 164
6.3 Η δομή και οι δείκτες της συμβολοσειράς 166
6.3.1 Δραστηριότητα, δείκτες συμβολοσειράς 167
6.4 Πράξεις και τελεστές στις συμβολοσειρές 168
6.4.1 Δραστηριότητα, πράξεις και τελεστές στις συμβολοσειρές 171
6.4.2 Δραστηριότητα, πράξεις και τελεστές στις συμβολοσειρές 171
6.5 Διάσχιση συμβολοσειράς 172
6.5.1 Δραστηριότητα (διάσχιση συμβολοσειρών) 173

6.6 Κωδικοποίηση συμβολοσειρών 174
6.6.1 Δραστηριότητα, συναρτήσεις, ord, chr 174
6.7 Αντικείμενα και λειτουργικότητα 176
6.8 Λειτουργικότητα συμβολοσειρών 177
6.8.1 Μέθοδοι συμβολοσειρών upper(), lower(), title() 177
6.8.2 Δραστηριότητα, μέθοδοι upper(), lower() 178
6.9 Πρόβλημα. Απόδοση Προσωρινών Διαπιστευτηρίων 179
6.9.1 Επίλυση του προβλήματος, Απόδοση Προσωρινών Διαπιστευτηρίων 179
6.10 Αναζήτηση 181
6.10.1 Μέθοδοι αναζήτησης της Python για τις συμβολοσειρές 183
6.10.2 Δραστηριότητες (Αναζήτηση) 184
6.11 Πρόβλημα Α. Στοίχιση πρότασης 184
6.12 Καθαρισμός, συνένωση, διαχωρισμός, αντικατάσταση σε συμβολοσειρές 187
6.12.1 Καθαρισμός συμβολοσειρών μέθοδοι lstrip(), rstrip(), strip() 187
6.12.2 Συνένωση, διαχωρισμός συμβολοσειρών μέθοδοι join(), split() 188
6.13 Ιεραρχική Σχεδίαση και τμηματικός προγραμματισμός 190
6.13.1 Πρόβλημα τηλεφωνικών αριθμών 190
6.13.2 Δραστηριότητα 194
6.14 Πρόβλημα αντικατάστασης κειμένου 195
6.14.1 Εύρεση και αντικατάσταση, μέθοδος replace() 198
6.14.2 Δραστηριότητα 198
6.15 Να θυμάμαι 199
6.16 Ερωτήσεις Κατανόησης 203
6.17 Ερωτήσεις Ανάπτυξης 206
6.18 Ασκήσεις 207
211
7. Ενσωματωμένες σύνθετες δομές δεδομένων 211
7.1 Σύνθετες δομές δεδομένων 213
7.2 Η δομή της πλειάδας 214
7.2.1 Στοιχεία μιας πλειάδας (πρόσβαση, διάσχιση) 216
7.2.2 Βασικές λειτουργίες πλειάδας, πράξεις και συναρτήσεις που εφαρμόζονται σε αυτή 217
7.2.3 Να θυμάμαι 221

7.2.4 Ερωτήσεις Κατανόησης 222
7.2.5 Ερωτήσεις Ανάπτυξης 223
7.2.6 Ασκήσεις 224
7.3 Η δομή της λίστας 225
7.3.1 Στοιχεία μιας λίστας (πρόσβαση, διάσχιση) 227
7.3.2 Βασικές λειτουργίες λίστας, πράξεις και συναρτήσεις που εφαρμόζονται σε αυτή. 228
7.3.3 Να θυμάμαι 234
7.3.4 Ερωτήσεις Κατανόησης 236
7.3.5 Ερωτήσεις Ανάπτυξης 237
7.3.6 Ασκήσεις 238
7.4 Η δομή του λεξικού 239
7.4.1 Στοιχεία ενός λεξικού (πρόσβαση, διάσχιση) 241
7.4.2 Βασικές λειτουργίες λεξικού, πράξεις και συναρτήσεις που εφαρμόζονται σε αυτό 243
7.4.3 Να θυμάμαι 246
7.4.4 Ερωτήσεις Κατανόησης 248
7.4.5 Ερωτήσεις Ανάπτυξης 249
7.4.6 Ασκήσεις 250
7.5 Η δομή του συνόλου 251
7.5.1 Στοιχεία ενός συνόλου (πρόσβαση, διάσχιση) 253
7.5.2 Βασικές λειτουργίες συνόλου, πράξεις και συναρτήσεις που εφαρμόζονται σε αυτό 254
7.5.3 Να θυμάμαι 259
7.5.4 Ερωτήσεις Κατανόησης 261
7.5.5 Ερωτήσεις Ανάπτυξης 262
7.5.6 Ασκήσεις 263
265
8. Λάθη, Παγίδευση Λαθών, Αρχεία, Διαχείριση Αρχείων 265
8.1 Λάθη στον κώδικα 267
8.2 Χειρισμός λαθών στην Python 269
8.2.1 Μηνύματα λαθών στην Python 269
8.2.2 Συντακτικά Λάθη (Syntax Errors) 269
8.2.3 Λάθη χρόνου εκτέλεσης (Run Time Errors) 269
8.2.4 Λογικά Λάθη 270

8.2.5 Δραστηριότητα 271
8.2.6 Εντολές παγίδευσης/διαχείρισης λαθών και αμυντικός προγραμματισμός 271
8.2.7 Δραστηριότητα, παράδειγμα σύνθετου αμυντικού προγραμματισμού 275
8.3 Κύρια και δευτερεύουσα μνήμη 277
8.4 Αρχεία κειμένου 278
8.4.1 Διαχείριση αρχείων κειμένου 278
8.4.2 Εύρεση τρέχοντος καταλόγου 280
8.4.3 Ανάγνωση των περιεχομένων ενός αρχείου κειμένου μέθοδοι .read(), .seek(), .tell() 281
8.4.4 Άνοιγμα για ανάγνωση των περιεχομένων ενός αρχείου κειμένου ανά γραμμή 283
8.4.5 Άνοιγμα αρχείου και προσάρτηση περιεχομένων 286
8.4.6 Ανάγνωση αρχείου και επεξεργασία περιεχομένου ανά χαρακτήρα 290
8.4.7 Εγγραφή σε αρχεία κειμένου, μέθοδος write() 292
8.4.8 Πρόβλημα, εγγραφή βαθμολογίων σπουδαστή σε αρχείο 293
8.4.9 Επιλυμένη δραστηριότητα, εισαγωγή βαθμολογίας προόδου σπουδαστών 294
8.5 Κωδικοποίηση των αρχείων κειμένου 298
8.5.1 Κωδικοποίηση αρχείων σε διαφορετικά ΛΣ 298
8.5.2 (**) Δραστηριότητα, αποθήκευση περιεχομένων αρχείου κειμένου σε διαφορετική κωδικοποίηση 300
8.5.3 Συντάκτες Κειμένου & Κωδικοποιήσεις 302
8.6 Άρθρωμα λογισμικού os 304
8.6.1 Δραστηριότητα - Εφαρμογές OS 304
8.7 Επιλυμένη - Κλιμακωτή Εφαρμογή 305
8.8 Να θυμάμαι 313
8.9 Ερωτήσεις Κατανόησης 318
8.10 Ερωτήσεις Ανάπτυξης 320
8.11 Ασκήσεις 321
9. ΒΙΒΛΙΟΓΡΑΦΙΑ - ΙΣΤΟΓΡΑΦΙΑ 322
9.1 Βιβλιογραφία 324
9.2 Ιστογραφία 324

Κεφάλαιο 1

1. Από τα προβλήματα στις υπολογιστικές λύσεις

Τελειώνοντας αυτό το μάθημα θα έχεις μάθει

- Τι σημαίνει υπολογιστική λύση
- Τα στάδια επίλυσης προβλήματος με Η/Υ
- Τις βασικές μεθόδους ανάπτυξης λογισμικού
- Τα πλεονεκτήματα διάφορων μεθόδων ανάπτυξης λογισμικού
- Να χρησιμοποιείς καλές πρακτικές ανάπτυξης λογισμικού
- Τα βασικά προγραμματιστικά υποδείγματα τις γλώσσες που τα υπηρετούν
- Τις λογικές συγγραφής προγράμματος ανάλογα με το είδος προγραμματισμού
- Μερικά βασικά χαρακτηριστικά της γλώσσας python

1.1 Από τα προβλήματα στις υπολογιστικές λύσεις

Στην καθημερινότητα των ανθρώπων προκύπτουν μικρά ή μεγάλα προβλήματα που απαιτούν λύση. Το ίδιο ισχύει φυσικά και στην διάρκεια της κατάρτισης των σπουδαστών. Όλοι έχουμε συναντήσει προβλήματα μαθηματικών, επιστήμης, τεχνολογίας. Τα προβλήματα αυτά έχουν συνήθως μια αυξημένη πολυπλοκότητα και πρέπει απλοποιηθούν πριν λυθούν. Όμως τι εννοούμε εδώ με τον όρο πρόβλημα;

Πρόβλημα χαρακτηρίζεται κάθε κατάσταση η οποία χρήζει αντιμετώπισης, απαιτεί λύση, αλλά η λύση της δεν είναι γνωστή, ούτε προφανής. Υπάρχουν προβλήματα των οποίων η λύση είναι εύκολο να βρεθεί, όμως υπάρχουν και άλλα που η λύση τους δεν είναι εύκολο να βρεθεί. Σε αυτή την περίπτωση λέμε ότι η λύση δεν είναι τετριμμένη.

Παραδείγματα τέτοιων «δύσκολων» προβλημάτων είναι: η ρύπανση του περιβάλλοντος, η αντιμετώπιση των ναρκωτικών, η θεραπεία ασθενειών, η ασφαλής πλοήγηση στο διαδίκτυο αλλά ακόμα και η δημιουργία μιας βάσης δεδομένων που διαχειρίζεται πολλά και ανόμοια δεδομένα.

Ένα πρόβλημα δύσκολο, που η λύση του δεν είναι τετριμμένη γιατί σε αυτή συμμετέχουν πολλοί παράγοντες οι οποίοι πιθανά αλληλεπιδρούν και μεταξύ τους χαρακτηρίζεται ως σύνθετο. Μερικά σύνθετα προβλήματα μπορούν να επιλυθούν με τη βοήθεια ηλεκτρονικού υπολογιστή και για αυτό χαρακτηρίζονται υπολογιστικά. Η επίλυση ενός υπολογιστικού προβλήματος είναι συχνά προσεγγιστική.

Στην πραγματικότητα υπάρχουν τριών ειδών προβλήματα που απασχολούν τους επιστήμονες και τους μηχανικούς: θεωρητικά, πειραματικά και υπολογιστικά. Από αυτά είναι κυρίως η τρίτη κατηγορία που απασχολεί τους τεχνικούς και ιδιαίτερα τους τεχνικούς εφαρμογών λογισμικού.

1.1.1 Επίλυση προβλήματος με Η/Υ

Η επίλυση ενός προβλήματος με ηλεκτρονικό υπολογιστή μπορεί, ανάλογα με το πρόβλημα, μπορεί να είναι αρκετά περίπλοκη. Γίνεται όμως απλούστερη αναλύοντάς τη σε επιμέρους στάδια. Υπάρχουν διάφορες εκδοχές για τα στάδια αυτά αλλά όλες περιγράφουν, λίγο – πολύ, την ίδια διαδικασία. Ακολουθεί μια αρκετά εξαντλητική παράθεση των σταδίων αυτών:

1. Ορισμός και κατανόηση του προβλήματος

Είναι φυσικό πρώτο στάδιο για την επίλυση ενός προβλήματος να είναι η κατανόησή του. Στο στάδιο αυτό η προσπάθεια εστιάζει στον ξεκάθαρο ορισμό του προβλήματος και τον προσδιορισμό των στοιχείων που το συνθέτουν.

Αυτό το στάδιο είναι συχνά από τα πιο δύσκολα. Ένα πολύ σύνθετο πρόβλημα για να γίνει κατανοητό πρέπει να γίνει κατανοητή η δομή του. Με τον όρο δομή, εννοούμε τα επιμέρους στοιχεία (τμήματα) που αποτελούν το πρόβλημα καθώς και τον τρόπο με τον τρόπο με τον οποίο αυτά συνδέονται και αλληλεπιδρούν. Για την κατανόηση της δομής είναι απαραίτητη η ανάλυση του προβλήματος στα επιμέρους στοιχεία του.

Γενικά με τον όρο Ανάλυση εννοείται ο διαχωρισμός ενός συνόλου στα συστατικά του στοιχεία (decomposition). Για το σκοπό αυτό είναι απαραίτητο το πρόβλημα να μορφοποιηθεί έτσι ώστε να απαλλαγεί από περιττές λεπτομέρειες που αυξάνουν χωρίς λόγο τον όγκο των στοιχείων που πρέπει να διαχειριστούμε. Η λειτουργία αυτή ονομάζεται αφαίρεση (abstraction).

Πολλές φορές πάντως η εμπλοκή του τεχνικού σε αυτό το στάδιο είναι περιορισμένη είτε γιατί το πρόβλημα έχει ήδη αναλυθεί από μηχανικούς ή/ και αναλυτές είτε γιατί μοιάζει πολύ στη δομή του με άλλα προβλήματα που έχουν ήδη επιλυθεί (pattern recognition). Για να είστε σίγουροι ότι ισχύει το τελευταίο πρέπει να έχει προσδιοριστεί με ακρίβεια η μορφή που η έξοδος, δηλαδή η λύση του προβλήματος, πρέπει να έχει. Στην πραγματικότητα μας ενδιαφέρει η μορφή τόσο της εισόδου όσο και της εξόδου (data representation). Για αυτό και αυτό το στάδιο συνδέεται ιδιαίτερα με το επόμενο.

2. Δεδομένα και Ζητούμενα

Στο στάδιο αυτό αναζητάτε αφενός ποια είναι τα βασικά στοιχεία/ δεδομένα που δίνονται στο πρόβλημά μας και αφετέρου τι χρειάζεται για να θεωρηθεί το πρόβλημα λυμένο. Τί πληροφορίες έχετε στη διάθεσή σας και τι χρειαζόσαστε; Είναι οι πληροφορίες κατάλληλες και επαρκής για να δοθεί λύση;

Παράλληλα, όπως αναφέρθηκε και νωρίτερα, πρέπει να αποφασίσετε ποια μορφή θα έχουν, τόσο τα δεδομένα όσο και τα ζητούμενα για να διευκολυνθεί η επίλυση (Data representation).

3. Κατασκευή αλγόριθμου

Αυτή είναι η καρδιά της διαδικασίας. Η συχνά μεγάλη δυσκολία στην κατασκευή κατάλληλου αλγόριθμου μπορεί να προκαλέσει προβλήματα και καθυστερήσεις. Πολλοί συχνά προσπαθούν να το συνδυάσουν είτε με τον έλεγχο και βελτιστοποίηση του αλγόριθμου είτε ακόμα και με την υλοποίησή του για να μειώσουν το χρόνο μέχρι την επίτευξη της λύσης.

Αυτό μπορεί να δουλέψει σε κάποιες περιπτώσεις απλών προβλημάτων, συνήθως όμως οδηγεί σε μεγαλύτερη σπατάλη χρόνου. Η σπατάλη αυτή μπορεί να αφορά στον εντοπισμό και τη διόρθωση λαθών που η ασάφεια του σχεδιασμού καθιστά δυσκολότερη ή ακόμα χειρότερα στην ανάγκη επανασχεδίασης δηλαδή κατασκευής νέου αλγόριθμου όταν η λύση αποκλίνει ιδιαίτερα από την επιθυμητή. Αντίθετα διαθέτοντας επαρκή χρόνο για τη σύνθεση ενός κατάλληλου αλγόριθμου η υλοποίηση και ο έλεγχός της απαιτούν λιγότερο χρόνο.

4. Έλεγχος και βελτιστοποίηση του αλγόριθμου

Συχνά η κατασκευή ενός αλγόριθμου που λύνει το υπό εξέταση πρόβλημα αποκαλύπτει και νέες δυνατότητες. Η ανάλυση της πολυπλοκότητας του αρχικού μας αλγόριθμου και ο έλεγχος για αδυναμίες όπως η διαχείριση ακραίων τιμών ή ειδικών περιπτώσεων μπορεί να οδηγήσει σε ακόμα καλύτερες επιλογές.

Η διόρθωση των αδυναμιών και η μείωση της πολυπλοκότητας ενός αλγόριθμου ονομάζονται βελτιστοποίηση.

Σε κάποιες περιπτώσεις στο στάδιο αυτό μπορεί να εντοπιστεί ακόμα και ένας τελείως διαφορετικός αλγόριθμος που κάνει την ίδια δουλειά με καλύτερο τρόπο. Αυτό μπορεί να αφορά τη σαφήνεια, την ταχύτητα ή τις απαιτήσεις σε μνήμη.

Δυστυχώς και αυτό το στάδιο συχνά παραλείπεται ή ενσωματώνεται στο στάδιο 6, επιβραδύνοντας έτσι τη συνολική διαδικασία.

5. Υλοποίηση αλγόριθμου

Το στάδιο αυτό αφορά στον προγραμματισμό της λύσης. Όσο καλύτερος είναι ο αλγόριθμος που προέκυψε από τα στάδια 3 και 4 τόσο ευκολότερος είναι ο προγραμματισμός, ακόμα και για κάποιον με βασική γνώση της επιλεγμένης γλώσσας προγραμματισμού.

Εδώ πρέπει να γίνει αντιληπτό ότι για πραγματικά καινούρια προβλήματα, που η επίλυσή τους δεν αποτελεί μια αναδιάταξη γνωστών λύσεων απλών επί μέρους προβλημάτων, η επιλογή της γλώσσας προγραμματισμού είναι μέρος της επίλυσης και γίνεται σε αυτό το σημείο. Εξαρτάται σε μεγάλο βαθμό από τα στάδια 2 και 3, και προηγείται όπως είναι φυσικό, του προγραμματισμού.

6. Έλεγχος, εντοπισμός και διόρθωση λαθών

Με την ολοκλήρωση της υλοποίησης είναι ώρα να δοκιμαστεί το πρόγραμμα, χρησιμοποιώντας εισόδους για τις οποίες είναι γνωστές οι έξοδοι που το πρόγραμμα θα έπρεπε να επιστρέφει και συγκρίνοντας τα αποτελέσματα. Αν τα αποτελέσματα είναι διαφορετικά έχει εντοπιστεί ένα λάθος.

Δεν είναι σωστό να υποθέτει κανείς ότι το λάθος είναι στην υλοποίηση, παρότι αυτό είναι συχνότερα η περίπτωση. Μπορεί όμως να υπάρχει ακόμα λάθος στον αλγόριθμο και να χρειαστεί να επιστρέψουμε στο στάδιο 3. Αυτός είναι άλλωστε ο λόγος που συχνά το στάδιο 4 παραλείπεται.

1.1.2 Μέθοδοι ανάπτυξης λογισμικού

Στην πράξη για την ανάπτυξη λογισμικού απαιτείται πιο οργανωμένη μεθοδολογία από την απλή εφαρμογή των σταδίων επίλυσης με Η/Υ, ιδιαίτερα όταν πρόκειται για μεγάλης κλίμακας εφαρμογές. Για το λόγο αυτό τα στάδια που περιγράψαμε στην προηγούμενη ενότητα έχουν εμπνεύσει διάφορες μεθόδους ανάπτυξης λογισμικού. Οι σημαντικότερες από αυτές είναι οι εξής:

1. Μοντέλο καταρράκτη (waterfall):

Μια γραμμική μέθοδος που περιλαμβάνει φάσεις όπως ανάλυση, σχεδιασμό, ανάπτυξη, δοκιμές και και ανάπτυξη/ εφαρμογή. Συχνά προστίθεται μια διαρκής φάση συντήρησης. Το μοντέλο καταρράκτη επιβάλλει τη μετάβαση στην επόμενη φάση μόνο μετά την ολοκλήρωση της προηγούμενης φάσης. Επίσης, υποθέτει την ολοκλήρωση της ανάπτυξης με την πρώτη διάσχιση των βημάτων πλην της συντήρησης.

2. Μοντέλο Σπείρας (Spiral):

Εστιάζει σε διαρκείς επαναλήψεις των βημάτων του μοντέλου του καταρράκτη για βελτίωση, επέκταση και ενσωμάτωση αλλαγών. Κάθε επανάληψη ενσωματώνει περισσότερα στοιχεία του προβλήματος και περιπτώσεις εφαρμογής.

3. Μοντέλο ευέλικτης ανάπτυξης (agile software development):

Μια ευέλικτη προσέγγιση που διαχωρίζει την ανάπτυξη σε μικρά, επαναλαμβανόμενα βήματα που ονομάζονται "επαναλήψεις". Οι απαιτήσεις και οι λύσεις εξελίσσονται διαρκώς απαιτώντας και επιτρέποντας τη συνεργασία μεταξύ ημιαυτόνομων ομάδων εργασίας που έχουν μέσες άκρες τη δομή μιας ολόκληρης ομάδας ανάπτυξης.

4. Προτυποποιημένο Μοντέλο:

Περιλαμβάνει προκαθορισμένα βήματα και διαδικασίες για την ανάπτυξη, όπως το RUP (Rational Unified Process).

5. Ανάπτυξη εξατομικευμένου λογισμικού (RAD):

Μια ταχεία προσέγγιση που επικεντρώνεται στη γρήγορη ανάπτυξη και διάθεση λογισμικού με βάση τις ανάγκες του πελάτη.

6. Επανασχεδιασμός (Re-engineering):

Αφορά την αναδιάρθρωση και βελτίωση υπάρχοντος λογισμικού για να προσαρμοστεί σε νέες απαιτήσεις.

7. Προσανατολισμένο στα αντικείμενα:

Χρησιμοποιεί αντικείμενα ως βασικές μονάδες ανάπτυξης, εστιάζοντας στην αλληλεπίδραση τους, την οποία και περιγράφει.

8. Συνεχής Παράδοση (Continuous Delivery):

Στοχεύει στην αυτοματοποιημένη και συνεχή παράδοση νέων εκδόσεων λογισμικού.

Κάθε μέθοδος έχει τα πλεονεκτήματά της, αλλά η επιλογή της κατάλληλης εξαρτάται από τις απαιτήσεις και τις προτεραιότητες του έργου. Ας δούμε τις τρεις πρώτες μεθόδους λίγο πιο αναλυτικά:

Το μοντέλο καταρράκτη

Το μοντέλο ανάπτυξης λογισμικού καταρράκτη είναι μια παραδοσιακή και σειριακή (γραμμική) προσέγγιση που ακολουθεί μια σειρά φάσεων όπου ολοκληρώνονται προαπαιτούμενες εργασίες πριν προχωρήσει σε επόμενη φάση. Ο όρος "Waterfall" (καταρράκτης) υποδηλώνει το γεγονός ότι οι φάσεις ρέουν φυσικά από τη μία μετά την άλλη, χωρίς επιστροφή σε προηγούμενες φάσεις. Οι βασικές φάσεις της μεθόδου είναι:

Ανάλυση: Καθορισμός των απαιτήσεων του λογισμικού με βάση τις ανάγκες των χρηστών και του πελάτη. Το αποτέλεσμα είναι η ανάπτυξη ενός αναλυτικού και λεπτομερούς σχεδιασμού.

Σχεδιασμός: Σχεδίαση της αρχιτεκτονικής του λογισμικού, των συστατικών του και των διεπαφών τους. Το αποτέλεσμα είναι ένας λεπτομερής σχεδιασμός της δομής και της λειτουργικότητας του λογισμικού.

Υλοποίηση: Ανάπτυξη του λογισμικού βάσει του σχεδιασμού. Οι προγραμματιστές γράφουν τον κώδικα και δημιουργούν τα συστατικά του λογισμικού.

Δοκιμές: Εκτέλεση δοκιμών για την επιβεβαίωση ότι το λογισμικό λειτουργεί σωστά και ανταποκρίνεται στις απαιτήσεις. Περιλαμβάνει δοκιμές μονάδων, ολοκλήρωσης και αποδοχής.

Ανάπτυξη/ εφαρμογή : Ανάπτυξη (deployment) του τελικού προϊόντος στον πελάτη μετά την ολοκλήρωση των φάσεων ανάλυσης, σχεδιασμού, υλοποίησης και δοκιμών.

Η μέθοδος καταρράκτη έχει τα πλεονεκτήματα της δομημένης προσέγγισης και της αυστηρής διαχείρισης του έργου. Ωστόσο, μπορεί να είναι δύσκολο να αντιμετωπίσει αλλαγές απαιτήσεων κατά τη διάρκεια της ανάπτυξης, καθώς κάθε φάση εξαρτάται από την προηγούμενη την οποία και ακολουθεί.

Το μοντέλο της σπείρας

Συνδυάζει στοιχεία από το παραδοσιακό μοντέλο καταρράκτη με την λογική της επανάληψης. Η Το μοντέλο χωρίζεται σε μια σειρά από επαναλήψεις των βημάτων, κάθε μία από τις οποίες αντιπροσωπεύει έναν «κύκλο» ή μια «σπείρα» ανάπτυξης. Κάθε νέα σπείρα ενσωματώνει περισσότερες απαιτήσεις και οδηγεί σε μια πληρέστερη έκδοση του λογισμικού. Οι βασικές φάσεις της μεθόδου είναι:

Σχεδιασμός: Καθορισμός των απαιτήσεων και του σχεδιασμού βάσει της κατανόησης των αναγκών του πελάτη. Σχεδίαση αρχιτεκτονικής, λειτουργικότητας και διεπαφών.

Αξιολόγηση και Ανάλυση Κινδύνων: Αναγνώριση, ανάλυση και αξιολόγηση των πιθανών κινδύνων που μπορεί να επηρεάσουν την ανάπτυξη και την ποιότητα του λογισμικού.

Υλοποίηση και Δοκιμές: Ανάπτυξη του λογισμικού, περιλαμβανομένης της υλοποίησης και των δοκιμών. Κάθε σπείρα μπορεί να περιλαμβάνει μια ή περισσότερες επαναλήψεις αυτού του βήματος.

Αξιολόγηση από τον Πελάτη: Παρουσίαση των εξελίξεων στον πελάτη και αξιολόγηση της απόδοσης του λογισμικού σύμφωνα με τις προσδοκίες του.

Κάθε σπείρα διανύει αυτές τις φάσεις, αλλά με περισσότερη πληροφορία και βελτιώσεις σε σύγκριση με την προηγούμενη δημιουργώντας ένα βελτιωμένο πρωτότυπο του επιδιωκόμενου λογισμικού. Αυτή η δομή επιτρέπει στην ομάδα ανάπτυξης να αντιμετωπίζει και να επιλύει προβλήματα καθώς προχωρά, εξασφαλίζοντας πιο ολοκληρωμένα και ακριβή αποτελέσματα. Αυτή η προσέγγιση βοηθάει να αντιμετωπιστούν αναπάντητες ερωτήσεις και αβεβαιότητες, καθώς και να ενσωματωθούν νέες απαιτήσεις που μπορεί να εμφανιστούν κατά τη διάρκεια της ανάπτυξης. Χρησιμοποιείται καλύτερα για μεγάλα έργα που περιλαμβάνουν συνεχείς βελτιώσεις.

Το μοντέλο ευέλικτης ανάπτυξης (agile)

Το μοντέλο ανάπτυξης λογισμικού Agile είναι μια ευέλικτη και επαναληπτική προσέγγιση που επικεντρώνεται στη συνεργασία, την ανταπόκριση στις αλλαγές και την παράδοση συχνών και μικρών εκδόσεων του λογισμικού. Βασίζεται σε αρχές που προωθούν την ευελιξία, την ομαδική συνεργασία και την ανταπόκριση στις ανάγκες του πελάτη Οι βασικές αρχές του μοντέλου περιλαμβάνουν:

Εξυπηρέτηση του Πελάτη: Το κύριο ζητούμενο είναι να ικανοποιούνται οι ανάγκες του πελάτη, ακόμα και αν αυτές αλλάζουν κατά τη διάρκεια της ανάπτυξης.

Ευελιξία: Η ικανότητα προσαρμογής σε αλλαγές, ακόμα και αργά στον κύκλο ζωή του έργου.Συχνή

Παράδοση: Η παράδοση συχνών, μικρών εκδόσεων του λογισμικού, γνωστών και ως "ίτερα", προκειμένου να αξιολογείται συνεχώς η πρόοδος.

Συνεργασία με τον Πελάτη: Συνεχής επικοινωνία και συνεργασία μεταξύ της ομάδας ανάπτυξης και του πελάτη για την καλύτερη κατανόηση των αναγκών.

Αυτοοργάνωση: Η ομάδα ανάπτυξης έχει την αυτονομία και την ευθύνη να αποφασίζει πώς θα υλοποιήσει τις απαιτήσεις.

Η ευέλικτη (agile) ανάπτυξη χρησιμοποιεί επαναλαμβανόμενες "σφαιρικές" περιόδους ανάπτυξης, γνωστές ως επαναλήψεις, όπου η ομάδα εργάζεται πάνω σε ένα τμήμα των απαιτήσεων, του σχεδιασμού, της υλοποίησης και των δοκιμών. Κάθε ευέλικτη επανάληψη οδηγεί σε ένα αυξημένο επίπεδο λειτουργικότητας του λογισμικού, επιτρέποντας στην ομάδα να λαμβάνει ανατροφοδοσία από τον πελάτη και να ενσωματώνει αλλαγές ή βελτιώσεις. Η ευέλικτη ανάλυση στοχεύει στην ανάπτυξη ενός περιβάλλοντος δημιουργικότητας, ταχείας μάθησης και πειραματισμού προς την κατεύθυνση της καινοτομίας, και μείωσης περιττών εργασιών μεγιστοποιώντας τον όγκο της ερργασίας που εκτελείται σε κάθε επανάληψη. Το μοντέλο ανταποκρίνεται στην ταχεία εξέλιξη των δυνατοτήτων των τεχνολογιών πληροφορικής και επικοινωνιών αλλά και την πολυπλοκότητα των προς επίλυση ζητημάτων, που προκύπτουν κατά την ανάπτυξη ψηφιακών υπηρεσιών. Επιλύει καλά προβλήματα που, δεν επιτρέπουν την εξαρχής κατανόηση των αναγκών και αξιολόγηση των πιθανών λύσεων.

1.1.3 Καλές πρακτικές ανάπτυξης λογισμικού και προγραμματισμού

Το λογισμικό που παράγεται με οποιαδήποτε μεθοδολογία ανάπτυξης πρέπει να μπορεί να βελτιωθεί, να τροποποιηθεί για να εξυπηρετήσει νέες ανάγκες, να αξιολογηθεί εκ νέου για οποιονδήποτε λόγο. Ιδιαίτερα αν πρόκειται για μια μεγάλη εφαρμογή, ένα παιχνίδι ή μια πλατφόρμα πρέπει να υπάρχει πρόβλεψη για εμπλοκή περισσότερων σχεδιαστών, αναλυτών και προγραμματιστών στην ανάπτυξη, αναβάθμιση, επέκταση και τροποποίησή του.

Για το σκοπό αυτό είναι απαραίτητη η σαφήνεια τόσο του σχεδιασμού όσο και της υλοποίησης. Βασικές πρακτικές για τη διασφάλιση αυτής της σαφήνειας είναι η οργανωμένη ανάπτυξη, η τμηματική (modular) ανάπτυξη και η **τεκμηρίωση**. Για την πρώτη χρησιμοποιούμε τις μεθοδολογίες που υλοποιούν τα μοντέλα της προηγούμενης παραγράφου. Η δεύτερη αφορά στην ανάπτυξη ανεξάρτητων τμημάτων που έχουν συγκεκριμένο σκοπό. Η τρίτη αφορά σε μερικά διαφορετικά πράγματα όπως:

1. Η καταγραφή των επιμέρους προβλημάτων στα οποία αναλύθηκε το κύριο πρόβλημα και των επιλογών επίλυσής τους.
2. Η καταγραφή του περιεχομένου των τμημάτων (modules) της εφαρμογής.
3. Η καταγραφή του ρόλου κάθε τμήματος κώδικα στην ευρύτερη εφαρμογή και της δομής του.

Η πρώτη καταγραφή γίνεται έξω από τον κώδικα σε συνοδευτικά έγγραφα που θα χρειαστεί να διαβάσει, διαγώνια έστω, όποιος ενταχθεί με καθυστέρηση στην ομάδα ανάπτυξης ή αναλάβει τροποποιήσεις, αναβαθμίσεις ή διορθώσεις στην εφαρμογή. Η δεύτερη μπορεί να γίνει και πάλι σε έγγραφα ή σε εκτεταμένα σχόλια εντός της εφαρμογής στην αρχή κάθε τμήματος (module). Η τρίτη γίνεται σχεδόν πάντα μέσα στην εφαρμογή με διευκρινιστικά σχόλια σε κάθε ενότητα κώδικα.

Τα σχόλια χρησιμοποιούνται για να επικοινωνήσουν πληροφορίες σχετικά με τη λογική που χρησιμοποιήθηκε στον πηγαίο κώδικα ή την αντιστοίχηση ονομάτων σε διεργασίες και μεταβλητές. Απευθύνονται αφενός σε άλλους προγραμματιστές που πιθανόν να εργαστούν στην εφαρμογή μας και αφετέρου σε εμάς τους ίδιους όταν επιστρέφουμε σε αυτή για να επεκτείνουμε, αναβαθμίσουμε ή διορθώσουμε τον κώδικα. Σκεφτείτε τον χρόνο που χρειάζεται για να κατανοήσετε ή να θυμηθείτε όλα όσα χρειάζεστε για να επέμβετε σε μια εφαρμογή με κάποιες χιλιάδες γραμμές κώδικα.

Σχόλια εντός εφαρμογής

Τα σχόλια εντός εφαρμογής αποτελούνται από γραμμές ή τμήματα γραμμών στον κώδικά μας που αγνοούνται από τους μεταγλωττιστές και τους διερμηνείς. Στην Python υπάρχουν δύο κατηγορίες σχολίων: σχόλια μιας γραμμής και σχόλια πολλαπλών γραμμών.

Τα σχόλια μιας γραμμής στην Python ξεκινούν με το χαρακτήρα hashtag (#) και επεκτείνονται ως το τέλος της γραμμής. Ο χαρακτήρας # αναγνωρίζεται σε οποιαδήποτε θέση αν και όχι μέσα σε μία συμβολοσειρά (string).

```
# Αυτό είναι σχόλιο Python.Συνήθως εξηγεί τι ακολουθεί.
Count = 1 # Άλλο ένα σχόλιο. Αφορά συνήθως ό,τι προηγείται.
Text1 = '# Αυτό δεν είναι σχόλιο. Περιλαμβάνεται σε string'
```

Τα σχόλια πολλαπλών γραμμών μπορούν να δομηθούν με δύο τρόπους. Είτε τοποθετούμε ένα hashtag (#) στην αρχή κάθε γραμμής, είτε περικλείουμε όλο το σχόλιο σε τριπλά αγγλικά διπλά εισαγωγικά (""").
Παρά την ευκολία ο δεύτερος τρόπος έχει τον κίνδυνο να καταλήξουν κάποια από τα σχόλια μας να αποτελούν τυχαία string στη μνήμη του υπολογιστή.

```
""" Αυτό είναι ένα σχόλιο πολλαπλών γραμμών στην Python. Έχει τον κίνδυνο,
π.χ. λόγω μιας τυχαίας διαγραφής, να καταλήξει ως συμβολοσειρά να καταλαμβάνει
χώρο μνήμης. Αυτό μπορεί να είναι πρόβλημα αν το σχόλιο είναι ιδιαίτερα μεγάλο
ή αν συμβεί σε πολλά σχόλια. """

# Αυτό είναι επίσης ένα σχόλιο πολλαπλών γραμμών στην
# Python. Δεν παρουσιάζει όμως το ίδιο πρόβλημα με το
# προηγούμενο.
```

Υπάρχουν ορισμένες ακόμα καλές πρακτικές για την ανάπτυξη λογισμικού, ανάμεσα στις οποίες περιλαμβάνονται οι ακόλουθες:

Ανάλυση και Σχεδιασμός: Διασφαλίζετε ότι οι απαιτήσεις είναι καλά αντιληπτές και το λογισμικό είναι σωστά σχεδιασμένο πριν από την υλοποίηση.

Επιλογή Κατάλληλης Μεθόδου: Επιλέγετε μια μέθοδο ανάπτυξης (καταρράκτη, σπείρα, agile κ.ά.) που ταιριάζει με τις ανάγκες του έργου. Σε κάποιες περιπτώσεις μπορεί να απαιτείται και κάποια ειδικά προσαρμοσμένη προσέγγιση.

Συχνή Παράδοση: Παραδίδετε συχνά νέες εκδόσεις του λογισμικού στον πελάτη για να λαμβάνετε ανατροφοδοσία και να διορθώνετε τυχόν προβλήματα γρήγορα.

Διαχείριση Αλλαγών: Διαχειρίζεστε τις αλλαγές στις απαιτήσεις με τρόπο που να μην επηρεάζεται η ποιότητα του λογισμικού.

Δοκιμές: Διασφαλίζετε ότι το λογισμικό δοκιμάζεται εκτενώς για την ανίχνευση και επίλυση σφαλμάτων πριν από την παράδοση.

Συνεργασία και Επικοινωνία: Διατηρείτε συχνή και ανοιχτή επικοινωνία μεταξύ των μελών της ομάδας ανάπτυξης και με τον πελάτη.

Αυτοματοποίηση: Χρησιμοποιείτε εργαλεία και διαδικασίες αυτοματοποίησης για τη διευκόλυνση των δοκιμών, της ολοκλήρωσης και της παράδοσης.

Καλή Διαχείριση Έκδοσης: Χρησιμοποιείτε συστήματα διαχείρισης εκδόσεων για τον έλεγχο των αλλαγών στον κώδικα και τη διασφάλιση της συνέπειας.

Συνεχής Βελτίωση: Αναλύετε την απόδοση της ομάδας και της διαδικασίας ανάπτυξης, προκειμένου να εντοπίζετε περιθώρια βελτίωσης. Το ίδιο βέβαια ισχύει και για τον κώδικα μετά την παράδοση.

Εκπαίδευση: Ενθαρρύνετε την εκπαίδευση και την ανάπτυξη των μελών της ομάδας για να διατηρείτε τις δεξιότητές τους ενημερωμένες. Εξίσου σημαντική είναι και η εκπαίδευση των χρηστών πάνω στην εφαρμογή.

Οι καλές πρακτικές διαφέρουν ανάλογα με τις ανάγκες και τον τύπο του έργου, αλλά αυτές οι γενικές αρχές μπορούν να βοηθήσουν στη βελτίωση της διαδικασίας ανάπτυξης λογισμικού.

1.2 Αλγόριθμοι & Γλώσσες Προγραμματισμού

1.2.1 Υποδείγματα προγραμματισμού και γλώσσες που τους ταιριάζουν

Υπάρχουν πολλοί τρόποι να προγραμματίσει κανείς. Οι τρόποι αυτοί, που συχνά έχουν σχέση με τις προγραμματιστικές δομές, την οργάνωση των δεδομένων ή τον βαθμό αφαίρεσης, ονομάζονται προγραμματιστικά υποδείγματα. Περιγράφουν τον τρόπο αντιμετώπισης του προβλήματος προς υπολογιστική επίλυση, τον τρόπο διαμόρφωσης και περιγραφής των αλγόριθμων, τον τρόπο διαμόρφωσης κώδικα και την προτεραιότητα σε ενέργειες, έννοιες ή αποτελέσματα. Μερικά βασικά υποδείγματα προγραμματισμού περιγράφονται στη συνέχεια.

Δομημένος Προγραμματισμός: Ο προγραμματισμός γίνεται με βάση τη σειρά εκτέλεσης των εντολών. Οι βασικές προγραμματιστικές δομές περιλαμβάνουν ακολουθιακές δομές, δομές επιλογής, βρόχους επανάληψης και υποπρογράμματα.

Αντικειμενοστρεφής Προγραμματισμός (ΑΟΠ): Η εστίαση είναι στη δημιουργία αντικειμένων, τα οποία συσχετίζονται μεταξύ τους και περιέχουν δεδομένα και συναρτήσεις. Αυτό επιτρέπει την επαναχρησιμοποίηση και τον οργανωμένο σχεδιασμό.

Δηλωτικός Προγραμματισμός: Ορίζει το τι πρέπει να εκτελεστεί, δίνοντας λιγότερη έμφαση στο πώς πρέπει να εκτελεστεί. Το πώς ρυθμίζεται από περιβάλλοντα, μεταγλωττιστές, πρωτόκολλα ή οδηγούς που αφορούν στο συγκεκριμένο κάθε φορά πλαίσιο (υπηρεσία, λειτουργικό ή και υλικό). Είναι φανερό ότι αυτό επιτρέπει την αγνόηση του υποκείμενου λογισμικού και υλικού κατά τον προγραμματισμό. Οι SQL ερωτήσεις σε βάσεις δεδομένων είναι παράδειγμα δηλωτικού προγραμματισμού.

Συναρτησιακός Προγραμματισμός: Βασίζεται στη χρήση συναρτήσεων ως βασικών μονάδων. Οι συναρτήσεις μεταχειρίζονται τα δεδομένα και παρέχουν αποτελέσματα, χωρίς να αλλάζουν την κατάσταση.

Αυτά είναι τα βασικά υποδείγματα προγραμματισμού, τα οποία αντιπροσωπεύουν διάφορες προσεγγίσεις και φιλοσοφίες στον τρόπο που σχεδιάζουμε και αναπτύσσουμε το λογισμικό.

Σκόπιμο είναι να αντιστοιχίσουμε τα υποδείγματα αυτά με κάποιες γλώσσες προγραμματισμού που είναι κατάλληλες για το καθένα από αυτά. Ας δούμε λοιπόν κάποια παραδείγματα :

Δομημένος Προγραμματισμός:

C: Ιδανική για υψηλή απόδοση και γρήγορη εκτέλεση. Συχνά χρησιμοποιείται για ενσωματωμένα συστήματα, λειτουργικά συστήματα και γραφικά προγράμματα.

Fortran: Κυρίως χρησιμοποιείται για επιστημονικούς υπολογισμούς και μαθηματικά προγράμματα.

Αντικειμενοστραφής Προγραμματισμός (ΑΟΠ):

Java: Κατάλληλη για εφαρμογές μεγάλης κλίμακας, δικτυακές εφαρμογές και Android εφαρμογές.

Python: Χρησιμοποιείται για γρήγορη ανάπτυξη, επιστημονικούς υπολογισμούς, web εφαρμογές, και πολλά άλλα.

Δηλωτικός Προγραμματισμός:

SQL: Χρησιμοποιείται για διαχείριση βάσεων δεδομένων και ερωτήματα.

HTML/CSS: Χρησιμοποιούνται για την κατασκευή ιστοσελίδων και τη μορφοποίηση του περιεχομένου.

Συναρτησιακός Προγραμματισμός:

Haskell: Ένα παράδειγμα γλώσσας που υποστηρίζει αποκλειστικά συναρτησιακό προγραμματισμό.

JavaScript: (με λειτουργίες όπως το λάμδα): Χρησιμοποιείται για δυναμικές ιστοσελίδες και εφαρμογές.

Σημειώστε ότι πολλές γλώσσες μπορούν να χρησιμοποιηθούν για διάφορους τύπους προγραμματισμού ανεξαρτήτως του βασικού τους προσανατολισμού. Για παράδειγμα η python αναφέρεται εδώ σε σχέση με τον αντικειμενοστρεφή προγραμματισμό, μπορεί όμως να εξυπηρετήσει και τον δομημένο εξίσου καλά.

1.2.2 Τα χαρακτηριστικά της Python

Η επιλογή της Python ως γλώσσας για αυτό το μάθημα έγινε στη βάση των χαρακτηριστικών της. Αυτά τα χαρακτηριστικά της δίνουν ευκολία χρήσης στην πρώτη επαφή και προσαρμοστικότητα οι οποίες είναι ιδιαίτερα επιθυμητές σε διδακτικό πλαίσιο. Ας δούμε όμως μερικά από αυτά τα χαρακτηριστικά:

Ανοιχτότητα: Η Python είναι γλώσσα ανοικτού λογισμικού. Τόσο η ίδια όσο και οι βιβλιοθήκες της είναι τόσο ελεύθερες (δηλαδή δωρεάν) όσο και, συνήθως, ανοικτές, επιτρέποντας στον προγραμματιστή πρόσβαση στον πηγαίο κώδικα και επομένως στη λογική πίσω από τον προγραμματισμό τους.

Αναγνώσιμος και Καθαρός Κώδικας: Ο κώδικας Python είναι ευανάγνωστος λόγω της έμφασης στη χρήση εσοχών και καλά καθορισμένων κανόνων μορφοποίησης. Υπάρχουν επίσης εύκολοι τρόποι εισαγωγής σχολίων και τεκμηρίωσης του κώδικα.

Δυνατότητα Πολλαπλών Υποδειγμάτων: Οι προγραμματιστές μπορούν να χρησιμοποιήσουν πολλαπλά υποδείγματα για να επιλύσουν προβλήματα αξιοποιώντας διαφορετικές προσεγγίσεις στον προγραμματισμό.

Υψηλού Επιπέδου Δομές Δεδομένων: Η Python παρέχει ενσωματωμένες δομές δεδομένων, όπως λίστες, δείκτες και λεξικά, που διευκολύνουν τη διαχείριση και επεξεργασία δεδομένων.

Υψηλού Επιπέδου Συναρτήσεις: Οι συναρτήσεις μπορούν να έχουν πολλαπλά ορίσματα και να επιστρέφουν πολλαπλές τιμές, διευκολύνοντας την οργάνωση του κώδικα. Παράλληλα λόγω υπερφόρτωσης η «ίδια» συνάρτηση μπορεί να διαχειριστεί διαφορετικά, σε αριθμό και τύπο, ορίσματα με διαφορετικό, κατά περίπτωση, τρόπο.

Δυνατότητα Αυτόματης Διαχείρισης Μνήμης: Η Python διαχειρίζεται τη δέσμευση και αποδέσμευση μνήμης αυτόματα, απαλλάσσοντας τον προγραμματιστή από αυτή την εργασία.

Ευρεία Βιβλιοθήκη Πακέτων: Η Python έχει μια εκτεταμένη βιβλιοθήκη πακέτων που καλύπτει ποικίλες ανάγκες, από επιστημονικούς υπολογισμούς έως γραφικά.

Υποστήριξη Κοινότητας: Η κοινότητα της Python προσφέρει σημαντική υποστήριξη μέσω της παροχής ενημερώσεων, βιβλίων, ενδείξεων και φόρουμ. Συμβάλει δε στην ανάπτυξης της βιβλιοθήκης πακέτων της.

Ανεξαρτησία από την πλατφόρμα: Η Python είναι διαθέσιμη για πολλές πλατφόρμες, συμπεριλαμβανομένων των Windows, macOS, Linux και Android.

Καταλληλότητα για σύγχρονες εφαρμογές: Η Python είναι ιδιαίτερα συμβατή με τη λογική των εγκιβωτισμένων εφαρμογών (containers), ενώ μπορεί να χρησιμοποιηθεί για ανάλυση δεδομένων (data analytics και big data), μηχανική μάθηση κ.ά.

Φυσικά, όπως όλες οι γλώσσες, έτσι και η Python έχει μειονεκτήματα. Για παράδειγμα, μέχρι τη στιγμή που γράφονται αυτές οι γραμμές, οι διαχείριση των νημάτων (threads) γίνεται από τον GIL (Global Interpreter Lock) που είναι γνωστός για την ακαταλληλότητά του για εύκολο πολύ-νηματικό συγχρονισμό. Όμως ο οργανισμός της Python (python.org) έχει ήδη ανακοινώσει την αλλαγή του σχετικού μοντέλου διαχείρισης νημάτων. Δυσκολίες παρουσιάζονται και με την αυθεντική παραλληλία, δηλαδή την εκμετάλλευση πολλαπλών επεξεργαστών ή πυρήνων, τουλάχιστον ως προς την επικοινωνία παράλληλων διεργασιών (processes).

1.3 Να θυμάμαι

Στάδια επίλυσης προβλήματος με Η/Υ

Τα στάδια επίλυσης ενός προβλήματος με Η/Υ είναι:

1. Ορισμός και κατανόηση του προβλήματος
2. Δεδομένα και Ζητούμενα
3. Κατασκευή αλγόριθμου
4. Έλεγχος και βελτιστοποίηση του αλγόριθμου
5. Υλοποίηση αλγόριθμου / προγραμματισμός
6. Έλεγχος, εντοπισμός και διόρθωση λαθών

Μοντέλα ανάπτυξης λογισμικού

Τα κύρια μοντέλα ανάπτυξης λογισμικού περιλαμβάνουν το μοντέλο καταρράκτη (waterfall), το μοντέλο της σπείρας (spiral) και το μοντέλο ευέλικτης ανάπτυξης (agile).

Άλλα μοντέλα περιλαμβάνουν το προτυποποιημένο Μοντέλο, την ανάπτυξη εξατομικευμένου λογισμικού (RAD), το μοντέλο επανασχεδιασμού (Re-engineering), το προσανατολισμένο στα αντικείμενα, το μοντέλο συνεχούς παράδοσης (Continuous Delivery) κ.ά.

Καλές πρακτικές

Οι κυριότερες καλές πρακτικές για την ανάπτυξη λογισμικού είναι η **οργανωμένη ανάπτυξη**, η **τμηματική (modular) ανάπτυξη** και, σημαντικότερη όλων, η **τεκμηρίωση**.

Κατά περίπτωση είναι χρήσιμες και άλλες καλές πρακτικές ανάμεσα στις οποίες η **συχνή παράδοση** τμημάτων του αναπτυσσόμενου προϊόντος, οι οργανωμένες και διεξοδικές **δοκιμές**, η έμφαση στη **διαχείριση αλλαγών και εκδόσεων**, η **συνεχής βελτίωση** της ομάδας ανάπτυξης και του παραγόμενου προϊόντος και η **εκπαίδευση** της ομάδας αλλά και των χρηστών.

Υποδείγματα προγραμματισμού

Τα βασικά υποδείγματα προγραμματισμού περιλαμβάνουν το Δομημένο Προγραμματισμό (ενδεικτικές γλώσσες C και Fortran), τον Αντικειμενοστρεφή Προγραμματισμό (Java και Python), τον Δηλωτικό (SQL και HTML/CSS) και τον Συναρτησιακό Προγραμματισμό (Haskell και Javascript).

Χαρακτηριστικά της Python

Τα κύρια χαρακτηριστικά της Python είναι:

1. Ανοιχτή και ελεύθερη
2. Αναγνωρίσιμος και καθαρός κώδικα,
3. Δυνατότητα πολλαπλών υποδειγμάτων
4. Υψηλού επιπέδου δομές δεδομένων
5. Υψηλού επιπέδου συναρτήσεις
6. Αυτόματη διαχείριση μνήμης
7. Ευρεία βιβλιοθήκη πακέτων
8. Υποστήριξη κοινότητας
9. Ανεξαρτησία από πλατφόρμες

1.4 Ερωτήσεις Κατανόησης

1. Υπάρχουν τριών ειδών προβλήματα που απασχολούν τους επιστήμονες και τους μηχανικούς: θεωρητικά, πειραματικά και υπολογιστικά.

   ΝΑΙ ΟΧΙ

2. Με τον όρο Σχεδίαση εννοείται ο διαχωρισμός ενός συνόλου στα συστατικά του στοιχεία (decomposition).

   ΝΑΙ ΟΧΙ

3. Το στάδιο της κατασκευής αλγορίθμου πολλές φορές παραλείπεται κατά τη διαδικασία επίλυσης προβλήματος με Η/Υ.

   ΝΑΙ ΟΧΙ

4. Η διόρθωση των αδυναμιών και η μείωση της πολυπλοκότητας ενός αλγόριθμου ονομάζονται βελτιστοποίηση.

   ΝΑΙ ΟΧΙ

5. Με τον όρο υλοποίηση αλγόριθμου εννοούμε απλά τον προγραμματισμό της λύσης μας.

   ΝΑΙ ΟΧΙ

6. Το μοντέλο της Σπείρας είναι ένα σειριακό μοντέλο ανάπτυξης λογισμικού.

   ΝΑΙ ΟΧΙ

7. Η μέθοδος RAD εστιάζει στη γρήγορη ανάπτυξη και διάθεση λογισμικού στον πελάτη.

   ΝΑΙ ΟΧΙ

8. Διάλεξε την κυριότερη καλή πρακτική στην ανάπτυξη λογισμικού:.

   Α. Συχνή παράδοση Β. Τεκμηρίωση ή Γ. Διεξοδικές δοκιμές

9. Η καλή διαχείριση έκδοσης αφορά:.

   Α. στην παράδοση του λογισμικού Β. στον έλεγχο των αλλαγών στον κώδικα

10. Επέλεξε ποιες γλώσσες είναι κατάλληλες για το υπόδειγμα δομημένου προγραμματισμού:.

    Α. C Β. SQL Γ. HTML/CSS Δ.Haskell Ε. Python

1.5 Ερωτήσεις Ανάπτυξης

1. Ποια είναι τα 6 στάδια επίλυσης προβλήματος με Η/Υ;
2. Ποιες είναι οι 5 φάσεις ανάπτυξης λογισμικού με το μοντέλο του καταρράκτη;
3. Ποιες είναι οι αρχές της ευέλικτης ανάπτυξης;
4. Τι προσπαθούμε να διασφαλίσουμε όταν εφαρμόζουμε καλές πρακτικές στην ανάπτυξη λογισμικού;
5. Με ποιο τρόπο εισάγουμε σχόλια μέσα στην εφαρμογή όταν προγραμματίζουμε σε γλώσσα Python;
6. Είναι κάθε γλώσσα κατάλληλη για ένα μόνο πτρογραμματιστικό υπόδειγμα; Δώσε ένα παράδειγμα.
7. Τι εξασφαλίζουν τα χαρακτηριστικά της Python που είναι ιδιαίτερα επιθυμητό στο πλαίσιο της διδασκαλίας; Κατά τη γνώμη σου, είναι αυτά επιθυμητά μόνο σε διδακτικό πλαίσιο;

Κεφάλαιο 2

2. Γνωριμία με τη γλώσσα Python

Τελειώνοντας αυτό το μάθημα θα έχεις μάθει

- Στοιχεία για το παρελθόν και το παρόν της γλώσσας Python
- Να εκτελείς απλές εντολές εξόδου στο Περιβάλλον του Διερμηνέα της Python
- Να δημιουργείς απλές ομάδες εντολών (συναρτήσεις)
- Να συντάσσεις και να εκτελείς απλά προγράμματα στο Περιβάλλον IDLE
- Να αλληλοεπιδράς με το χρήστη συντάσσοντας απλές εντολές Εισόδου
- Να πραγματοποιείς απλές αριθμητικές πράξεις
- Τη λειτουργία των χαρακτήρων - ακολουθιών διαφυγής
- Να χρησιμοποιείς απλές σταθερές και μεταβλητές
- Να αναγνωρίζεις ορισμένους τύπους λαθών στον κώδικά σου

2.1 Εισαγωγή στη γλώσσα Python

2.1.1 Ιστορικά στοιχεία και χαρακτηριστικά

Η γλώσσα προγραμματισμού Python είναι μία σύγχρονη γλώσσα υψηλού επιπέδου. Παρουσιάστηκε στις αρχές του 1990, με οδηγό δημιουργό τον Guido van Rossum για χρήση ως γλώσσα γενικού σκοπού για την επίλυση κάθε είδους προβλημάτων. Χαρακτηρίζεται από την υψηλή αισθητική, την καλή αναγνωσιμότητα, την ευκολία εκμάθησης, την ταχύτητα ανάπτυξης, την πληθώρα βιβλιοθηκών.

2.1.2 Λήψη και εγκατάσταση της γλώσσας

Το πακέτο εγκατάστασης της γλώσσας και τα εργαλεία της είναι ελεύθερα ακόμα και για εμπορική χρήση σύμφωνα με την OSI (Open Source License) άδεια την οποία διαχειρίζεται το ίδρυμα Python Software Foundation.

Την τελευταία έκδοση της γλώσσας Python 3.11.2 μπορείτε να κατεβάσετε από εδώ ανάλογα με το Λειτουργικό Σύστημα (ΛΣ) που διαθέτετε ή από τον παρακάτω πίνακα.

Version	Operating System
Gzipped source tarball	Source release
XZ compressed source tarball	Source release
macOS 64-bit universal2 installer	macOS
Windows embeddable package (32-bit)	Windows
Windows embeddable package (64-bit)	Windows
Windows embeddable package (ARM64)	Windows
Windows installer (32-bit)	Windows
Windows installer (64-bit)	Windows
Windows installer (ARM64)	Windows

Ο επίσημος δικτυακός τόπος υποστήριξης της γλώσσας Python βρίσκεται στο σύνδεσμο Welcome to Python.org.

2.1.3 Εφαρμογές της γλώσσας Python

Η γλώσσα Python είναι πολύ δημοφιλής στους τομείς της ανάπτυξης εφαρμογών για το διαδίκτυο (web development), στην επιστήμη δεδομένων (Data Science, Data analytics, Data Visualisation), στην τεχνητή νοημοσύνη (Artificial Intelligence), στη μηχανική μάθηση (Machine Learning, deep learning), στους αυτοματισμούς εργασιών (task automations), στην ανάπτυξη οικονομικών εφαρμογών, αλλά και σε πολλά είδη καθημερινών εφαρμογών.

Χρησιμοποιείται από μεγάλες εταιρείες όπως: Intel, IBM, NASA, Pixar, Netflix, Facebook, JP Morgan Chase, Spotify κά.

Στην εκπαίδευση, στην επιστήμη της Πληροφορικής αλλά και σε άλλους κλάδους, χρησιμοποιείται από πολλούς εκπαιδευτικούς οργανισμούς και εκατοντάδες πανεπιστήμια στον κόσμο για την εκμάθηση της αλγοριθμικής και του προγραμματισμού. Στην Ελλάδα εντάχθηκε στο πρόγραμμα σπουδών της Β' Θμιας Επαγγελματικής Εκπαίδευσης το 2016.

2.1.4 Ενσωματωμένα περιβάλλοντα συγγραφής κώδικα

Η Python είναι διερμηνευόμενη (interpreted) γλώσσα και διαθέτει δύο περιβάλλοντα σύνταξης εντολών - κώδικα.

α) **Διαδραστικό περιβάλλον** με δυνατότητα γραφής και άμεσης εκτέλεσης μεμονωμένων εντολών που ονομάζεται κέλυφος (shell). Εδώ, κάθε εντολή αρχικά ελέγχεται από το διερμηνέα. Εάν είναι ορθή στη σύνταξή της, μεταφράζεται σε γλώσσα χαμηλού επιπέδου κατανοητή από τον υπολογιστή και στη συνέχεια εκτελείται από το διερμηνέα εμφανίζοντας άμεσα αποτελέσματα ή σε διαφορετική περίπτωση κάποιο μήνυμα λάθους. Το κέλυφος είναι περιβάλλον κατάλληλο για δοκιμές ορθότητας των εντολών του προγραμματιστή και χρήσιμο για την τμηματική συγγραφή του πηγαίου κώδικα (source code).

Το περιβάλλον του κελύφους (shell)

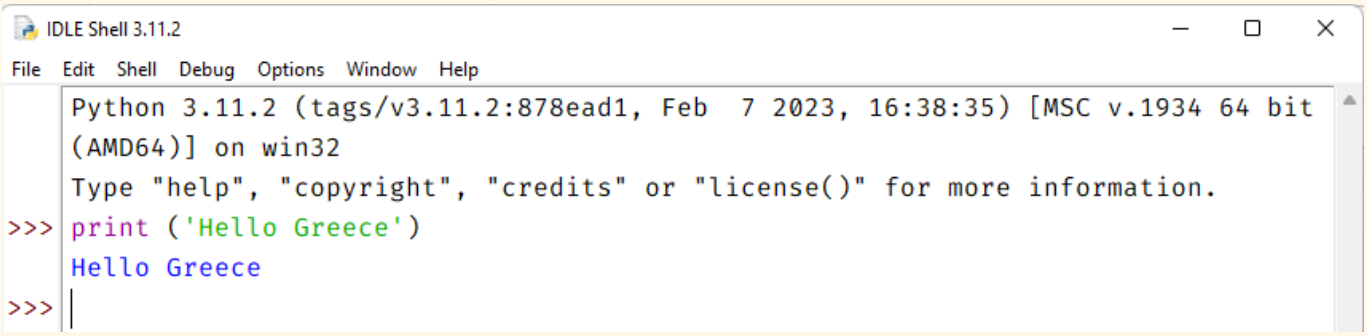
Στην εικόνα που ακολουθεί βλέπουμε το κέλυφος της Python το οποίο χρησιμοποιείται για άμεση εκτέλεση εντολών. Χαρακτηριστικό γνώρισμα του διερμηνέα αποτελεί το σύμβολο προτροπής (command prompt) `>>>`

Στο κέλυφος κάθε εντολή συντάσσεται σε μία ή σε περισσότερες γραμμές και εκτελείται πατώντας το πλήκτρο <enter>.

Η προτεραιότητα των πράξεων, η αποτίμηση/υπολογισμός (evaluation) των αριθμητικών και λοιπών εκφράσεων και η χρήση των διαφόρων τελεστών ακολουθεί στις περισσότερες περιπτώσεις τους βασικούς κανόνες των μαθηματικών. Αυτά είναι θέματα στα οποία θα γίνει πιο αναλυτική αναφορά σε επόμενα κεφάλαια.

Παράδειγμα 2.1

Ανοίξτε το περιβάλλον του κελύφους συντάξτε και εκτελέστε τις πρώτες σας εντολές για να εμφανίσετε κάποια μηνύματα και να κάνετε βασικές πράξεις:




```
>>> print('Καλημέρα' + 'Ελλάδα')
ΚαλημέραΕλλάδα
>>> print(7 + 3 * 10)
37
>>> print(7 * 8 == 56)
True
>>> print(2 ** 3 + 2)
10
>>> print(10 * '$')
$$$$$$$$$$
>>>
```

β) **Περιβάλλον με δυνατότητα ανάπτυξης ολόκληρων προγραμμάτων εντολών** (scripts = σεναρίων). Το ενσωματωμένο αυτό περιβάλλον ανάπτυξης ονομάζεται IDLE. Αποτελείται από έναν απλό συντάκτη προγραμμάτων (editor) και κάποια απλά, βασικά βοηθητικά εργαλεία (programming tools) για τον προγραμματιστή. Τα προγράμματα (σενάρια: scripts) περιέχουν τον πηγαίο κώδικα και αποθηκεύονται με τη μορφή αρχείων με επέκταση (.py). Το IDLE αποτελεί ένα πολύ βασικό περιβάλλον ανάπτυξης εφαρμογών που ανήκει στην κατηγορία των ονομαζόμενων IDE (Integrated Development Environment). Για πιο σύνθετα έργα απαιτούνται πιο ολοκληρωμένα τέτοια περιβάλλοντα ανάπτυξης με περισσότερες δυνατότητες και βοηθητικά εργαλεία για τον προγραμματιστή. Στο Διαδίκτυο υπάρχουν διαθέσιμα δεκάδες τέτοια περιβάλλοντα καλύπτοντας τις απαιτήσεις των χρηστών τους. Το «καλύτερο» από αυτά δε, είναι ένας χαρακτηρισμός μάλλον υποκειμενικός. Με την πάροδο του χρόνου και πολλές δοκιμές ο καθένας θα βρει αυτό που ταιριάζει καλύτερα στις ανάγκες του.

Το περιβάλλον του συντάκτη (Python IDLE)

Η Python όπως αναφέρθηκε είναι γλώσσα υψηλής αισθητικής και καλής αναγνωσιμότητας. Στο παράθυρο που ακολουθεί περιλαμβάνεται προς επίδειξη ο ολοκληρωμένος κώδικας ενός προγράμματος/σεναρίου που θα μελετηθεί ως άσκηση σε επόμενο κεφάλαιο στο IDLE.

Παράδειγμα 2.2

*Μπορείτε να εκτελέσετε το πρόγραμμα **από το σύνδεσμο στο τέλος του** και να εξηγήσετε τη λειτουργία του από τα αποτελέσματα που εμφανίζονται στην οθόνη;*

```
from datetime import datetime
from time import sleep
```

```
def speak(num):
    # Ορισμός ενός λεξικού για τους αριθμούς 1..19
    ones = {
        0: "",
        1: "Ένα",
        2: "Δύο",
        3: "Τρία",
        4: "Τέσσερα",
        5: "Πέντε",
        6: "Έξι",
        7: "Επτά",
        8: "Οκτώ",
        9: "Εννιά",
        10: "Δέκα",
        11: "Έντεκα",
        12: "Δώδεκα",
        13: "Δεκατρία",
        14: "Δεκατέσσερα",
        15: "Δεκαπέντε",
        16: "Δεκαέξι",
        17: "Δεκαεπτά",
        18: "Δεκαοκτώ",
        19: "Δεκαεννιά",
    }

    # Ορισμός ενός λεξικού για τα πολ/σια του 10 από 20..90
    tens = {
        20: "Είκοσι",
        30: "Τριάντα",
        40: "Σαράντα",
        50: "Πενήντα",
        60: "Εξήντα",
        70: "Εβδομήντα",
        80: "Ογδόντα",
        90: "Ενενήντα",
    }
    if num < 20:
        return ones[num]
    elif num < 100:
        return tens[num - (num % 10)] + ones[num % 10]


while True:
```

```
    now = datetime.now()
    now = now.strftime("%d/%m/%Y %H:%M:%S")
    h = int(now[11:13])
    m = int(now[14:16])
    s = int(now[17:19])
    print(h, m, s, speak(h), speak(m), speak(s))
    sleep(1)
```

Σύνδεσμος κώδικα στο replit

2.2 Εντολή εκτύπωσης της γλώσσας Python

Τα περισσότερα προγράμματα εμφανίζουν τα αποτελέσματά τους στην οθόνη του χρήστη**. Η εμφάνιση αποτελεσμάτων υπολογισμών, μηνυμάτων κ.ά. είτε στο κέλυφος** (ή και στα ολοκληρωμένα σενάρια/προγράμματα) γίνεται χρησιμοποιώντας την εντολή εξόδου της γλώσσας `print`.

Τα μέρη προς εκτύπωση τοποθετούνται στο ζεύγος των παρενθέσεων που ακολουθούν την εντολή. Εάν αυτά αφορούν σε κάποιο μήνυμα που περιέχει κείμενο περικλείεται σε τόνους (μονούς ή διπλούς), εάν αφορά σε κάποιο υπολογισμό τοποθετείται χωρίς τόνους και διαχωρίζεται με κόμμα από τα υπόλοιπα μέρη.

Παράδειγμα 2.3

Συντάξτε και εκτελέστε στο κέλυφος τις εντολές:

```
>>> print('Hello World', 2000 + 23)
Hello World 2023

>>> print('Hello' + 'World', 10 * 200 + 23)
HelloWorld 2023

>>> print('Hello' * 2 + 'World', 10 ** 4 / 5 + 23)
HelloHelloWorld 2023.0

>>> print('Hello\nWorld', 20, '\n', 23)
Hello
World 20

23
```

Παρατηρήστε ότι στις παραπάνω εντολές τα μηνύματα τα οποία ήταν ανάμεσα σε τόνους εμφανίστηκαν ακριβώς όπως συντάχθηκαν, ενώ οι πράξεις υπολογισμού εκτός τόνων αποτιμήθηκαν και εμφανίστηκε το αποτέλεσμά τους.

2.2.1 Δραστηριότητα

α. Απαντήστε τις παρακάτω ερωτήσεις οι οποίες αφορούν στο παράδειγμα 2.3
- Ποια είναι τα σύμβολα της πρόσθεσης και του πολλαπλασιασμού
- Είναι εφικτή η πράξη της πρόσθεσης στο κείμενο
- Είναι εφικτή η πράξη του πολλαπλασιασμού στο κείμενο
- Ποιο σύμβολο χρησιμοποιείται για την πράξη της διαίρεσης
- Ποια η λειτουργία του συμβόλου `**`
- Για ποιο λόγο πιστεύετε ότι σε όλες τις προηγούμενες εντολές εμφανίζεται 2023 ενώ στην προτελευταία 2023.0
- Ποιος είναι ο ρόλος του συμβόλου `'\n'` στην τελευταία εντολή `print`

β. Συντάξτε και εκτελέστε στο κέλυφος την εντολή με την οποία θα εμφανίσετε:

- 5 φορές τη λέξη SOS διαχωρισμένες με ένα κενό
- 5 γραμμές με τη λέξη SOS και μία κενή γραμμή στο τέλος
- Από τη φράση `Beautiful is better than ugly` η οποία προέρχεται από το zen της Python εμφανίστε μία λέξη σε κάθε γραμμή
- Το άθροισμα του αθροίσματος των ηλικιών σε έτη όλων των μελών της οικογενείας σας
- Το μέσο όρο της ηλικίας των μελών της οικογενείας σας
- Την τελική τιμή ενός προϊόντος με αρχική τιμή 80 € και ΦΠΑ 23%
- Το συνολικό κόστος μίας παράστασης για μία 5μελή οικογένεια (2 γονείς + 3 παιδιά) με τιμή εισιτηρίου 10 € για ενήλικες και 6 € για παιδιά

Παράδειγμα 2.4

Το zen της Python μπορείτε να το δείτε και με την παρακάτω εντολή

```
>>> import this
The Zen of Python, by Tim Peters

   1. Beautiful is better than ugly.
   2. Explicit is better than implicit.
   3. Simple is better than complex.
   4. Complex is better than complicated.
   5. Flat is better than nested.
   6. Sparse is better than dense.
   7. Readability counts.
   8. Special cases aren't special enough to break the rules.
   9. Although practicality beats purity.
   10.      Errors should never pass silently.
   11.      Unless explicitly silenced.
   12.      In the face of ambiguity, refuse the temptation to guess.
   13.      There should be one-- and preferably only one --obvious way to do it.
   14.      Although that way may not be obvious at first unless you're Dutch.
   15.      Now is better than never.
   16.      Although never is often better than *right* now.
   17.      If the implementation is hard to explain, it's a bad idea.
   18.      If the implementation is easy to explain, it may be a good idea.
   19.      Namespaces are one honking great idea -- let's do more of those!
```

Η μετάφρασή του στα ελληνικά από την ομάδα TasPython της Ελληνικής Κοινότητας Προγραμματιστών Python και τον οδηγό εκμάθησης Βήμα Προς Βήμα του Δ. Λεβεντέα.

```
1. Όμορφο είναι καλύτερο από άσχημο.
2. Άμεσο είναι καλύτερο από έμμεσο.
3. Απλό είναι καλύτερο από σύνθετο.
4. Σύνθετο είναι καλύτερο από περίπλοκο.
5. Επίπεδο είναι καλύτερο από εμφωλευμένο.
6. Αραιό είναι καλύτερο από πυκνό.
7. Η αναγνωσιμότητα μετράει.
8. Οι ειδικές περιπτώσεις δεν είναι αρκετά ειδικές ώστε να σπάνε τους κανόνες.
9. Ωστόσο η πρακτικότητα υπερτερεί της αγνότητας.
10.Τα λάθη δεν θα πρέπει ποτέ να αποσιωπώνται.
11.Εκτός αν αποσιωπώνται ζητά.
12.Όταν αντιμετωπίζεις την αμφιβολία, αρνήσου τον πειρασμό να μμαντέψεις.
13.Θα πρέπει να υπάρχει ένας— και προτιμητέα μόνο ένας —προφανής τρόπος να το
   κάνεις.
```

```
14.Αν και αυτός ο τρόπος μπορείς να μην είναι προφανής εκτός αν είσαι Ολλανδός.
15.Τώρα είναι καλύτερα από ποτέ.
16.Αν και ποτέ είναι συχνά καλύτερα από ακριβώς τώρα.
17.Αν η υλοποίηση είναι δύσκολο να εξηγηθεί, τότε είναι κακή ιδέα.
18.Αν η υλοποίηση είναι εύκολο να εξηγηθεί, τότε ίσως είναι καλή ιδέα.
19. Τα ονόματα χώρου  είναι μία φοβερά καλή ιδέα – ας κάνουμε περισσότερα από αυτά
    !
```

2.3 Ομάδες εντολών με την εντολή def της Python

Στο κέλυφος τη γλώσσας κάθε εντολή πατώντας <enter> εκτελείται. Έτσι εάν ο χρήστης επιθυμεί να εμφανιστούν περισσότερα μηνύματα συνεχόμενα το ένα μετά από το άλλο δίνοντας τις αντίστοιχες εντολές print, κάθε φορά θα παρεμβάλλεται ανάμεσα σε κάθε εντολή και το αποτέλεσμά της προηγούμενης προκαλώντας μη επιθυμητό αποτέλεσμα.

Παράδειγμα 2.5

Συντάξτε και εκτελέστε στο κέλυφος τις εντολές:

```
>>> print('Αντώνης Παπαδόπουλος')
Αντώνης Παπαδόπουλος
>>> print('antonis.papadop@ellak.gr')
antonis.papadop@ellak.gr
>>> print('2134545455')
2134545455
```

Υπάρχει όμως η δυνατότητα ο χρήστης να δημιουργήσει στο κέλυφος ή στα ολοκληρωμένα προγράμματά του τις δικές του «ομάδες» εντολών δίνοντάς τους κάποιο όνομα και στη συνέχεια να τις καλέσει ώστε να εκτελεστούν όλες μαζί εμφανίζοντας τα αποτελέσματα. Οι ομάδες εντολών δημιουργούνται είτε στο κέλυφος είτε στα προγράμματα (πιο σύνηθες) χρησιμοποιώντας την εντολή def (define: ορίζω) της Python.

Παράδειγμα 2.6

Συντάξτε την ακόλουθη ομάδα εντολών στο κέλυφος:

```
>>> def card():
        print('Αντώνης Παπαδόπουλος')
        print('antonis.papadop@ellak.gr')
        print('2134545455')
```

Στη συνέχεια εκτελέστε την ομάδα εντολών που ορίστηκε προηγουμένως με το όνομά της card()

```
>>> card()
Αντώνης Παπαδόπουλος
antonis.papadop@ellak.gr
2134545455
```

Παρατηρήσεις

- *Παρατηρείστε ότι οι εντολές μετά το σύμβολο (:) τοποθετήθηκαν με αυτόματη εσοχή από τη γλώσσα υπονοώντας ότι περιέχονται εντός της ομάδας εντολών με όνομα* card().
- *Έτσι στο προηγούμενο παράδειγμα δημιουργήθηκε η ομάδα εντολών με το όνομα card(). Δίνοντας το όνομά της στο κέλυφος καλείται η λειτουργία της ομάδας και εκτελούνται όλες οι εντολές που περιέχονται σε αυτή.*
- *Δοκιμάστε να επιλέξετε από το Μενού Επιλογών Shell/Restart Shell και να εκτελέσετε πάλι την ομάδα εντολών που δημιουργήθηκε προηγουμένως. Τι παρατηρείτε;*

Παράδειγμα 2.7

Στο παράδειγμα που ακολουθεί θα δοθεί στα παραπάνω μηνύματα εμφάνιση κάρτας εργασίας (business card), τα μηνύματα μετά το σύμβολο `# είναι επεξηγηματικά σχόλια` *και δεν εκτελούνται από τη γλώσσα (στο παράδειγμα μπορείτε να μην τα γράψετε).*

Συντάξτε στο κέλυφος (προσέχοντας να μην αφήσετε επιπλέον κενά ειδικά αριστερά των εντολών) και εκτελέστε την ομάδα εντολών card() που ακολουθεί:

```
>>> def card():
        print('----------------------------------------')
        print()           # Τύπωσε 1 κενή γραμμή
        print()           # Τύπωσε 1 κενή γραμμή
        print('           ' , 'Αντώνης Παπαδόπουλος') #10 κενά και το όνομα
        print(10 * ' ' , 'antonis.papadop@ellak.gr') #10 κενά και το email
        print(10 * ' ' , '2134545455') # 10 κενά και το τηλέφωνο
        print(2 * '\n')   # Τύπωσε 2 κενές γραμμές
        print(40 * '-')   # Τύπωσε 40 φορές το χαρακτήρα -
```

Καλέστε την ομάδα εντολών `card()`

```
>>> card()
```

Το αποτέλεσμα στην οθόνη φαίνεται παρακάτω:

```
----------------------------------------



          Αντώνης Παπαδόπουλος
          antonis.papadop@ellak.gr
          2134545455



----------------------------------------
```

Παράδειγμα 2.8

Τροποποιήσετε στη συνέχεια την ομάδα εντολών `card()` *χρησιμοποιώντας στην παρένθεση και στον κώδικα τα ονόματα* `name, email` *και εκτελέστε όπως φαίνεται παρακάτω:*

Σημείωση

Για να εμφανιστούν στο κέλυφος οι προηγούμενες ή οι επόμενες εντολές ώστε να μην πληκτρολογηθούν ξανά από το χρήστη χρησιμοποιείται ο συνδυασμός πλήκτρων (Alt + p) = (previous) και (Alt + n) = (next)

```
>>> def card(name, email):
```

```
        print(40 * '-')
        print(2 * '\n')
        print(10 * ' ' , name)
        print(10 * ' ' , email)
        print(2 * '\n')
        print(40 * '-')
```

Έπειτα εκτελέστε στο κέλυφος την εντολή και δείτε το αποτέλεσμα:

```
>>> card('Χρήστος Βασιλόπουλος', 'chris.vasilop@ellak.gr')
```

Σημείωση

Οι «ομάδες» εντολών που ορίζονται με την εντολή `def` όπως θα αναφερθεί σε επόμενο κεφάλαιο που θα μελετηθούν αναλυτικότερα ονομάζονται **συναρτήσεις** (functions) και χρησιμοποιούνται όχι μόνο για την ομαδοποίηση ομάδων μηνυμάτων αλλά κυρίως (μεταξύ άλλων) ως εργαλείο του τμηματικού προγραμματισμού όπου διαχωρίζεται ένα σύνθετο πρόβλημα σε μικρότερα μέρη ώστε να διευκολύνεται η επίλυσή του. Τα ονόματα δε εντός της παρένθεσης ονομάζονται παράμετροι ή ορίσματα της συνάρτησης.

2.3.1 Δραστηριότητα

Μπορείτε να καλέσετε κατάλληλα την ομάδα εντολών (συνάρτηση) `card()` ώστε να εμφανίσετε τις κάρτες τριών ή τεσσάρων συσπουδαστών σας όπως και τη δική σας στο περιβάλλον του κελύφους;

Διαλέξτε το σπουδαστή/στρια:

α. με το μικρότερο και
β. με το μεγαλύτερο ονοματεπώνυμο

Τι παρατηρείτε;

Συζητήστε με ποιους τρόπους θα μπορούσατε να αντιμετωπίσετε το αισθητικό αποτέλεσμα;

2.3.2 Δραστηριότητα

α. Δημιουργήστε μία συνάρτηση με όνομα `stair_down()` η οποία θα δημιουργεί σκαλοπάτια 5 γραμμών που κατεβαίνουν χρησιμοποιώντας το σύμβολο *. Δηλαδή:

```
>>> stairs_down()
*
**
***
****
*****
```

β. Τροποποιήστε τη συνάρτηση ώστε να δημιουργεί τα σκαλοπάτια με όποιο σύμβολο επιθυμεί ο χρήστης. Δηλαδή:

```
>>> stairs_down('$')
```

```
$
$$
$$$
$$$$
$$$$$
```

γ. Δημιουργήστε μία συνάρτηση με όνομα `stairs_up(symbol)` η οποία θα δημιουργεί σκαλοπάτια 5 γραμμών που θα ανεβαίνουν χρησιμοποιώντας όποιο σύμβολο επιλέξει ο χρήστης. Δηλαδή:

```
>>> stairs_up('#')
    #
   ##
  ###
 ####
#####
```

2.4 Επίλυση ενός συνηθισμένου προβλήματος

Στη συνέχεια θα παρουσιαστεί σταδιακά πως μπορεί να χρησιμοποιηθεί η γλώσσα για να επιλυθεί ένα συνηθισμένο πρόβλημα μισθοδοσίας:

Παράδειγμα 2.9

Ο υπάλληλος Αντώνης Παπαδόπουλος λαμβάνει βασικό μηνιαίο μισθό 714 ευρώ, έχει 2 παιδιά και παίρνει μηνιαίο επίδομα για το κάθε ένα παιδί 27 ευρώ. Διαθέτει πανεπιστημιακό πτυχίο και λαμβάνει γι' αυτό, επίδομα πτυχίου σπουδών το οποίο προσαυξάνει κατά 8.5 % τον βασικό μισθό του. Οι συνολικές αποδοχές του μειώνονται κατά 14% λόγω κρατήσεων για ασφάλεια, και περίθαλψη. Ζητείται να υπολογίσετε και να εμφανίσετε τις καθαρές αποδοχές του.

Ανάλυση του προβλήματος

Οι καθαρές αποδοχές ενός υπαλλήλου για να υπολογισθούν πρέπει να βρεθούν:

1. οι μεικτές αποδοχές του οι οποίες υπολογίζονται ως: **βασικός μισθός + επιδόματα**
2. οι κρατήσεις του οι οποίες υπολογίζονται: **14% επί των μεικτών αποδοχών**
3. τέλος οι καθαρές αποδοχές του οι οποίες υπολογίζονται: **μεικτές αποδοχές - κρατήσεις**

Λύση στο κέλυφος

Συντάξτε στο κέλυφος τις εντολές εκτελέστε:

```
>>> print(714 + 2 * 27 + 8.5 / 100 * 714)
828.69

>>> print(14 / 100 * 828.69)
116.01660000000003

>>> print(828.69 - 116.01660000000003)
712.6734
```

2.4.1 Δραστηριότητα

Μπορείτε να συμπληρώσετε τα κενά ώστε να υπολογισθεί και να εμφανιστεί το παρακάτω αποτέλεσμα σε μία γραμμή;

```
>>> print ((714 + 2 * 27 + 8.5 / 100 * 714) * ______, ______)

712.6734 Ευρώ
```

Παρατηρήσεις

- Στην έκφραση υπολογισμού της εντολής `print` τα γνωστά σύμβολα των πράξεων `+, *, /` ακολουθούν την προτεραιότητα των πράξεων όπως αυτή ορίζεται στα μαθηματικά, όπως και 2 ζεύγη παρενθέσεων για διαφορετικό σκοπό το καθένα:
- Το εσωτερικό ζεύγος `(714 + 2 * 27 + 0.085 * 714) * ____` για να αλλάξει η προτεραιότητα των μαθηματικών πράξεων ομαδοποιώντας με το ζεύγος των παρενθέσεων το μισθό χωρίς τις κρατήσεις και:

- Το εξωτερικό ζεύγος των παρενθέσεων ώστε η εντολή `print(.....)` να εμφανίσει όλα αυτά που περικλείει στην οθόνη.

2.4.2 Βελτιώσεις στη λύση του προβλήματος μισθοδοσίας

Οι παραπάνω λύσεις υπολογίζουν μεν τις καθαρές αποδοχές του υπαλλήλου αλλά:

- δεν έχουν καμία φιλικότητα προς το χρήστη και
- δεν μπορούν εύκολα να χρησιμοποιηθούν για κάποιον άλλο υπάλληλο εκτός αυτού του παραδείγματος.

Οι βελτιώσεις που θα ακολουθήσουν θα αφορούν:

α. στη βελτίωση της διεπαφής (UI - User Interface) της λύσης και
β. στην παραμετροποιησή της ώστε να γίνει πιο γενική για να μπορεί να χρησιμοποιηθεί πιο εύκολα και για άλλους υπαλλήλους.

Η επίλυση του ίδιου προβλήματος θα γίνει ως ολοκληρωμένο πρόγραμμα στη συνέχεια στο περιβάλλον IDLE.

Παράδειγμα 2.10

Δημιουργείστε στο συντάκτη (editor IDLE) ένα νέο αρχείο (File / New) και συντάξτε τον παρακάτω κώδικα.

```
name = 'Παπαβασιλείου Γεώργιος'
basic = 714
kids = 2
gross = basic + kids * 27 + 8.5 / 100 * basic
net = gross – 14 / 100 * gross
print('Ο υπάλληλος',name, 'λαμβάνει καθαρά',net , 'Ευρώ')
```

Αποθηκεύστε το σενάριο δίνοντας όνομα με λατινικούς χαρακτήρες που αρχίζει από γράμμα εκτελέστε από το μενού Run ή πατώντας το πλήκτρο F5.

Το αποτέλεσμα είναι σίγουρα βελτιωμένο αφού η κατάλληλη προσαρμογή της εντολής `print` δίνει στο χρήστη ένα πιο σαφές μήνυμα και η χρήση των ποσοτήτων `name, basic, kids, gross, net` φαίνεται να απλοποιεί την παράσταση που περικλείει η εντολή `print`.

```
Ο υπάλληλος Παπαβασιλείου Γεώργιος λαμβάνει 712.6734 Ευρώ
>>>
```

Παρατηρήσεις

- Στη σύνταξη του προγράμματος στο IDLE, το όνομα, ο βασικός μισθός, ο αριθμός των παιδιών, οι μεικτές αποδοχές, οι καθαρές αποδοχές τοποθετήθηκαν στις λέξεις `name, basic, kids, gross, net` αντίστοιχα, δημιουργώντας τρεις σχέσεις ονομάτων - τιμών. Μία (1) που αντιστοιχεί σε περιεχόμενο χαρακτήρων (με το περιεχόμενο σε τόνους) και τέσσερις (4) που αντιστοιχούν σε αριθμούς.

- Για τα θέματα των σχέσεων ονομάτων-τιμών, των πράξεων, των προτεραιοτήτων και του τύπου των περιεχομένων των μεταβλητών θα γίνει πιο εκτενής αναφορά στη συνέχεια.
- Οι λέξεις `name, basic, kids` στον προγραμματισμό δημιουργούν μία συσχέτιση μεταξύ του ονόματος και του περιεχομένου τους (το οποίο παραμένει σταθερό σε όλη τη διάρκεια εκτέλεσης του προγράμματος) και ονομάζονται συμβολικές σταθερές (symbolic constants). Όπως και οι βοηθητικές ποσότητες υπολογισμού `gross, net`. Με τη χρήση τους είναι εφικτό για τον επόμενο υπάλληλο αλλάζοντας τις τιμές τους και εκτελώντας εκ νέου το πρόγραμμα να υπολογισθεί ο νέος μισθός χωρίς τροποποίηση της έκφρασης υπολογισμού του.

2.4.3 Δραστηριότητα

Τροποποιήστε το παραπάνω πρόγραμμα που συντάχθηκε όπως φαίνεται παρακάτω:

```
name = 'Παπαβασιλείου Γεώργιος'
basic = 714
kids = 2
gross = basic + kids * 27 + 8.5 / 100 * basic
net = gross – 14 / 100 * gross
print('Ο υπάλληλος',name, 'λαμβάνει καθαρά',net , 'Ευρώ')
```

Μπορείτε να απαντήσετε στις παρακάτω ερωτήσεις;

1. Η εντολή `print` και τα μηνύματα έχουν άλλο χρώμα μπορείτε να εξηγήσετε το λόγο ;
2. Δοκιμάστε να αλλάξετε την `print` γράφοντας με κεφαλαίο το 1ο γράμμα δηλαδή `Print(...)`. Τι παρατηρείτε ;
3. Προσθέστε ένα κενό αριστερά από την εντολή `print` και εκτελέστε τον κώδικα, τι παρατηρείτε ;
4. Αλλάξτε την εντολή 5 σε

   ```
   net = gross – 14 % gross
   ```

   τι παρατηρείτε;

Απαντήσεις και σχολιασμός των αποτελεσμάτων της Δραστηριότητας

1. *Παρατηρείται ότι η αλλαγή του ονόματος της εντολής `print` σε `Print` είχε σαν αποτέλεσμα το παρακάτω μήνυμα όπου ο διερμηνέας της Python επισημαίνει στο σχετικό μήνυμα λάθους ότι η εντολή print είναι διαφορετική από την εντολή Print. Αυτό συμβαίνει διότι η Python είναι case sensitive γλώσσα (δηλαδή στις εντολές της οι πεζοί χαρακτήρες είναι διαφορετικοί από τους αντίστοιχους σε κεφαλαία).*

```
Traceback (most recent call last):
  File "C:/Users/Administrator/AppData/Local/Programs/Python/Python311/my_script
s/DIEK/kef_01/ask1_test.py", line 5, in <module>
    Print ('Ο υπάλληλος',name, 'λαμβάνει',\
NameError: name 'Print' is not defined. Did you mean: 'print'?
>>>
```

2. *Ο διερμηνέας αρνείται να εκτελέσει το πρόγραμμα εμφανίζοντας ένα αναδυόμενο παράθυρο μηνύματος (pop up message) προειδοποιώντας για συντακτικό λάθος στη αριστερή στοίχιση του κώδικα η οποία είναι ιδιαίτερα σημαντική στην Python.*
3. *Το σύμβολο τελεστής % έχει διαφορετική χρήση και λειτουργία από αυτήν του επί τις 100 που χρησιμοποιείται λεκτικά στα ποσοστά, έτσι εμφανίζεται λάθος στο αριθμητικό αποτέλεσμα.*

Παρατηρήσεις

- Τα λάθη στις ερωτήσεις 1 και 2 της προηγούμενης δραστηριότητας ονομάζονται συντακτικά λάθη (Syntax errors) διότι παραβιάζουν τους συντακτικούς κανόνες γραφής (σύνταξης) της γλώσσας.
- Το λάθος στην ερώτηση 3 ονομάζεται λογικό λάθος (logical error) διότι ενώ ο κώδικας εκτελείται κανονικά η λανθασμένη χρήση των συμβόλων των πράξεων εμφανίζει λάθος αποτελέσματα.

2.5 Χρήση εντολής εισόδου δεδομένων input της Python

Τα προγράμματα συνήθως απαιτούν αλληλεπίδραση με το χρήστη, δηλαδή την ώρα που εκτελείται το πρόγραμμα να μπορεί αυτός να το τροφοδοτεί με τα δεδομένα που επιθυμεί.

Για παράδειγμα ο χρήστης του προγράμματος μισθοδοσίας που υλοποιήθηκε στην προηγούμενη παράγραφο, κάθε φορά που επιθυμεί να υπολογίσει το μισθό ενός νέου υπαλλήλου θα πρέπει **να τροποποιεί τον κώδικα** του σεναρίου αλλάζοντας τα δεδομένα των σταθερών: `name`, `basic`, `kids`. Αυτό σημαίνει ότι ο χρήστης κάθε φορά που συμβαίνει αυτό θα χρειάζεται:

- να ανοίξει το συντάκτη (IDLE)
- να φορτώσει τον πηγαίο κώδικα του προγράμματος
- να κάνει τις κατάλληλες διορθώσεις
- και έπειτα να τον εκτελέσει

Παράδειγμα 2.11

Για να αποφευχθεί και να απλοποιηθεί αυτή η κατάσταση τροποποιείστε το προηγούμενο πρόγραμμα χρησιμοποιώντας την εντολή εισόδου `input` όπως φαίνεται παρακάτω, ώστε ο χρήστης να μπορεί να εισάγει τα νέα δεδομένα όχι διορθώνοντας τον πηγαίο κώδικα με τον συντάκτη (edit time) αλλά την ώρα που εκτελείται το πρόγραμμα (run time).

2.5.1 Βελτίωση της λύσης του προβλήματος μισθοδοσίας με τη χρήση εντολών εισόδου

Παράδειγμα 2.12

Συντάξτε σε νέο (File / New) τον παρακάτω κώδικα.

```
name = input('Δώστε το όνομα του υπαλλήλου :')
basic = int(input('Δώστε το βασικό μισθό :'))
kids = int(input('Δώστε τον αριθμό παιδιών :'))

gross = basic + kids * 27 + 8.5 / 100 * basic
net = gross – 14 / 100 * gross
print('Ο υπάλληλος',name, 'λαμβάνει καθαρά',\
net, 'Ευρώ')
```

Παρατηρήσεις

Οι τιμές των `name, basic, kids` οι οποίες αλλάζουν κατά τη διάρκεια εκτέλεσης του σεναρίου δέχονται πλέον τιμές από το χρήστη σε run-time χρόνο, δηλαδή την ώρα που εκτελείται το πρόγραμμα. Αυτό επιτυγχάνεται με την εντολή εισόδου input. Οι ποσότητες `name, basic, kids` όπως και τις `gross, net`, αντί για σταθερές είναι πλέον μεταβλητές (variables, το περιεχόμενο τους μεταβάλλεται κάθε φορά που θα εκτελείται το πρόγραμμα). Το αποτέλεσμα στην οθόνη θυμίζει τώρα ένα κανονικό πρόγραμμα.

```
Δώστε το όνομα του υπαλλήλου :Γεωργίου
Δώστε το βασικό μισθό :967
Δώστε τον αριθμό παιδιών :3
Ο υπάλληλος Γεωργίου λαμβάνει 953.4734000000001 Ευρώ
>>>
```

Παράδειγμα 2.13

1. *Αφού εκτελέσετε το παραπάνω σενάριο στη συνέχεια προσπαθήστε να απαντήσετε:*
   - *α. Τι κάνουν οι εντολές `int` ? γιατί δεν έχει `int` το `input` της μεταβλητής `name` ?*
   - *β. Ποιος ο ρόλος του χαρακτήρα \ στο τέλος της γραμμής ;*
2. *Δοκιμάστε να τροποποιήσετε και να εκτελέσετε το πρόγραμμα ως εξής:*

```
print('Δώστε το όνομα του υπαλλήλου :', end='')
name = input()
print('Δώστε το βασικό μισθό :')
basic = int(input())
kids = int(input('Δώστε τον αριθμό παιδιών :'))

gross = basic + kids * 27 + 8.5 / 100 * basic
net = gross – 14 / 100 * gross
print('Ο υπάλληλος',name, 'λαμβάνει καθαρά',\
net, 'Ευρώ')
```

Τι παρατηρείτε ?

- α. Ποιο το αποτέλεσμα της παραμέτρου `end=''` στο τέλος της εντολής `print`
- β. Αλλάξτε την `end=''` με την `end='\n'`
- γ. Στη συνέχεια αλλάξτε με την `end = '\t'` και μετά με την `end = '\t\t'`
- δ. Τι υπολογίζουν οι νέες μεταβλητές `gross, net` ?

3. Εκτελέστε το παραπάνω πρόγραμμα δίνοντας τα παρακάτω δεδομένα:

```
Δώστε το όνομα του υπαλλήλου :Γεωργίου
Δώστε το βασικό μισθό :940.45
```

Μπορείτε να συνεχίσετε την εκτέλεση του προγράμματος ;

Απαντήσεις και σχολιασμός των αποτελεσμάτων του παραδείγματος

1.
   - *α. Η εντολή `int` μετατρέπει τα δεδομένα τύπου χαρακτήρα σε ακεραίους*
   - *β. Συνεχίζεται η σύνταξη της μακροσκελούς εντολής στην επόμενη γραμμή*

2. *Η παράμετρος `end=''` χρησιμοποιείται από την Python ώστε να ρυθμίζεται τι θα τυπωθεί στο τέλος της γραμμής*
   - *α. Εάν οι τόνοι δεν περιέχουν κάτι δε θα τυπωθεί τίποτα*

β. *Εάν οι τόνοι περιέχουν* `'\n'` *αφού τυπωθεί η γραμμή θα μετακινηθεί ο δρομέας στην αρχή της επόμενης (είναι η προεπιλεγμένη ρύθμιση: default) δηλαδή εάν δεν χρησιμοποιήσουμε την παράμετρο* `'\n'` *η λειτουργία εκτελείται έτσι κι αλλιώς), προσοχή δεν εμφανίζεται κενή γραμμή.*

γ. *Εάν οι τόνοι περιέχουν* `'\t'` *μετά το τέλος της εκτύπωσης ο δρομέας θα μετακινηθεί κατά μία εσοχή δεξιά.*

δ. *Υπολογίζουν τις μικτές και τις καθαρές αποδοχές αντίστοιχα*

3. *Όχι, διότι το αλφαριθμητικό περιεχόμενο της εντολής* `input(940.75)` *δεν αντιστοιχεί σε ακέραιο προκαλώντας λάθος χρόνου εκτέλεσης (run time error) και διακοπή του προγράμματος.*

2.6 Ενδεικτική Ολοκληρωμένη Λύση στο πρόβλημα της μισθοδοσίας

Από το αρχικό και «συγκεκριμένο πρόβλημα μισθοδοσίας» που αφορά σε ένα και μόνο υπάλληλο μίας επιχείρησης:

Ο υπάλληλος Αντώνης Παπαδόπουλος λαμβάνει βασικό μηνιαίο μισθό 714 ευρώ, έχει 2 παιδιά και παίρνει μηνιαίο επίδομα για το κάθε ένα παιδί 27 ευρώ. Διαθέτει πανεπιστημιακό πτυχίο και λαμβάνει γι' αυτό, επίδομα πτυχίου σπουδών το οποίο προσαυξάνει κατά 8.5 % τον βασικό μισθό του. Ο συνολικές αποδοχές του μειώνονται κατά 14% λόγω κρατήσεων, για ασφάλεια, περίθαλψη κλπ. Ζητείται να υπολογίσετε και να εμφανίσετε τις καθαρές αποδοχές του.

σε πραγματικές συνθήκες θα έπρεπε να αντιμετωπισθεί το πιο γενικό πρόβλημα μισθοδοσίας το οποίο θα αφορούσε σε όλους τους υπαλλήλους ΠΕ της επιχείρησης:

Για να υπολογισθούν οι αποδοχές των υπαλλήλων ΠΕ (Πανεπιστημιακής Εκπαίδευσης) της επιχείρησης και να τυπωθεί το σημείωμα μισθοδοσίας τους πρέπει να είναι γνωστά από τη νομοθεσία:

α. Το επίδομα σε ευρώ για κάθε παιδί
β. Το % επίδομα σπουδών επί του βασικού τους μισθού
γ. Το % κρατήσεων επί των συνολικών αποδοχών τους

για κάθε υπάλληλο θα δίνονται ως δεδομένα:

α. Το όνομα του υπαλλήλου
β. Ο βασικός μισθός του υπαλλήλου,
γ. Ο αριθμός των παιδιών του υπαλλήλου

Στη συνέχεια αφού γίνουν οι απαιτούμενοι υπολογισμοί να εμφανίζεται το σημείωμα μισθοδοσίας τους.

Παράδειγμα 2.14

Δημιουργείστε στο συντάκτη (editor) ένα νέο αρχείο (File / New) και συντάξτε τον παρακάτω κώδικα.

```
# Περιοχή Δηλώσεων ΣΤΑΘΕΡΩΝ
DEDUCTIONS_RATE = 14  # % Ποσοστό Κρατήσεων
KIDS_BENEFIT = 27  # Επίδομα Τέκνου
STUDIES_BENEFIT = 8.5  # % Επίδομα Σπουδών

# Είσοδος Δεδομένων Υπαλλήλου
employee_name = input("Δώστε το όνομα του υπαλλήλου : ")
minimum_wage = float(input("Δώστε το μηνιαίο βασικό μισθό του υπαλλήλου :
"))
num_of_kids = int(input("Δώστε τον αριθμό των παιδιών του υπαλλήλου : "))

# Υπολογισμός Αποτελεσμάτων
studies_benefit = minimum_wage * STUDIES_BENEFIT / 100
```

```
kids_benefit = num_of_kids * KIDS_BENEFIT
gross_wages = minimum_wage + kids_benefit + studies_benefit
deductions = gross_wages * DEDUCTIONS_RATE / 100
net_wages = gross_wages - deductions

# Εμφάνιση Αποτελεσμάτων
print("-------------------------------------------------")
print("Μισθοδοσία Υπαλλήλου \t", employee_name)
print("-------------------------------------------------")
print("Μικτές Αποδοχές \t\t", gross_wages)
print("Επίδομα Τέκνων \t\t\t", kids_benefit)
print("Επίδομα Σπουδών \t\t", studies_benefit)
print("Συνολικές Κρατήσεις \t\t", deductions)
print("-------------------------------------------------")
print("Καθαρές Αποδοχές  \t\t", net_wages)
```

Αποτελέσματα δοκιμαστικής εκτέλεσης του παραπάνω προγράμματος

```
Δώστε το όνομα του υπαλλήλου : Παπαβασιλείου Γεώργιος
Δώστε το μηνιαίο βασικό μισθό του υπαλλήλου : 714
Δώστε τον αριθμό των παιδιών του υπαλλήλου : 2
-------------------------------------------------------------------
Μισθοδοσία Υπαλλήλου     Παπαβασιλείου Γεώργιος
-------------------------------------------------------------------
Μικτές Αποδοχές          828.69
Επίδομα Τέκνων           54
Επίδομα Σπουδών          60.69
Συνολικές Κρατήσεις      116.0166
-------------------------------------------------------------------
Καθαρές Αποδοχές         712.6734
```

Σύνδεσμος κώδικα στο replit

Σχολιασμός της ενδεικτικής λύσης του προβλήματος μισθοδοσίας

Η αλήθεια είναι ότι στην αγορά εργασίας δεν καλούμαστε να επιλύσουμε ασκήσεις από ένα βιβλίο αλλά προβλήματα της εργασιακής καθημερινότητας. Πολλές φορές αυτός που θέτει τα προβλήματα προς επίλυση δεν έχει ολοκληρωμένη εικόνα όσον αφορά στη διαδικασία της κατασκευής ενός προγράμματος. Δε γνωρίζει τις διαδικασίες τροποποίησης, συντήρησης ενός λογισμικού. Όσο πιο παραμετροποιήσιμο γίνει το λογισμικό που αναπτύσσουμε απαιτεί μεν πιο πολύ χρόνο στην αρχική συγγραφή του αλλά μπορεί να καλύψει περισσότερες εκφάνσεις και διαφοροποιήσεις του αρχικού προβλήματος.

Έτσι οι λόγοι που οδήγησαν στην ενδεικτική λύση είναι:

- *Η λύση να μπορεί να χρησιμοποιηθεί για περισσότερους από έναν υπαλλήλους.*
- *Η λήψη μέριμνας ότι, στοιχεία που δεν μεταβάλλονται από υπάλληλο σε υπάλληλο αλλά αλλάζουν πιο σπάνια και αφορούν σε όλους τους υπαλλήλους, όπως το % κρατήσεων, το επίδομα*

τέκνων και το επίδομα σπουδών θα μπορούσαν να οριστούν στην αρχή του προγράμματος ως σταθερές. Έτσι κάθε φορά (αλλά όχι συχνά) που υπάρχει νέα νομοθεσία να μπορεί και ένας απλός χρήστης να αλλάξει τις αρχικές τιμές χωρίς να παρέμβει στο κύριο τμήμα του κώδικα που αφορά στους υπολογισμούς με κίνδυνο να κάνει κάποιο λάθος.

- *Η δόμηση του κώδικα σε τμήματα λειτουργιών (ΕΙΣΟΔΟΣ – ΕΠΕΞΕΡΓΑΣΙΑ – ΕΞΟΔΟΣ) ώστε να είναι σαφές τι γίνεται και που ώστε όταν απαιτούνται αλλαγές η παρέμβαση να γίνεται μόνο στο συγκεκριμένο τμήμα.*
- *Ο ορισμός ονομάτων σταθερών και μεταβλητών που να βγάζουν νόημα και να ακολουθούν τους κανόνες PEP8 βελτιώνοντας την αναγνωσιμότητα του παραγόμενου κώδικα.*
- *Η τοποθέτηση επεξηγηματικών σχολίων (comments). Αυτά θα φανούν ιδιαίτερα χρήσιμα εάν χρειαστεί να παρέμβει στον κώδικα κάποιος άλλος προγραμματιστής προκειμένου να κάνει τροποποιήσεις ή προσθήκες. Αποτελούν τμήμα της τεκμηρίωσης λογισμικού (software documentation).*
- *Η ερώτηση που αφορά στο πως θα μπορούσε να τροποποιηθεί το πρόγραμμα ώστε η λύση να καλύπτει εκτός από υπαλλήλους ΠΕ και υπαλλήλους ΔΕ (Δευτεροβάθμιας Εκπαίδευσης), ΜΤ (Μεταπτυχιακού Τίτλου Σπουδών), ΔΤ (Διδακτορικού Τίτλου Σπουδών) κλπ. ξεφεύγει από το πλαίσιο γνώσεων του τρέχοντος κεφαλαίου.*

2.7 Να θυμάμαι

Πηγαίος κώδικας (source code)

Αποτελείται από το σύνολο των οδηγιών – εντολών σε κάποια γλώσσα προγραμματισμού οι οποίες έχουν γραφεί από το δημιουργό του προγράμματος σε κάποιο περιβάλλον σύνταξης (editor). Είναι εκτελέσιμος από τον υπολογιστή MONO μετά από, έλεγχο της ορθότητάς του και στη συνέχεια της μετατροπής του σε γλώσσα μηχανής.

PEP 8

Είναι ένα έγγραφο το οποίο περιέχει οδηγίες και βέλτιστες πρακτικές για το πως πρέπει να γράφεται ο κώδικας σε γλώσσα Python. Το έγγραφο αυτό συντάχθηκε από τους δημιουργούς της γλώσσας και ο πρωταρχικός του στόχος είναι να βελτιώσει την ομοιογένεια, την αναγνωσιμότητα και τη συνέπεια του κώδικα. Περιγράφει χαρακτηριστικά που αφορούν το στυλ της γραφής του κώδικα, την ονοματολογία σταθερών, μεταβλητών, συναρτήσεων, σχολίων, κλάσεων, κενές περιοχές κ.α.

PEP 8 exists to improve the readability of Python code. But why is readability so important? Why is writing readable code one of the guiding principles of the Python language?

As Guido van Rossum said, "Code is read much more often than it is written." You may spend a few minutes, or a whole day, writing a piece of code to process user authentication. Once you've written it, you're never going to write it again. But you'll definitely have to read it again. That piece of code might remain part of a project you're working on. Every time you go back to that file, you'll have to remember what that code does and why you wrote it, so readability matters.

***Δηλαδή**:*

Όπως είπε ο Guido van Rossum, «Ο κώδικας διαβάζεται πολύ πιο συχνά από ό, τι γράφεται». Μπορείτε να αφιερώσετε λίγα λεπτά ή μια ολόκληρη μέρα γράφοντας ένα κομμάτι κώδικα για να επεξεργαστείτε τον έλεγχο ταυτότητας χρήστη. Μόλις το γράψεις, δεν πρόκειται να το ξαναγράψεις ποτέ. Αλλά σίγουρα θα πρέπει να το διαβάσετε ξανά. Αυτό το κομμάτι κώδικα μπορεί να παραμείνει μέρος ενός έργου στο οποίο εργάζεστε. Κάθε φορά που επιστρέφετε σε αυτό το αρχείο, θα πρέπει να θυμάστε τι κάνει αυτός ο κώδικας και γιατί τον γράψατε, επομένως η αναγνωσιμότητα έχει σημασία.

Πηγή

Συμβολικές Σταθερές (symbolic constants)

Οι σταθερές σε μία γλώσσα προγραμματισμού είναι ποσότητες σε ένα πρόγραμμα η αναφορά σε αυτές γίνεται με ένα συμβολικό όνομα και η τιμή τους δεν αλλάζει κατά τη διάρκεια εκτέλεσης του προγράμματος. Σύμφωνα με PEP 8 στις σταθερές δίνονται ονόματα σε ΚΕΦΑΛΑΙΑ.

Πχ `FPA = 0.23`

Μεταβλητές (variables)

Οι μεταβλητές σε μία γλώσσα προγραμματισμού είναι οντότητες σε ένα πρόγραμμα η τιμή των οποίων μπορεί να μεταβληθεί σε κάθε εκτέλεση αλλά και κατά τη διάρκεια εκτέλεσης του ιδίου προγράμματος. Οι μεταβλητές μπορούν να περιέχουν δεδομένα αριθμητικά (`int`, `float`), αλφαριθμητικά (`str`) ή λογικά (`bool` αυτός ο τύπος θα αναλυθεί σε επόμενο κεφάλαιο).

Τα ονόματά των μεταβλητών μπορεί να ξεκινούν από γράμμα ή την κάτω παύλα (_) και να περιέχουν γράμματα πεζά ή κεφαλαία, αριθμούς και την κάτω παύλα (_). Δεν μπορούν να

ξεκινούν από αριθμό. Δεν μπορούν να αποδοθούν ονόματα δεσμευμένων λέξεων πχ `print`, `input` κλπ.

Σύμφωνα με PEP 8 για τις μεταβλητές είναι σκόπιμο να χρησιμοποιούνται ονόματα σε πεζούς χαρακτήρες. Όταν πρόκειται για σύνθετες λέξεις προκειμένου να βελτιωθεί η αναγνωσιμότητα χωρίζονται τα συνθετικά τους με κάτω παύλα (_). Πχ `first_name` και όχι `FirstName`.

Όπως θα αναλυθεί και στη συνέχεια οι μεταβλητές και οι σταθερές στην Python είναι ετικέτες σε αντικείμενα διαφόρων τύπων δημιουργώντας συσχετίσεις (αναφορές) μεταξύ των ονομάτων και του περιεχομένου τους.

Σύνταξη της εντολής εξόδου print

```
print(<έκφραση>, <έκφραση>, .. , <έκφραση>)
```

Όταν η έκφραση είναι μήνυμα τοποθετείται σε τόνους, μονούς ή διπλούς. Περισσότερα για τη σύνταξη της εντολής και επιπλέον δυνατότητες της θα αναφερθούν σε επόμενο κεφάλαιο. Η εντολή print κάθε φορά που τυπώνει κάποιο περιεχόμενο αλλάζει προωθεί το δρομέα ώστε η επόμενη εκτύπωση να γίνεται στην αρχή της επόμενης γραμμής.

```
print(<έκφραση>, <έκφραση>, .. , <έκφραση>, end = '')
```

Η παράμετρος end , στο τέλος της εντολής μπορεί να αλλάξει το αποτέλεσμα της εμφάνισης:

- `end = ''`, η εκτύπωση θα συνεχιστεί στην τρέχουσα γραμμή .
- `end = '\n'`, η εκτύπωση θα συνεχιστεί στην επόμενη γραμμή (ίδιο αποτέλεσμα με την εντολή χωρίς end).
- `end = '-'`, η εκτύπωση θα συνεχιστεί στην τρέχουσα γραμμή χωρίζοντας τα αποτελέσματα με τον χαρακτήρα `-`.

Σύνταξη της εντολής εισόδου input

```
<όνομα μεταβλητής> = input(<Μήνυμα προτροπής>)
```

Η εντολή εισόδου `input` διακόπτει τη ροή εκτέλεσης του προγράμματος εμφανίζοντας στην οθόνη το μήνυμα προτροπής. Ο χρήστης πληκτρολογεί το δεδομένο εισόδου η τιμή του οποίου εκχωρείται στο όνομα της μεταβλητής.

```
<όνομα μεταβλητής> = int(input(<Μήνυμα προτροπής>))
```

Σε περίπτωση που τα δεδομένα είναι ακέραιος αριθμός η εντολή `input` περικλείεται στην εντολή `int` για να μετατρέψει το δεδομένο που θα δώσει ο χρήστης σε ακέραιο τύπο.

```
<όνομα μεταβλητής> = float(input(<Μήνυμα προτροπής>))
```

Σε περίπτωση που τα δεδομένα είναι πραγματικός αριθμός η εντολή `input` περικλείεται στην εντολή `float` για να μετατρέψει το δεδομένο που θα δώσει ο χρήστης σε τύπο κινητής υποδιαστολής (θα γίνει αναφορά σε επόμενο κεφάλαιο).

Σε περίπτωση που το δεδομένο είναι αλφαριθμητικό (όπως πχ στο όνομα του υπαλλήλου) δεν απαιτείται κάποια μετατροπή.

Λάθη στον κώδικα

Τα λάθη που εμφανίζονται στον κώδικα ανήκουν σε 3 κατηγορίες:

α) Τα λάθη σύνταξης (syntax errors) τα οποία είναι και τα πιο συχνά και είναι αυτά τα οποία παραβιάζουν τους κανόνες σύνταξης της γλώσσας (πχ λάθος στη γραφή των εντολών, αναγραμματισμοί στα ονόματα των μεταβλητών, λάθη στοίχισης κλπ.)

β) Τα λογικά λάθη (logical errors) στα οποία ενώ ο κώδικας εκτελείται κανονικά εν τούτοις εμφανίζει λάθος αποτελέσματα λόγω κακού σχεδιασμού.

γ) Τα λάθη χρόνου εκτέλεσης (run time errors) τα οποία προκαλούν βίαιη διακοπή της εκτέλεσης του προγράμματος (λάθος στην εισαγωγή δεδομένων, πράξεις οι οποίες δεν είναι εφικτές κλπ.)

Χαρακτήρες Διαφυγής (Escape Sequences)

Χρησιμοποιούνται σε συνδυασμό με την εντολή `print`

`\n` Νέα Γραμμή

`\t` Μονός Στηλοθέτης

`\\` Όταν θέλουμε να εμφανίσουμε `\`

`\'` Όταν θέλουμε να εμφανίσουμε `'`

`\"` Όταν θέλουμε να εμφανίσουμε `''`

Σχόλια - Τεκμηρίωση (comments)

Τα σχόλια στην Python τοποθετούνται μετά τον χαρακτήρα `#`. Έτσι ότι γραφεί από το σύμβολο αυτό και μέχρι το τέλος της τρέχουσας γραμμής παρακάμπτεται από το διερμηνέα. Μαζί με την κατάλληλη επιλογή ονομάτων σταθερών, μεταβλητών κ.ά. αποτελούν τμήμα της τεκμηρίωσης του προγράμματος. Η τελευταία απευθύνεται σε αυτούς που αναπτύσσουν και συντηρούν ένα λογισμικό και περιέχει χρήσιμες πληροφορίες σχετικές με τη σχεδίαση, ανάπτυξη, υλοποίηση του λογισμικού.

Σημειώσεις

1. Ο χαρακτήρας `\` στη μέση μιας μακροσκελής εντολής επιτρέπει στον προγραμματιστή να συνεχίζει στην επόμενη γραμμή τη σύνταξη της εντολής του. Σύμφωνα με PEP 8 το μέγιστο μήκος μίας εντολής δεν πρέπει να ξεπερνά τους 79 χαρακτήρες.

2. Οι εντολές `print`, `input`, `int`, `float` κανονικά ονομάζονται συναρτήσεις (functions) για τις οποίες θα μιλήσουμε αναλυτικά σε επόμενο κεφάλαιο.

3. Η ορθή ονομασία των προγραμμάτων στην Python είναι σενάρια εντολών (scripts).

4. Η Python είναι case sensitive γλώσσα προγραμματισμού, δηλαδή έχει σημασία εάν χρησιμοποιούνται κεφαλαίοι ή πεζοί (μικροί) χαρακτήρες.

5. Οι εντολές της γλώσσας γράφονται με πεζούς χαρακτήρες.

6. Όσο επουσιώδης είναι η στοίχιση των εντολών και η τοποθέτηση κενών στο δεξιό μέρος των εντολών, τόσο ουσιώδης είναι η στοίχιση των εντολών της Python από τα αριστερά. Ενώ δείχνει δύσκολη και ενοχλητική, στην πράξη αποδεικνύεται ότι είναι πολύ χρήσιμη λειτουργικά, παράγοντας ταυτόχρονα κώδικα με υψηλό βαθμό αναγνωσιμότητας.

2.8 Ερωτήσεις

2.8.1 Ερωτήσεις κατανόησης

1. Η Python είναι μία γλώσσα προγραμματισμού που διαθέτει:
    a. Μεταφραστή
    b. Διερμηνέα
    c. Συντάκτη
    d. Όλα τα παραπάνω
    e. Τίποτα από τα παραπάνω
2. Το κέλυφος της Python χρησιμοποιείται για:
    a. Τη συγγραφή ολοκληρωμένων σεναρίων
    b. Τη μεταγλώττιση σεναρίων
    c. Εκτέλεση μεμονωμένων εντολών
    d. Όλα τα παραπάνω
3. Τα αρχεία πηγαίου κώδικα της Python έχουν επέκταση:
    a. `.pyc`
    b. `.py`
    c. `.pyt`
    d. `.pht`
    e. `.pn`
4. Η εντολή `Print('Hello World', end = '\n\n')` θα:
    a. Τυπώσει το μήνυμα Hello World και μία κενή γραμμή
    b. Τυπώσει το μήνυμα Hello World και θα μετακινήσει το δρομέα στην επόμενη
    c. Τυπώσει το μήνυμα Hello World και 2 κενές γραμμές
    d. Θα εμφανίσει μήνυμα λάθους
5. Εάν `a = 6, b = 12, c = 18` τι θα εμφανίσουν οι παρακάτω εντολές:
    a. `print(a + b + c) / 3` _____
    b. `print(a + b + c / 3)` _____

c. `print((a + b + c) / 3)` _____

d. `print('MO=', a + b + c / 3)` _____

e. `print(MO=, (a + b + c) / 3)` _____

6. Το αποτέλεσμα της εντολής `grade = input('Δώσε τη βαθμολογία του μαθητή:')` είναι να: δεχθεί από το χρήστη τη βαθμολογία και

   a. να την τοποθετήσει ως ακέραιο στη μεταβλητή `grade`

   b. να την τοποθετήσει ως πραγματικό στη μεταβλητή `grade`

   c. να την τοποθετήσει ως κείμενο στη μεταβλητή `grade`

   d. να εμφανίσει μήνυμα λάθους

7. Εκτελώ στο διερμηνέα τη μοναδική εντολή `print(hello)` θα:

   a. Εμφανίσει μήνυμα λάθους

   b. Εμφανίσει το μήνυμα **hello**

   c. Εμφανίσει το περιεχόμενο της μεταβλητής **hello**

   d. Δεν θα εμφανίσει τίποτα γιατί η **hello** είναι άγνωστη

8. Εάν η μεταβλητή `price` περιέχει την τιμή ενός προϊόντος ποιες εντολές εμφανίζουν το ζητούμενο:

   a. `print(price * 75%)` Τιμή με έκπτωση 75% ΝΑΙ ΟΧΙ

   b. `print(price * 75 / 100)` Τιμή με έκπτωση 75% ΝΑΙ ΟΧΙ

   c. `print(price * 75 / 100)` Τιμή με έκπτωση 25% ΝΑΙ ΟΧΙ

   d. `print(price * 0.75)` Τιμή με έκπτωση 25% ΝΑΙ ΟΧΙ

   e. `print(price - price * 25 / 100)` Τιμή με έκπτωση 25% ΝΑΙ ΟΧΙ

   f. `print(price - price * 25 / 100)` Τιμή με έκπτωση 75% ΝΑΙ ΟΧΙ

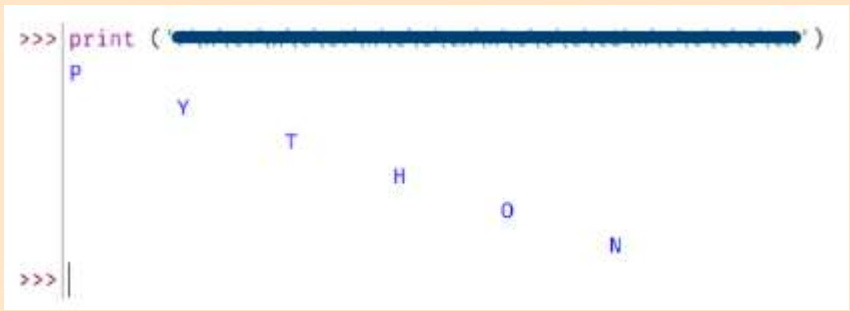
   g. `print(price - price * 0.25)` Τιμή με έκπτωση 25% ΝΑΙ ΟΧΙ

2.8.2 Ερωτήσεις ανάπτυξης

1. Για ποιο λόγο συνήθως χρησιμοποιούμε το διερμηνέα (shell) στην Python ?
2. Πως ονομάζεται το περιβάλλον ανάπτυξης σεναρίων στην Python και ποιες δυνατότητες έχει ?
3. Τι είναι τα IDE και από ποιους χρησιμοποιούνται?
4. Πως μπορεί ο προγραμματιστής να συνεχίζει στην επόμενη γραμμή μία εντολή μεγάλου μήκους ? Ποιο είναι το μέγιστο προτεινόμενο μήκος μία γραμμής εντολών στο IDLE?
5. Τι ονομάζουμε μεταβλητές σε μία γλώσσα προγραμματισμού ?
6. Ποιοι οι κανόνες ονομασίας μεταβλητών στην Python?
7. Ποια είναι η εντολή εισόδου στην Python και ποια η σύνταξή της ?
8. Ποια είναι η εντολή εξόδου στην Python και ποια η σύνταξή της ?
9. Να αναφέρετε τρεις χαρακτήρες διαφυγής στην Python δώστε παραδείγματα εντολών για τον καθένα και τι εμφανίζεται?
10. Ποιες είναι οι κατηγορίες λαθών στις γλώσσες προγραμματισμού δώστε από ένα παράδειγμα λάθους.
11. Τι σημαίνει ο όρος «Η γλώσσα είναι case sensitive». Η Python είναι case sensitive ?
12. Είναι σημαντική η αριστερή στοίχιση των εντολών στην Python?. Εξηγήστε πως επηρεάζει η τοποθέτηση περιττών κενών πριν (αριστερά) από μία εντολή της Python.

2.9 Ασκήσεις

1. (*) Στο κέλυφος (shell) της Python με μία και μοναδική εντολή `print` εμφανίστε το παρακάτω μήνυμα:



2. (*) Συντάξτε ένα σενάριο σε γλώσσα Python στο οποίο θα ζητούνται από το χρήστη με ανάλογα μηνύματα προτροπής: το βάρος του, το ύψος του. Στη συνέχεια να υπολογίζεται και να εμφανίζεται ο δείκτης βάρους/ύψους BMI. Λεπτομέρειες για το τρόπο υπολογισμού του θα βρείτε στον σύνδεσμο BMI (Body Mass Index).

3. (*) Βρείτε την τρέχουσα ισοτιμία Δολαρίου και Ευρώ και αναπτύξτε σενάριο σε γλώσσα Python το οποίο αφού ζητήσει από το χρήστη κάποιο ποσό με ανάλογο μήνυμα προτροπής σε κάποιο νόμισμα, θα το μετατρέπει και θα εμφανίζει την ισοτιμία του στο άλλο νόμισμα. Να αναπτύξετε δύο προγράμματα μετατροπής α) € → $ και β) $ → €.

4. (**) Συντάξτε σενάριο σε γλώσσα Python στο οποίο θα έχει ως σταθερά την τρέχουσα τιμή της αμόλυβδης βενζίνης ανά λίτρο και στη συνέχεια αφού ζητηθούν από το χρήστη με ανάλογα μηνύματα προτροπής, α) Η κατανάλωση βενζίνης σε λιτ/100 χλμ ενός οχήματος και β) Η απόσταση που διανύθηκε σε χλμ θα υπολογίζεται και θα εμφανίζεται το συνολικό κόστος του καυσίμου που καταναλώθηκε σε όλη την απόσταση.

5. (**) Τέσσερις (4) φίλοι έπαιξαν ένα δελτίο ΠΡΟΠΟ βάζοντας ο καθένας ένα διαφορετικό ποσό για να πληρωθεί το κόστος του δελτίου. Να φτιάξετε σενάριο σε γλώσσα Python στο οποίο θα ζητούνται: α)το ποσό συμμετοχής του καθενός φίλου στο δελτίο αυτό καθώς και β)το ποσό των κερδών. Στη συνέχεια θα υπολογίζεται και θα εμφανίζεται το ποσό που θα πάρει ο καθένας από τα κέρδη εάν αυτό υπολογιστεί αναλογικά με τη συμμετοχή του στο κόστος του δελτίου.

6. (***) Σε ένα κατάστημα πώλησης πλακιδίων ζητείται να αναπτυχθεί ένα πρόγραμμα στο οποίο θα ζητούνται:

    a) Η επιφάνεια σε τμ που επιθυμούμε να καλυφθεί με πλακίδια δαπέδου,

    b) Η διάσταση των διαθέσιμων πλακιδίων σε εκατοστά (πχ 25 x 25 ή 20 x 30 κλπ)

    c) Το κόστος των πλακιδίων ανά τμ δ) Πόσα τμ πλακιδίων περιέχει κάθε κιβώτιο.

    Στη συνέχεια θα υπολογίζονται και θα εμφανίζονται:

    a) Το πλήθος των κιβωτίων που θα χρειαστούμε,

    b) Το κόστος τους,

    c) Το ποσό του ΦΠΑ (23%) και

    d) Το τελικό πληρωτέο ποσό.

    (Σημείωση: Δεν ενοχλεί η εμφάνιση μη ακέραιου πλήθους κιβωτίων).

7. (**) Δημιουργήστε μία εφαρμογή σε γλώσσα Python υπολογισμού και εμφάνισης των μορίων στις πανελλαδικές εξετάσεις. Ο υπολογισμός του αριθμού μορίων κάθε υποψηφίου που συμμετέχει στις εξετάσεις, γίνεται ως εξής: ο γραπτός βαθμός σε καθένα από τα τέσσερα πανελλαδικά εξεταζόμενα μαθήματα τα οποία προβλέπονται στην Ομάδα Προσανατολισμού όπου ανήκει ο υποψήφιος για το συγκεκριμένο Επιστημονικό Πεδίο, πολλαπλασιάζεται με τον αντίστοιχο συντελεστή βαρύτητας, όπως αυτός καθορίστηκε με απόφαση της Συγκλήτου για κάθε Σχολή. Τα 4 ανωτέρω γινόμενα προστίθενται και το τελικό άθροισμα πολλαπλασιάζεται επί 1.000, για να προκύψει ο συνολικός αριθμός μορίων κάθε υποψηφίου. Το άθροισμα όλων των συντελεστών βαρύτητας πρέπει να είναι 1. (πχ 0.25 x 4 εάν είναι ίδιος για όλα τα μαθήματα).

2.9.1 Λύσεις των Ασκήσεων

1. Λύση Άσκησης 1
2. Λύση Άσκησης 2
3. Λύση Άσκησης 3 με μία προέκταση που θα συζητήσετε με τον εκπαιδευτή σας
4. Λύση Άσκησης 4
5. Λύση Άσκησης 5
6. Λύση Άσκησης 6
7. Λύση Άσκησης 7

Κεφάλαιο 3

3. Τύποι δεδομένων και εκφράσεις

Τελειώνοντας αυτό το μάθημα θα έχεις μάθει

- Να αναφέρεις τους περιορισμούς στην ονομασία των μεταβλητών
- Να περιγράφεις τους τρεις τύπους δεδομένων που χρησιμοποιεί η Python.
- Να αναφέρεις τουλάχιστον δύο τελεστές από κάθε κατηγορία
- Να επεξηγείς με παραδείγματα τη χρήση τελεστών
- Να ορίζεις την προτεραιότητα πράξεων σε μια παράσταση
- Να επεξηγείς το λόγο της απώλειας ακρίβειας στην τιμή πραγματικών (float)
- Να αναγνωρίζεις και να συντάσσεις σχόλια μίας ή πολλαπλών γραμμών.

3.1 Μεταβλητές

Σχεδόν πάντα σε ένα πρόγραμμα χρειαζόμαστε να αποθηκεύσουμε προσωρινά κάποια δεδομένα στη μνήμη του υπολογιστή. Για το σκοπό αυτό χρησιμοποιούμε τις μεταβλητές.

Η μεταβλητή, λοιπόν, είναι ένα συμβολικό όνομα μιας περιοχής της μνήμης στην οποία μπορούμε να γράψουμε, να τροποποιήσουμε και να ανακτήσουμε δεδομένα μέσω του ονόματος αυτού.

Παραδείγματα μεταβλητών

- `x = 8`
- `onoma = "Νίκος Δήμου"`
- `flag = True`

Από τα παραπάνω παραδείγματα καταλαβαίνουμε ότι σε μεταβλητές μπορούμε να "αποθηκεύσουμε" διαφορετικά πράγματα ή όπως λέμε στον προγραμματισμό διαφορετικούς τύπους δεδομένων. Οι τύποι αυτοί μπορεί να είναι αριθμητικοί (numerical), συμβολοσειρές (strings), λογικές (boolean) τιμές.

Το "=" αποτελεί τον τελεστή εκχώρησης μιας τιμής σε μια μεταβλητή και δεν έχει την έννοια της μαθηματικής ισότητας. Στην Python η εντολή x = x +5 έχει το νόημα "εκτέλεσε το δεύτερο μέρος της παράστασης και το αποτέλεσμα που θα βρεις δώστο (εκχώρησέ το) στο πρώτο". Άρα η το αποτέλεσμα αυτής της εντολής είναι η αύξηση της τιμής της μεταβλητής x κατά 5.

Για τα ονόματα που χρησιμοποιούμε στις μεταβλητές υπάρχουν ορισμένοι περιορισμοί.

1. Το όνομα της μεταβλητής δεν μπορεί να αρχίζει με αριθμό (π.χ το `1onoma` είναι λάθος ή με κάτω παύλα"_").
2. Το όνομα της μεταβλητής δεν μπορεί να περιέχει κάποιο ειδικό σύμβολο (π.χ $, &, % κλπ) εκτός από την κάτω παύλα "_".
3. Το όνομα της μεταβλητής δεν μπορεί να είναι ίδιο με κάποια δεσμευμένη λέξη της Python. Για παράδειγμα μια μεταβλητή δεν μπορεί αν ονομαστεί print γιατί η print είναι εντολή της Python.

Επίσης η Python διαφοροποιεί τα πεζά γράμματα από τα κεφαλαία. Η μεταβλητή `Onoma` είναι διαφορετική από το `onoma` και φυσικά και οι δυο είναι διαφορετικές από το ONOMA.

3.2 Τύποι Δεδομένων

Οι τύποι δεδομένων προσδιορίζουν τον τρόπο παράστασης των δεδομένων εσωτερικά στον υπολογιστή, καθώς και το είδος της επεξεργασίας τους από αυτόν.

Στην Python δεν χρειάζεται να δηλώσουμε τον τύπο δεδομένων μιας μεταβλητής. Η ίδια μεταβλητή μπορεί στην αρχή να αποθηκεύσει έναν αριθμό (π.χ `neo = 12`) και αργότερα μια συμβολοσειρά (π.χ `neo = "Κώστας"`)

Οι τύποι δεδομένων στη Python

Αριθμητικοί (Numerical).

Διακρίνονται σε ακεραίους (int- integer) και σε κινητής υποδιαστολής (float - floating point) που αντιστοιχούν στους πραγματικούς αριθμούς.

Συμβολοσειρές -αλφαριθμητικά (strings).

Είναι μια ακολουθία από χαρακτήρες (γράμματα, αριθμοί, σύμβολα) μπορούν να περιέχουν κενά και μπορούν να είναι με λατινικούς ή ελληνικούς χαρακτήρες. Πρέπει απαραίτητα να περικλείονται με μονά ή διπλά εισαγωγικά, αλλά όχι ανάμικτα.

Λογικοί (Boolean).

Ο τύπος αυτός δέχεται μόνο δύο τιμές, την τιμή `True` (Αληθής) και την τιμή `False` (Ψευδής).

Παραδείγματα

Μεταβλητή	Εκχώρηση τιμής	Τύπος Μεταβλητής
`x`	`x = 116`	Ακέραιος
`ypsos`	`ypsos = 1.86`	Κινητής υποδιαστολής (float)
`paxos`	`paxos – 4.3481E 4`	Float (Το E 4 σημαίνει 104)
`day`	`day ="σήμερα είναι Τρίτη"`	Συμβολοσειρά (string)
`flag`	`Flag = True`	Λογικός
`flag1`	`Flag1 = 4>7`	Λογικός (με τιμή False)

Περισσότερα για τις μεταβλητές

Τα πάντα στην Python αναπαρίστανται με αντικείμενα και κάθε αντικείμενο έχει μια ταυτότητα (identity), έναν τύπο και μία τιμή. Για παράδειγμα, στην απόδοση τιμής x = 8, το 8 είναι ένα αντικείμενο, τύπου int (ακέραιος) και τιμή 8.
Το x είναι η μεταβλητή και λειτουργεί ως ετικέτα με όνομα x και αναφέρεται (δείχνει) το αντικείμενο 8. Το x ως τύπο δεδομένων λαμβάνει τον τύπο του αντικειμένου που αναφέρεται. Στο παράδειγμα x = 8 η μεταβλητή x έχει ακέραιο τύπο δεδομένων

3.3 Αριθμητικές και λογικές εκφράσεις

Χρησιμοποιώντας μεταβλητές κάθε τύπου δεδομένων, αριθμούς και τελεστές (operators) μπορούμε να σχηματίσουμε σύνθετες εκφράσεις. Για παράδειγμα μπορούμε να εκφράσουμε ένα μαθηματικό τύπο που πραγματοποιεί κάποιο υπολογισμό.

Στη γλώσσα Python χρησιμοποιούμε τους παρακάτω βασικούς αριθμητικούς τελεστές

3.3.1 Αριθμητικοί τελεστές

Τελεστής	Σύμβολο	Παράδειγμα
Πρόσθεση	`+`	`2.5 + 3.2` δίνει `5.7`
Αφαίρεση	`-`	`17 – 12` δίνει `5`
Πολλαπλασιασμός	`*`	`4 * 6` δίνει `24`
Διαίρεση	`/`	`12 / 3` δίνει `4.0`
Ύψωση σε δύναμη	`**`	`2 ** 3` δίνει `8`
Ακέραια διαίρεση (πηλίκο)	`//`	`7 // 3` δίνει `2`
Υπόλοιπο διαίρεσης	`%`	`7 % 3` δίνει `1`

Σημείωση

Στην πράξη της διαίρεσης ακόμα αν και οι δύο αριθμοί είναι ακέραιοι το αποτέλεσμα θα είναι πραγματικός- κινητής υποδιαστολής (float).

Σε μια έκφραση όπου υπάρχουν πολλοί αριθμητικοί τελεστές υπάρχει μια ιεραρχία πράξεων. Η ιεραρχία αυτή είναι:

- Ύψωση σε δύναμη
- Πολλαπλασιασμός, διαίρεση, ακέραια διαίρεση, υπόλοιπο
- Πρόσθεση, αφαίρεση

Σε τελεστές που είναι στο ίδιο επίπεδο οι πράξεις εφαρμόζονται από αριστερά προς τα δεξιά. Σε κάθε περίπτωση είναι πολύ καλή συνήθεια σε σύνθετες εκφράσεις με πράξεις να χρησιμοποιούμε παρενθέσεις.

Παράδειγμα 3.1

```
2 * 7 // 3     δίνει 4   ((2 * 7) // 3)
2 * 7.0 % 3    δίνει 2   ((2 * 7.0) % 3)
```

Οι αριθμητικοί τελεστές `"+"` (πρόσθεση) και `"*"` (πολλαπλασιασμός) έχουν μια ιδιαίτερη λειτουργία όταν εφαρμόζονται σε συμβολοσειρές ή σε λίστες. Σε αυτές τις περιπτώσεις ο τελεστής

"+" έχει την έννοια της συνένωσης ενώ ο "*" την έννοια της επανάληψης και παράθεσης. (Οι συμβολοσειρές και οι λίστες θα αναφερθούν με λεπτομέρεια σε άλλα κεφάλαια).

Παράδειγμα 3.2

```
>>> 'Καλη' + 'μέρα'
'Καλημέρα'
>>> [2, 3, 4] + [5, 6, 7]
[2, 3, 4, 5, 6, 7]
>>> 'Καλο'* 3
'ΚαλοΚαλοΚαλο'
>>> [2, 3, 4] * 2
[2, 3, 4, 2, 3, 4]
```

3.3.2 Συγκριτικοί η σχεσιακοί τελεστές

Χρησιμοποιούνται για τη σύγκριση δύο τιμών (ή μεταβλητών ή εκφράσεων). Το αποτέλεσμα της σύγκρισης είναι μια από τις δύο λογικές τιμές `True` (Αληθής) ή `False` (Ψευδής).

Συγκριτικός Τελεστής	Σύμβολο	Παράδειγμα
Μικρότερο	<	a < 3.5
Μικρότερο ή ίσο	<=	b <= 19
Μεγαλύτερο	>	Baros > 80
Μεγαλύτερο ή ίσο	>=	Ypsos >= 1.7
Ίσο	==	x == 3
Διαφορετικό (διάφορο)	!=	y != 8

Θα πρέπει να γνωρίζουμε ότι γενικά σε μια σύγκριση συμβολοσειρών, οι αριθμοί είναι "μικρότεροι" από τα γράμματα (ελληνικά , λατινικά). Τα κεφαλαία γράμματα είναι "μικρότερα" από τα πεζά και τα λατινικά γράμματα είναι "μικρότερα" από τα ελληνικά.

Παράδειγμα 3.3

```
>>> '23' < 'ab'
True
>>> 'ab' < 'A'
False
>>> '23' < 'A'
True
>>> 'a' < 'α'
```

3.3.3 Τελεστές λογικών πράξεων (λογικοί τελεστές)

Οι τελεστές λογικών πράξεων στην Python είναι οι ακόλουθοι:

Λογικός Τελεστής	Σύμβολο	Παράδειγμα
Σύζευξη (ΚΑΙ)	`and`	`a1 and b1`
Διάζευξη (Ή)	`or`	`b2 or a2`
Άρνηση (ΟΧΙ)	`not`	`not(neo)`

Το αποτέλεσμα μιας απλής λογικής πράξης με δύο μεταβλητές Α και Β υπολογίζεται σύμφωνα με τον πίνακα.

A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Η προτεραιότητα των λογικών τελεστών είναι `not`, `and`, `or`, αλλά σε κάθε περίπτωση προτείνεται η χρήση παρενθέσεων.

3.4 Ακρίβεια πράξεων με πραγματικούς αριθμούς (float) στην Python

Στις πράξεις και συγκρίσεις με πραγματικούς αριθμούς κάποιες φορές εμφανίζεται απροσδόκητο αποτέλεσμα όπως στα παραδείγματα που ακολουθούν. Αυτό οφείλεται στο γεγονός ότι η Python αποθηκεύει τους πραγματικούς αριθμούς στο δυαδικό σύστημα αρίθμησης πριν κάνει την πράξη και στη συνέχεια μετατρέπει το αποτέλεσμα στο δεκαδικό σύστημα για να το εμφανίσει. Επίσης ένα αριθμός κινητής υποδιαστολής έχει μέχρι 16 δεκαδικά ψηφία. Αυτό έχει σαν αποτέλεσμα μικρή απώλεια ακρίβειας στις πράξεις .

Παράδειγμα 3.4

```
>>> a1 = 6.3
>>> b1 = 9
>>> a1 * b1
56.699999999999996
```

```
>>> 2 > 1.999999999999999
True
>>> 2 > 1.9999999999999999
False
```

Πολύ μεγάλοι αριθμοί

Η python δεν βάζει όρια στο μέγεθος ακεραίων αριθμών.

Παράδειγμα 3.5

```
>>> 1234 ** 56
```

```
129911902554871451941032084396235137754657820101273923843790127046242594330
550946489256784853624729020106139515647384910944921186523865849056275359066
26235291168250476992921 6
```

Στους αριθμούς κινητής υποδιαστολής (float) οι πολύ μεγάλοι ή πολύ μικροί αριθμοί γράφονται σε επιστημονική μορφή.

Παράδειγμα 3.6

Το `e + 23` έχει την έννοια ότι ο προηγούμενος αριθμός 3.90640863009858 πολλαπλασιάζεται επί 1023

Αν ήταν `e - 23` τότε ο προηγούμενος αριθμός διαιρείται με 1023

```
>>> 2345.6 ** 7
3.90640863009858e+23
```

3.5 Δημιουργία και αποτίμηση εκφράσεων

Μια έκφραση στην Python είναι ένας απλός η σύνθετος συνδυασμός από σταθερές, μεταβλητές και συναρτήσεις που συνδέονται με τελεστές και παρενθέσεις.

Η αποτίμηση μιας έκφρασης γίνεται με απόδοση τιμών στις μεταβλητές, τον υπολογισμό τυχών συναρτήσεων και στην εκτέλεση των πράξεων σύμφωνα με την ιεραρχία τους. Η τιμή μιας έκφρασης μπορεί να είναι αριθμητική, συμβολοσειρά ή λογική.

Παράδειγμα 3.7

```
>>> a = 7
>>> b = 4.3
>>> c = a + 2 > b
>>> print(c)
True
>>> d = a + 3
>>> print(d)
10
```

Παράδειγμα 3.8

Στην Φυσική στην επιταχυνόμενη κίνηση υπάρχει ο τύπος που υπολογίζει το διάστημα x που κάνει ένα κινητό με αρχική ταχύτητα υ, με επιτάχυνση a σε χρόνο t είναι.

$$x = \upsilon t + \frac{1}{2}\alpha t^2$$

Ο υπολογισμός της έκφρασης αυτής στην Python γίνεται:

```
>>> u = 2
>>> t = 30
>>> a = 3
>>> x = u * t + (1 / 2) * a * t ** 2
>>> print(x)
1410.0
```

3.6 Σχόλια εντολών ή μιας γραμμής και σχόλια πολλαπλών γραμμών

Τα προγράμματα που γράφουμε δεν είναι πάντα εύκολο να κατανοηθούν από άλλους ακόμα και από εμάς που το γράψαμε αν το διαβάζουμε μετά από ένα αρκετά μεγάλο χρονικό διάστημα. Χρειάζεται λοιπόν μέσα στα προγράμματα να υπάρχουν επεξηγηματικά σχόλια.

Τα σχόλια αγνοούνται από τους μεταγλωττιστές και τους διερμηνευτές. Είναι γραμμένα με σκοπό να παρέχουν πληροφορίες σχετικά με τη λογική που χρησιμοποιείται στον κώδικα σε άλλους προγραμματιστές ή στον εαυτό μας. Δεν συμμετέχουν στην εκτέλεση του κώδικα αλλά ενισχύουν την αναγνωσιμότητα του κώδικα.

Υπάρχουν δύο τύποι σχολίων που χρησιμοποιούνται στην Python τα σχόλια μια γραμμής και τα σχόλια πολλαπλών γραμμών

Τα σχόλια μια γραμμής μπορούν να τοποθετηθούν σχεδόν οπουδήποτε μέσα στον κώδικα και το χαρακτηριστικό τους είναι ότι αρχίζουν από τον χαρακτήρα hastag `"#"`. Ο μεταγλωττιστής (ή διερμηνευτής) αγνοεί οτιδήποτε αρχίζει με τον χαρακτήρα `"#"` μέχρι το τέλος της τρέχουσας γραμμής.

Παράδειγμα 3.9

```
# το πρόγραμμα αυτό υπολογίζει το εμβαδόν μια πλάκας σχήματος ορθογωνίου
platos = 40           #Το πλάτος της πλάκας
ypsos = 50                        # Το ύψος της πλάκας
embadon = platos * ypsos# Εμβαδόν = Πλάτος x Ύψος
print (embadon)             # Εκτύπωση εμβαδού πλάκας

# Τέλος προγράμματος
```

Τα σχόλια πολλαπλών γραμμών είναι, όπως αναφέρει και το όνομά τους, σχόλια που καταλαμβάνουν περισσότερες από μια γραμμές. Στην πράξη πρόκειται για μικρά επεξηγηματικά κείμενα. Συνήθως τα τοποθετούμε στην αρχή του προγράμματος ή μπροστά από σύνθετες συναρτήσεις για να επεξηγήσουμε τη λειτουργία τους και τη λογική που έχουμε ακολουθήσει.

Τα σχόλια αυτά αρχίζουν με τρία διπλά εισαγωγικά `"""` και τελειώνουν πάλι με τρία διπλά εισαγωγικά `"""`. Οτιδήποτε βρίσκεται μεταξύ των εισαγωγικών αγνοείται.

Παράδειγμα 3.10

```
"""Το πρόγραμμα αυτό χρησιμοποιεί δύο αρχεία. Το αρχείο με τα στοιχεία
των  πελατών με όνομα pelates και το αρχείο κινήσεων, που περιέχει τις
αγορές των πελατών, με όνομα transactions"""
```

Σημειώσεις

- Θα μπορούσαμε να έχουμε ένα σχόλιο πολλών γραμμών χωρίς τα τριπλά εισαγωγικά αν στην αρχή κάθε γραμμής τοποθετούσαμε τον χαρακτήρα #.

- Τα τριπλά εισαγωγικά όπως τα μονά και τα διπλά μπορούν να χρησιμοποιηθούν στον ορισμό μιας συμβολοσειράς. Τα τριπλά εισαγωγικά χρησιμοποιούνται όταν το περιεχόμενο της συμβολοσειράς καταλαμβάνει περισσότερες από μια γραμμές.

Σημείωση

Τα τριπλά εισαγωγικά όπως τα μονά και τα διπλά μπορούν να χρησιμοποιηθούν στον ορισμό μιας συμβολοσειράς. Τα τριπλά εισαγωγικά χρησιμοποιούνται όταν το περιεχόμενο της συμβολοσειράς καταλαμβάνει περισσότερες από μια γραμμές.

Παράδειγμα 3.11

```
comment = """Το πρόγραμμα αυτό χρησιμοποιεί δύο αρχεία. Το αρχείο με τα στοιχεία των  πελατών με όνομα pelates και το αρχείο κινήσεων, που περιέχει τις αγορές των πελατών, με όνομα transactions"""
```

Δεσμευμένες λέξεις στην Pyhton

Οι δεσμευμένες λέξεις της python (οι οποίες δεν μπορούν να χρησιμοποιηθούν ως ονόματα μεταβλητών) είναι:

```
'False', 'None', 'True', 'and', 'as', 'assert', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',
'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield'
```

3.7 Δραστηριότητες

3.7.1 Δραστηριότητα

Ποια από τα παρακάτω ονόματα μεταβλητών είναι αποδεκτά και ποια όχι.

1. `Price`
2. `Price-3`
3. `Price_4`
4. `global`
5. `pprice`
6. `2price`
7. `Πρıce`
8. `Pr$ce`
9. `_price`

3.7.2 Δραστηριότητα

Υπολογίστε αρχικά στο χαρτί και στη συνέχεια στο κέλυφος την τιμή των παραστάσεων

1. `3 + 2 * (6 // 3) * 4 - 5`
2. `4 * 2 ** 2 / 8 + 6`
3. `2 + 7 * (3 + 4) – 2 ** 2`
4. `14 % 5 – 25 % 8`

3.7.3 Δραστηριότητα

Υπολογίστε αρχικά στο χαρτί και στη συνέχεια στο κέλυφος την τιμή των παραστάσεων

1. `((3 > 7) and (12 > 5)) or (3 > 2)`
2. `(3 > 7) and ((12 > 5) or (3 > 2))`

3.7.4 Δραστηριότητα

Αν B = 12, b = 6 , h = 4 υπολογίστε το εμβαδόν του τραπεζίου με βάσεις B, b και ύψος h με χρήση του τύπου $E = \frac{(B+b)*h}{2}$

Για την δευτεροβάθμια εξίσωση $ax^2 + bx - c = 0$ όπου a = 2, b = 3 και c = -5 να υπολογίσετε τις ρίζες τις από τον τύπο $x1 = \frac{-b+\sqrt{b^2-4*a*c}}{2*a}$ και $x2 = \frac{-b-\sqrt{b^2-4*a*c}}{2*a}$

3.8 Να θυμάμαι

Μεταβλητή

Ένα συμβολικό όνομα(ετικέτα) μιας περιοχής της μνήμης του υπολογιστή. Χρησιμοποιείται για την προσωρινή αποθήκευση δεδομένων.

Τύποι μεταβλητών

Οι μεταβλητές είναι τριών τύπων 1) αριθμητικοί , 2) Συμβολοσειρές, 3) Λογικοί

Τελεστές

- Οι τελεστές είναι τριών τύπων 1) Αριθμητικοί, 2) Συγκριτικοί, 3) Λογικοί
- Προσοχή: Οι αριθμητικοί τελεστές `"+"` και `"*"` όταν εφαρμοστούν σε συμβολοσειρές έχουν ειδική σημασία.

Ακρίβεια πράξεων με αριθμούς float

Λόγω του τρόπου με τον οποίο η python αποθηκεύει του αριθμούς κινητής υποδιαστολής μπορεί σε ορισμένες πράξεις το αποτέλεσμα να είναι μια προσέγγιση του κανονικού αποτελέσματος

Μεγάλοι αριθμοί

Για την αναπαράσταση μεγάλων αριθμών κινητής υποδιαστολής η Python χρησιμοποιεί τον επιστημονικό τρόπο, ο οποίο χρησιμοποιεί δυνάμεις του 10.

Σχόλια

- Υπάρχουν τα σχόλια γραμμής τα οποία ξεκινάνε με το χαρακτήρα '#'
- Υπάρχουν επίσης τα σχόλια πολλών γραμμών που ξεκινάνε και τελειώνουν με τους χαρακτήρες `"""` (τρία διπλά εισαγωγικά)

Σημειώσεις

- Οι εντολές της γλώσσας γράφονται με πεζούς χαρακτήρες.
- Όσο επουσιώδης είναι η στοίχιση των εντολών και η τοποθέτηση κενών στο δεξιό μέρος των εντολών, τόσο ουσιώδης είναι η στοίχιση των εντολών της Python από τα αριστερά. Ενώ δείχνει δύσκολη και ενοχλητική, στην πράξη αποδεικνύεται ότι είναι πολύ χρήσιμη λειτουργικά, παράγοντας ταυτόχρονα κώδικα με υψηλό βαθμό αναγνωσιμότητας.

Κεφάλαιο 4

4. Αρθρώματα Λογισμικού, Συναρτήσεις

Τελειώνοντας αυτό το μάθημα θα έχεις μάθει

- Ότι όλα στην Python είναι αντικείμενα
- Τι είναι Συνάρτηση και ποια είναι η δομή της
- Τα πλεονεκτήματα της χρήσης των συναρτήσεων
- Τις βασικές ενσωματωμένες συναρτήσεις της γλώσσας Python
- Τι είναι άρθρωμα λογισμικού και πως φορτώνεται σε ένα πρόγραμμα
- Τα ενσωματωμένα αρθρώματα λογισμικού της γλώσσας Python
- Τη δομή και τη λειτουργία των συναρτήσεων
- Να υλοποιείς απλές συναρτήσεις και να επιστρέφεις αποτελέσματα
- Να καλείς τις συναρτήσεις που χρειάζεσαι μέσα στο πρόγραμμά σου
- Να υλοποιείς τα δικά σου αρθρώματα λογισμικού
- Πως λειτουργεί η κλήση συνάρτησης από άλλη συνάρτηση
- Τι είναι η στοίβα χρόνου εκτέλεσης
- Τις βασικές αρχές και τα πλεονεκτήματα της αρχής της αφαιρετικότητας
- Τι είναι εμβέλεια των μεταβλητών και πως λειτουργεί

4.1 Αντικείμενα στη γλώσσα Python

Στη γλώσσα προγραμματισμού Python, τα αντικείμενα (objects) αποτελούν τον πυρήνα της γλώσσας και αναπαριστούν τα δομικά στοιχεία του προγραμματισμού. Ένα αντικείμενο είναι μια συλλογή δεδομένων (μεταβλητών) και λειτουργιών (μεθόδων) που σχετίζονται με αυτά τα δεδομένα.

Στη γλώσσα Python, τα πάντα είναι αντικείμενα. Ακόμη και οι απλές μορφές δεδομένων, όπως οι αριθμοί (ακέραιοι ή κινητής υποδιαστολής) και οι συμβολοσειρές, είναι αντικείμενα. Επίσης, οι δομές δεδομένων που θα συναντήσουμε στην πορεία όπως οι λίστες, οι πλειάδες τα λεξικά τα σύνολα μέχρι και οι συναρτήσεις που θα αναφερθούν αμέσως μετά είναι επίσης αντικείμενα. Οπότε πολύ συχνά στην ορολογία των κεφαλαίων θα γίνεται αναφορά στον όρο αυτό.

Τα αντικείμενα όπως θα αναλυθεί σε επόμενο κεφάλαιο δημιουργούνται από προγραμματιστικές οντότητες οι οποίες ονομάζονται κλάσεις.

Τα αντικείμενα θα μελετηθούν αναλυτικότερα σε επόμενα κεφάλαια.

4.2 Συναρτήσεις στη γλώσσα Python

Στο 2ο Κεφάλαιο υλοποιήθηκε η «ομάδα εντολών» `card()` η οποία εμφάνιζε μία κάρτα εργασίας (business card):

```
>>> def card(name, email):
        print(40 * '-')
        print(2 * '\n')
        print(10 * ' ' , name)
        print(10 * ' ' , email)
        print(2 * '\n')
        print(40 * '-')
```

Η εκτέλεση της ομάδας εντολών με συγκεκριμένες τιμές εισόδου για τις μεταβλητές `name` και `mail` δίνει το αποτέλεσμα:

```
>>> card('Χρήστος Βασιλόπουλος', 'chris.vasilop@ellak.gr')
----------------------------------------



          Χρήστος Βασιλόπουλος
          chris.vasilop@ellak.gr



----------------------------------------
```

Μία τέτοια «ομάδα εντολών» στη γλώσσα Python ονομάζεται συνάρτηση. Οι συναρτήσεις που μπορούν να χρησιμοποιηθούν στη γλώσσα Python είναι είτε:

- Ενσωματωμένες στο αρχικό πακέτο εγκατάστασης όπου χρειάζεται απλά να κληθούν με το όνομά τους (ενσωματωμένες built-in),
- Εντός ενσωματωμένων βιβλιοθηκών/αρθρωμάτων (modules) όπου χρειάζεται πρώτα να φορτωθεί η αντίστοιχη βιβλιοθήκη και μετά να κληθεί η συνάρτηση που περιέχει με το όνομά της πριν να χρησιμοποιηθούν,
- Εντός εξωτερικών βιβλιοθηκών οι οποίες πρέπει πρώτα να εγκατασταθούν από τα αποθετήρια λογισμικού της Python, στη συνέχεια να φορτωθούν στο πρόγραμμα και σε τρίτο χρόνο να κληθούν οι συναρτήσεις που περιέχουν.
- Συναρτήσεις του χρήστη τις οποίες υλοποιεί σύμφωνα με τις ανάγκες του προγράμματός του.

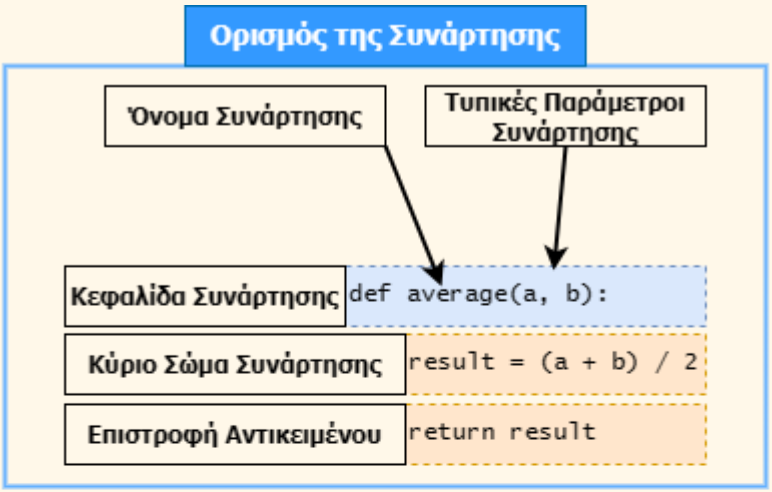
Συνάρτηση (Function) στη γλώσσα Python είναι ένα τμήμα κώδικα το οποίο εκτελεί μία συγκεκριμένη λειτουργία. Μπορεί να διαθέτει στην κεφαλίδα της είσοδο δεδομένων με τη χρήση

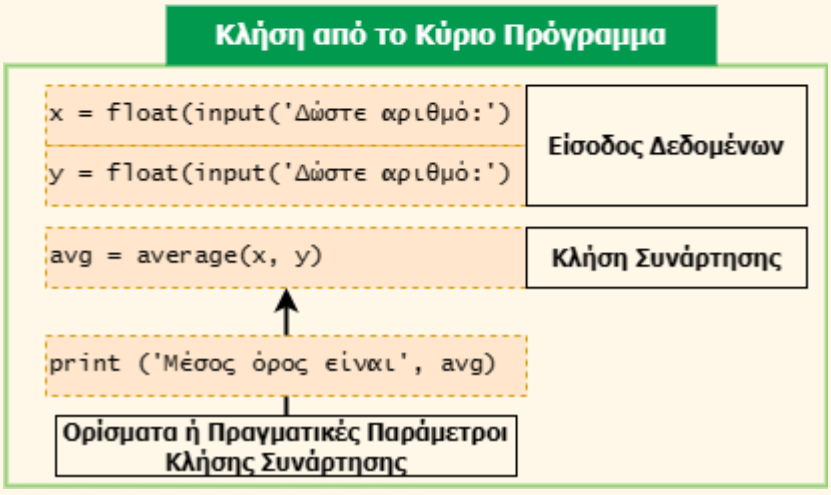
μεταβλητών οι οποίες ονομάζονται τυπικές παράμετροι (formal parameters), διαθέτει πάντα κύριο σώμα όπου υλοποιούνται οι βασικές υπολογιστικές λειτουργίες και μπορεί να επιστρέφει ένα αντικείμενο ή να εκτυπώνει αποτελέσματα ή να τροποποιεί κάποιο αντικείμενο. Ο ορισμός μίας συνάρτησης χρήστη μπορεί να γίνει μέσω της ειδικής εντολής `def` (`def`ine: ορίζω).

Η διαδικασία κατά την οποία συντάσσεται το όνομα της συνάρτησης με ή χωρίς παραμέτρους ανάλογα τις απαιτήσεις της και στη συνέχεια εκτελείται ονομάζεται **κλήση της συνάρτησης**. Κατά την κλήση της συνάρτησης από το πρόγραμμα καθορίζονται οι τιμές που θα λάβουν οι παράμετροι εάν υπάρχουν, οι τιμές αυτές που δίνονται κατά την κλήση της συνάρτησης ονομάζονται ορίσματα (arguments) ή πραγματικές παράμετροι (actual parameters).

Από τη στιγμή που θα οριστεί μία συνάρτηση και φορτωθεί στη μνήμη, μπορεί να κληθεί, είτε από το κύριο πρόγραμμα, είτε από το κέλυφος, είτε από άλλη συνάρτηση, είτε και εντός των παραμέτρων της όπως θα μελετηθεί στην πορεία του κεφαλαίου αυτού.

Κάθε φορά που καλείται μία συνάρτηση από κάποιο σημείο ενός προγράμματος ή άλλης συνάρτησης, ο έλεγχος εκτέλεσης της σειράς εντολών του προγράμματος διακόπτεται και μεταφέρεται στη συνάρτηση. Μετά το πέρας της συνάρτησης ο έλεγχος εκτέλεσης μεταφέρεται και πάλι πίσω ακριβώς μετά την εντολή που κλήθηκε αρχικά η συνάρτηση. Η λειτουργία αυτή θα περιγραφεί αναλυτικά παρακάτω στο κεφάλαιο αυτό.





4.2.1 Η χρησιμότητα και τα πλεονεκτήματα των συναρτήσεων

Οι συναρτήσεις χρησιμοποιούνται στην υλοποίηση για την οργάνωση του κώδικα σε μικρότερα, αυτόνομα τμήματα τα οποία μπορούν να κληθούν από άλλα προγράμματα ή και από άλλες συναρτήσεις.

Η χρήση συναρτήσεων κατά τη συγγραφή κώδικα παρουσιάζει πολλά πλεονεκτήματα:

- **Επαναχρησιμοποίηση κώδικα** (code reuse): Μια συνάρτηση ορίζεται μία φορά και μπορεί να κληθεί πολλές φορές σε διάφορα μέρη του προγράμματος. Αυτό επιτρέπει την αποφυγή της επανάληψης συγγραφής του ίδιου κώδικα και διευκολύνει τη συντήρηση και αναβάθμιση του προγράμματος. Επίσης, συναρτήσεις που επιτελούν συχνές (δημοφιλείς) λειτουργίες μπορούν να αποτελέσουν μέρη αρθρωμάτων λογισμικού και να χρησιμοποιηθούν και από άλλους προγραμματιστές.
- **Δομή και οργάνωση του κώδικα** (code structure): Οι συναρτήσεις βοηθούν στον διαχωρισμό του κώδικα σε λογικά αυτόνομες μονάδες. Αυτό καθιστά τον κώδικα πιο οργανωμένο, πιο ευανάγνωστο και πιο εύκολο στη συντήρηση.
- **Αφαίρεση πολυπλοκότητας** (abstraction): Με τη χρήση συναρτήσεων, μειώνεται η πολυπλοκότητα του κύριου κώδικα. Ενώ μία συνάρτηση μπορεί να έχει σύνθετη λειτουργικότητα η απλή κλήση της από τον κύριο κώδικα κάνει το κύριο πρόγραμμα πιο κατανοητό και ευκολότερο στη διόρθωση και στη συντήρησή του.

Οι **συναρτήσεις** γενικά στις γλώσσες προγραμματισμού ανήκουν στα **ΥΠΟ-προγράμματα** και μαζί με τις **διαδικασίες** (procedures) και αποτελούν τη βάση του τμηματικού προγραμματισμού και εργαλείο της ιεραρχικής σχεδίασης προγραμμάτων. Στη γλώσσα Python δεν υπάρχουν όμως διαδικασίες παρά μόνο συναρτήσεις.

Η **ιεραρχική σχεδίαση** επίλυσης ενός σύνθετου συνήθως προβλήματος περιλαμβάνει τη διαδικασίας διάσπασής του σε μικρότερα (και άρα απλούστερα) προβλήματα τα οποία είναι πιο εύκολο να επιλυθούν και η λύση των οποίων επιλύει και το αρχικό πρόβλημα. Η τεχνική αυτή σχεδίασης ονομάζεται και από επάνω προς τα κάτω (top down design). Υλοποιείται με τον **τμηματικό προγραμματισμό** όπου επιλύεται κάθε υποπρόβλημα με ένα υποπρόγραμμα (στην Python με συνάρτηση). Η σύνθεση των επιμέρους υποπρογραμμάτων οδηγεί και στη λύση του αρχικού προβλήματος.

Ο τμηματικός προγραμματισμός μειώνει τα λάθη, επιτρέπει την ευκολότερη παρακολούθηση, κατανόηση και συντήρηση του προγράμματος από τρίτους.

4.3 Ενσωματωμένες συναρτήσεις στο περιβάλλον της γλώσσας

Η γλώσσα προγραμματισμού Python προσφέρει μια πληθώρα ενσωματωμένων συναρτήσεων (άμεσα διαθέσιμες στον προγραμματιστή), εκτελώντας βασικές λειτουργίες που παρέχονται **απευθείας** από τη γλώσσα χωρίς την ανάγκη εισαγωγής κάποιας επιπλέον βιβλιοθήκης. Αυτές οι ενσωματωμένες συναρτήσεις παρέχουν έτοιμες λειτουργίες και χρησιμοποιούνται ευρέως για την εκτέλεση διαφόρων εργασιών και την επεξεργασία δεδομένων. Κάποιες από τις βασικές ενσωματωμένες συναρτήσεις είναι:

Η συνάρτηση print()

Η συνάρτηση `print()` χρησιμοποιείται για την εκτύπωση κειμένου ή μεταβλητών στην οθόνη. Αυτή η συνάρτηση είναι πολύ χρήσιμη για την εμφάνιση αποτελεσμάτων, μηνυμάτων και αναφορών κατά την εκτέλεση του προγράμματος. Μπορεί να χρησιμοποιηθεί για να εμφανιστεί απλό κείμενο ή για την τιμή μιας μεταβλητής ή μεταβλητών ή και για τον υπολογισμό μίας έκφρασης. Επίσης, μπορεί να συνδυάσει κείμενο και μεταβλητές χρησιμοποιώντας τον τελεστή `"+"` για τη συνένωση.

Παράδειγμα 4.1

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης print

```
name = 'Python'
ver = '3.11.3'
print('Χρησιμοποιούμε τη γλώσσα ' + name + ' ' + ver)
Χρησιμοποιούμε τη γλώσσα Python 3.11.3
```

Η συνάρτηση input():

Η συνάρτηση `input()` χρησιμοποιείται για να ληφθούν δεδομένα εισόδου από τον χρήστη. Όταν καλείται αυτή η συνάρτηση, η εκτέλεση του προγράμματος σταματά περιμένοντας από τον χρήστη να πληκτρολογήσει δεδομένα από το πληκτρολόγιο. Ο χρήστης μπορεί να εισάγει μια συμβολοσειρά ή έναν αριθμό, τα οποία επιστρέφονται με μορφή κειμένου ως αποτέλεσμα της συνάρτησης input().

Παράδειγμα 4.2

Συντάξτε και εκτελέστε για να δείτε τη χρήση της συνάρτησης `input`

```
name = input('Δώστε το όνομά σας:')
print('Καλησπέρα', name)
number = input('Δώστε έναν αριθμό:')
print('Το διπλάσιο του αριθμού είναι', 2 * number)
```

Η συνάρτηση len():

Η συνάρτηση `len()` χρησιμοποιείται για να υπολογισθεί το μήκος ενός αντικειμένου, όπως μιας συμβολοσειράς, μίας λίστας ή μίας πλειάδας (δομές που θα μελετηθούν σε επόμενα κεφάλαια). Επιστρέφει τον αριθμό των στοιχείων που περιέχει το αντικείμενο. Αυτή η συνάρτηση είναι

χρήσιμη για τον έλεγχο του «μεγέθους» ενός αντικειμένου πριν εκτελεσθούν κάποιες λειτουργίες ή πριν την επεξεργασία του.

Παράδειγμα 4.3

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `len`

```
>>> name = 'Αναστάσιος'
>>> print(len(name))
10
# Δημιουργία μίας δομής τύπου λίστας που θα αναφερθεί σε επόμενο κεφάλαιο
>>> weekdays = ['Δευτέρα', 'Τρίτη', 'Τετάρτη', 'Πέμπτη', 'Παρασκευή',
'Σάββατο', 'Κυριακή']
>>> print(len(weekdays))
7
```

Η συνάρτηση abs():

Η συνάρτηση `abs()` χρησιμοποιείται για την επιστροφή της απόλυτης τιμής ενός αριθμού. Απόλυτη τιμή ενός αριθμού είναι η απόστασή του από το μηδέν στον αριθμητικό άξονα, χωρίς να λαμβάνεται υπόψη το πρόσημό του.

Παράδειγμα 4.4

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `abs()`

```
>>> num1 = 8
>>> num2 = -8
>>> print(abs(num1), abs(num2))
8 8
```

Η συνάρτηση int():

Η συνάρτηση `int()` χρησιμοποιείται για τη μετατροπή ενός αντικειμένου σε ακέραιο (integer) τύπο δεδομένων. Επιστρέφει τον ακέραιο αριθμό που αντιστοιχεί στην είσοδο της συνάρτησης.

Παράδειγμα 4.5

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `int()`

```
>>> number_string = "18"
>>> print(number_string * 2)
1818
>>> number_int = int(number_string)
>>> print(number_int * 2)
36
```

Η συνάρτηση float():

Η συνάρτηση `float()` χρησιμοποιείται για τη μετατροπή ενός αντικειμένου σε τύπο δεδομένων αριθμού κινητής υποδιαστολής (πραγματικό αριθμό). Επιστρέφει τον πραγματικό αριθμό που αντιστοιχεί στην είσοδο.

Παράδειγμα 4.6

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `float()`

```
>>> number_string = "18.6"
>>> print(number_string * 2)
18.618.6
>>> number_float = float(number_string)
>>> print(number_float * 2)
37.2
```

Η συνάρτηση str():

Η συνάρτηση `str()` χρησιμοποιείται για τη μετατροπή ενός αντικειμένου σε αντικείμενο συμβολοσειράς (string). Επιστρέφει μια συμβολοσειρά που αναπαριστά το αντικείμενο.

Παράδειγμα 4.7

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `str()`

```
>>> number = 34
>>> print(number * 2)
68
>>> number_string = str(number)
>>> print(number_string * 2)
3434
```

Η συνάρτηση bool():

Η συνάρτηση `bool()` χρησιμοποιείται για τη μετατροπή ενός αντικειμένου σε αντικείμενο λογικού τύπου (boolean). Κάθε τέτοιο αντικείμενο έχει ΜΟΝΟ δύο λογικές τιμές (`True` ή `False`). Αν η είσοδος είναι ένας ΜΗ μηδενικός αριθμός ή μια ΜΗ κενή συμβολοσειρά, η `bool()` επιστρέφει `True`. Σε άλλες περιπτώσεις, όπως για μηδενικό αριθμό ή κενή συμβολοσειρά, επιστρέφει `False`.

Παράδειγμα 4.8

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `bool()`

```
>>> a = 12
>>> b = 'Hello'
>>> c = 0
>>> d = ''
>>> print(bool(a), bool(b), bool(c), bool(d))
True True False False
```

Η συνάρτηση type():

Η συνάρτηση `type()` χρησιμοποιείται για την επιστροφή του τύπου δεδομένων ενός αντικειμένου. Επιστρέφει μια αναπαράσταση του τύπου ως αντικείμενο.

Παράδειγμα 4.9

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `type()`

```
>>> a = 14
>>> b = 3.14
>>> c = 'Python'
>>> d = True
>>> print(type(a), type(b), type(c), type(d))
<class 'int'> <class 'float'> <class 'str'> <class 'bool'>
```

Η συνάρτηση pow()

Η συνάρτηση `pow()` χρησιμοποιείται για τον υπολογισμό της δύναμης ενός αριθμού. Δέχεται δύο ορίσματα: τη βάση και τον εκθέτη, και επιστρέφει το αποτέλεσμα της εκθετικής λειτουργίας τους. Η συνάρτηση `pow()` εκτελεί παρόμοια λειτουργία με τον αριθμητικό τελεστή ύψωσης σε δύναμη `**`

Παράδειγμα 4.10

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `pow()`

```
>>> result = pow(2, 8)
>>> print(result)
256
>>> print (2 ** 8)
256

>>> riza3 = pow(27, 1/3)
>>> print(riza3)
3.0
```

Η συνάρτηση divmod():

Η συνάρτηση `divmod()` όταν δέχεται ακεραίους αριθμούς χρησιμοποιείται για την εκτέλεση της ακέραιας (ευκλείδειας διαίρεσης) διαίρεσης:

- `divmod(x, y) = a, b ⇔ y * a + b = x`

Επιστρέφει ένα ζεύγος τιμών (αντικείμενο πλειάδας: tuple, όπως θα αναφερθεί σε επόμενο κεφάλαιο), το οποίο αποτελείται από το ακέραιο πηλίκο και το ακέραιο υπόλοιπο της διαίρεσης των αριθμών εισόδου.

Σημείωση

Το ακέραιο πηλίκο και το ακέραιο υπόλοιπο μπορεί να ληφθεί χρησιμοποιώντας και τους τελεστές `//`, `%`. *Η ευκλείδεια διαίρεση είναι ιδιαίτερα χρήσιμη όπως θα αναφερθεί σε επόμενο κεφάλαιο σε*

υπολογισμούς πχ για εύρεση αρτίων ή περιττών ακεραίων, πολλαπλασίων αριθμού, διαχωρισμό σε ποσότητες ακέραιες:

- *πόσες τάξεις μέγιστου αριθμού των 17 ατόμων δημιουργούνται από 65 μαθητές*
- *πόσα χαρτονομίσματα των 50, 20, 10 € θα επιστραφούν για ποσό ανάληψης 180 € από το ATM της τράπεζας*

Παράδειγμα 4.11

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `divmod()`

```
>>> print(divmod(7, 4))
(1, 3)
>>> print( 7 // 4, 7 % 4)
1 3
>>> print(divmod(8, 4))
(2, 0)
```

Η συνάρτηση round():

Η συνάρτηση `round()` χρησιμοποιείται για τη στρογγυλοποίηση ενός αριθμού στον πλησιέστερο ακέραιο. Μπορεί να δεχθεί δύο ορίσματα: τον αριθμό που θα στρογγυλοποιηθεί και τον αριθμό των δεκαδικών ψηφίων που θα διατηρηθούν (προαιρετικό). Επιστρέφει τον αριθμό που θα προκύψει μετά τη στρογγυλοποίηση.

Παράδειγμα 4.12

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `round()`

```
>>> pi = 3.14159
>>> r0 = round(pi, 0)
>>> r1 = round(pi, 1)
>>> r2 = round(pi, 2)
>>> r3 = round(pi, 3)
>>> r4 = round(pi)
>>> print(r0, r1, r2, r3, r4)
3.0 3.1 3.14 3.142 3
```

Παρατήρηση

Η προκαθορισμένη παράμετρος της συνάρτησης είναι 0 επιστρέφοντας όμως ακέραιο όπως φαίνεται από την τελευταία εντολή:

- `round(pi)` → `3`
- `round(pi, 0)` → `3.0`

Η συνάρτηση bin():

Η συνάρτηση `bin()` χρησιμοποιείται για τη μετατροπή ενός ακεραίου αριθμού στη δυαδική του (binary) αναπαράσταση. Επιστρέφει μια συμβολοσειρά που αντιστοιχεί στον δυαδικό αριθμό του ορίσματος εισόδου.

Παράδειγμα 4.13

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `bin()`

```
>>> num = 131
>>> print(bin(131))
0b10000011
```

Η συνάρτηση hex():

Η συνάρτηση `hex()` χρησιμοποιείται για τη μετατροπή ενός ακεραίου αριθμού στη δεκαεξαδική του (hexadecimal) αναπαράσταση. Επιστρέφει μια συμβολοσειρά που αντιστοιχεί στον δεκαεξαδικό αριθμό της εισόδου.

Παράδειγμα 4.14

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `hex()`

```
>>> number_hex = hex(45)
>>> print(number_hex)
0x2d
```

Σημείωση

Η συνάρτηση `int()` *που αναφέρθηκε μπορεί να χρησιμοποιηθεί για τη μετατροπή ενός αριθμού από ένα αριθμητικό σύστημα πχ δυαδικό ή δεκαεξαδικό στο δυαδικό.*

Παράδειγμα 4.15

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `int()` *για μετατροπή αριθμού από δεκαεξαδικό ή δυαδικό στο δεκαδικό σύστημα*

```
>>> number_hex = hex(45)
>>> print(number_hex)
0x2d
>>> number_int = int(number_hex, 16)
>>> print(number_int)
45
>>> number_bin = bin(45)
>>> print(number_bin)
0b101101
>>> number_int = int(number_bin, 2)
>>> print(number_int)
45
```

Η συνάρτηση eval():

Η συνάρτηση `eval()` χρησιμοποιείται για την αποτίμηση (evaluation) μίας συμβολοσειράς η οποία περιέχει κώδικα (σε μορφή κειμένου) και επιστρέφει το αποτέλεσμα της εκτέλεσης του κώδικα αυτού.

Παράδειγμα 4.16

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `eval()`

```
>>> eval('print(34 * 47)')
1598
>>> eval("divmod(11, 3)")
(3, 2)
```

Η συνάρτηση help():

Η συνάρτηση `help()` δέχεται ως παράμετρο ένα αντικείμενο και χρησιμοποιείται για τη λήψη βοήθειας σχετικά με τις λειτουργίες του. Η βοήθεια που εμφανίζεται απορρέει από την τεκμηρίωση που έχει γραφεί για το αντικείμενο αυτό:

- Από τους δημιουργούς της γλώσσας εάν πρόκειται για ενσωματωμένη λειτουργία
- Από τους προγραμματιστές του αντικειμένου εάν πρόκειται για εξωτερική λειτουργία

Παράδειγμα 4.17

Συντάξτε και εκτελέστε για τη χρήση της συνάρτησης `help()`

```
>>> help(divmod)
Help on built-in function divmod in module builtins:

divmod(x, y, /)
    Return the tuple (x//y, x%y).  Invariant: div*y + mod == x.

>>> help(bin)
Help on built-in function bin in module builtins:

bin(number, /)
    Return the binary representation of an integer.

    >>> bin(2796202)
    '0b1010101010101010101010'

>>> help(print)
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

Σημείωση

Ήδη σε όλα σχεδόν σε όλα τα προηγούμενα παραδείγματα έγινε εμφανές ότι μία συνάρτηση μπορεί να δέχεται ως παράμετρο μία άλλη συνάρτηση. Στα παραδείγματα που ακολουθούν η εντολή `print` η οποία είναι μία συνάρτηση δέχεται στις τυπικές παραμέτρους της άλλες συναρτήσεις:

```
>>> print(len(weekdays))
>>> print(type(a), type(b), type(c), type(d))
>>> print(divmod(7, 4))
>>> help(divmod)
```

4.4 Αρθρώματα Λογισμικού

Η γλώσσα προγραμματισμού Python είναι γνωστή για την ευκολία χρήσης, την ευανάγνωστη σύνταξη και την ευελιξία της. Ένα από τα πλεονεκτήματα της Python είναι η πληθώρα αρθρωμάτων λογισμικού (modules) που προσφέρονται από την κοινότητα υποστήριξής της. Τα αρθρώματα λογισμικού γενικά είναι κομμάτια λογισμικού που παρέχουν έτοιμες λειτουργίες και λύσεις για διάφορες προγραμματιστικές ανάγκες. Τα αρθρώματα λογισμικού που έχει ο χρήστης τη δυνατότητα να χρησιμοποιήσει στο πρόγραμμά του μπορεί να είναι:

- ενσωματωμένα στη γλώσσα (built-in)
- σε εξωτερικές βιβλιοθήκες λογισμικού τις οποίες πρέπει να κατεβάσει και να εγκαταστήσει πριν τις χρησιμοποιήσει από τα αποθετήρια λογισμικού
- δικής του υλοποίησης που θα ενσωματώνει στο πρόγραμμά του

Στη συνέχεια θα μελετηθεί:

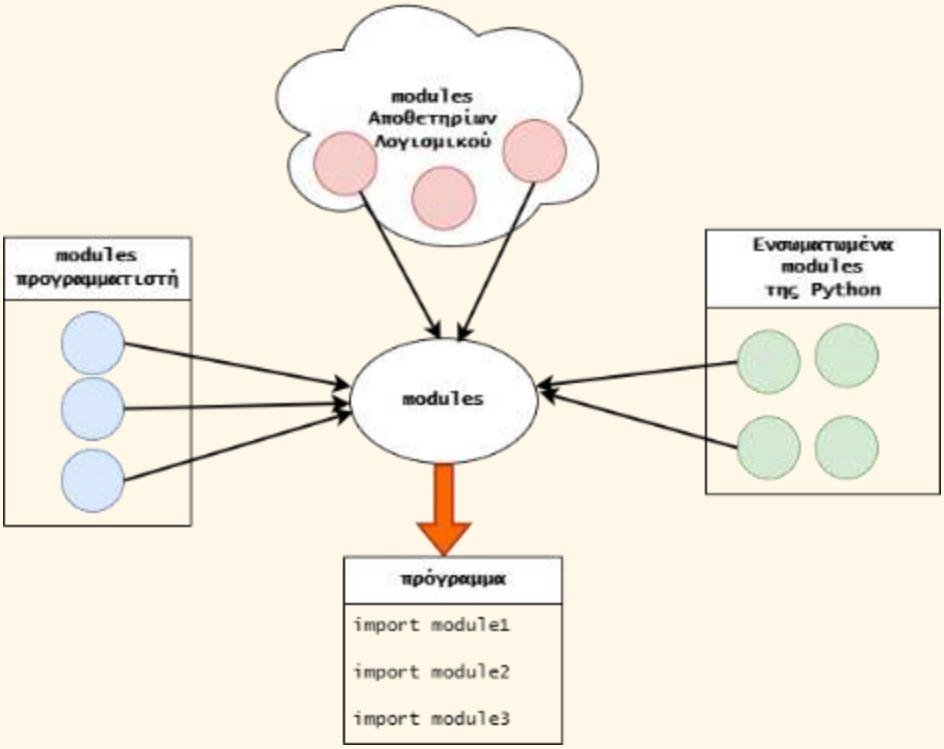
- πως γίνεται η φόρτωση των ενσωματωμένων αρθρωμάτων
- πως γίνεται η χρήση τους
- πως επεκτείνονται οι δυνατότητες της γλώσσας με αυτά
- πως χρησιμοποιούνται ορισμένες λειτουργίες από τα βασικά αρθρώματα λογισμικού

Σημείωση

Σε αυτό το κεφάλαιο θα γίνει αναφορά ΜΟΝΟ στα ενσωματωμένα αρθρώματα λογισμικού και στα αρθρώματα χρήστη

Παράδειγμα 4.18

Ο προγραμματιστής όπως φαίνεται στο σχήμα που ακολουθεί έχει τη δυνατότητα να φορτώνει αρθρώματα λογισμικού για χρήση στο πρόγραμμά του προερχόμενα από διάφορες πηγές.



4.4.1 Φόρτωση ενσωματωμένων αρθρωμάτων

Στην Python, τα ενσωματωμένα αρθρώματα λογισμικού είναι συλλογές από συναρτήσεις, κλάσεις και μεθόδους που παρέχουν επιπλέον λειτουργικότητα στη γλώσσα. Αυτά τα αρθρώματα είναι ενσωματωμένα στο βασικό πακέτο εγκατάστασης και μπορούν να φορτωθούν και να χρησιμοποιηθούν απευθείας στις εφαρμογές του χρήστη.

Για τη χρήση ενός αρθρώματος (module) στο πρόγραμμα χρειάζεται πρώτα να φορτωθεί αυτό πριν τη χρήση του. Ως παράδειγμα θα χρησιμοποιηθεί το module `math`. Με παρόμοιο τρόπο μπορεί να φορτωθεί οποιοδήποτε από τα υπόλοιπα αρθρώματα λογισμικού που θα αναφερθούν.

Το άρθρωμα math:

Το άρθρωμα `math` παρέχει μια σειρά από μαθηματικές λειτουργίες και συναρτήσεις. Με το άρθρωμα math, γίνονται διαθέσιμα για χρήση στο πρόγραμμα, μαθηματικά εργαλεία υπολογισμού (συναρτήσεις/ μέθοδοι) που είναι χρήσιμα στην υπολογιστική γεωμετρία, την υπολογιστική τριγωνομετρία, τον μαθηματικό λογισμό κα. Το άρθρωμα `math` είναι ιδιαίτερα χρήσιμο για επιστημονικές και μηχανικές εφαρμογές, καθώς και για υπολογισμούς που αφορούν στην ακρίβεια και στην προσέγγιση των αριθμητικών τιμών.

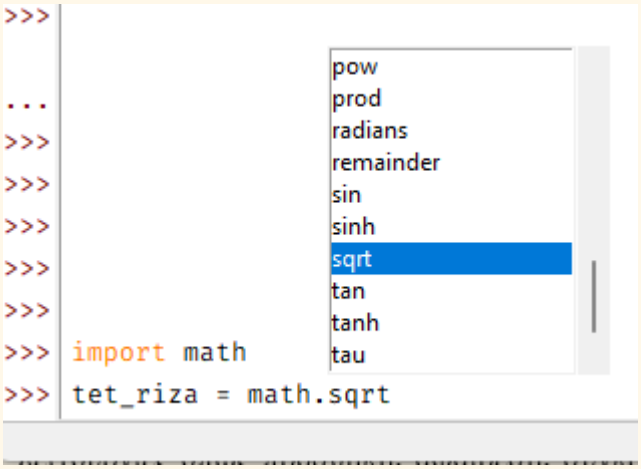
Παράδειγμα 4.19

Συντάξτε και εκτελέστε τις παρακάτω εντολές για φόρτωση και χρήση του αρθρώματος `math`

1ος Τρόπος Φόρτωσης και χρήσης ΟΛΟΥ του αρθρώματος

```
# Φόρτωση ολόκληρου του αρθρώματος math μαζί με τις συναρτήσεις του
import math
# Κλήση και χρήση με τη σημειογραφίας τελείας (dot notation) της συνάρτησης
# υπολογισμού της τετραγωνικής ρίζα αριθμού
tet_riza = math.sqrt(25)
print(tet_riza)
5.0
# Κλήση από το άρθρωμα math του αριθμού π
print(math.pi)
3.141592653589793
```

Χρήση της σημειογραφίας τελείας στο κέλυφος



2ος Τρόπος Φόρτωσης και χρήσης ΜΟΝΟ τμηματικών λειτουργιών (συναρτήσεων) του αρθρώματος

```
# Φόρτωση από το άρθρωμα math μόνο της συνάρτησης υπολογισμού
# της τετραγωνικής ρίζας και του αριθμού π
from math import sqrt, pi
 # Απευθείας κλήση των λειτουργιών χωρίς σημειογραφία τελείας
print(sqrt(81))
9.0
print(pi)
3.141592653589793
```

3ος Τρόπος Φόρτωσης και χρήσης ΟΛΩΝ των επιμέρους λειτουργιών (συναρτήσεων) του αρθρώματος

```
from math import *
# Απευθείας κλήση των λειτουργιών χωρίς σημειογραφία τελείας
print(sqrt(81))
9.0
print(pi)
3.141592653589793
```

Σημείωση

Δομή της λίστας στη γλώσσα Python*. Σε επόμενο κεφάλαιο (αυτό των Δομών Δεδομένων) θα παρουσιαστεί αναλυτικά η Δομή Δεδομένων της λίστας. Έως τότε όμως επειδή θα χρησιμοποιείται συχνά η ορολογία* ***λίστα****, αρκεί να γνωρίζετε ότι στην Python μπορεί να δημιουργηθεί μία συλλογή στοιχείων (αντικειμένων) μέσα σε αγκύλες που διαχωρίζονται με κόμμα (,) η οποία ονομάζεται λίστα:*

Παράδειγμα 4.20

Συντάξτε και εκτελέσετε για τη δημιουργία λιστών τιμών

```
# Δημιουργία λιστών στοιχείων συμβολοσειράς, ακεραίων, πραγματικών
>>> rgb_colors = ['Red', 'Green', 'Blue']
>>> dice_numbers = [1, 2, 3, 4, 5, 6]
>>> grades = [12.5, 17.8, 19.2, 10]
>>> grades += [15]
>>> print(grades)
[12.5, 17.8, 19.2, 10, 15]
>>> print(type(rgb_colors), type(dice_numbers), type(grades))
<class 'list'> <class 'list'> <class 'list'>
```

Το άρθρωμα random:

Το άρθρωμα random παρέχει λειτουργίες για τη δημιουργία τυχαίων τιμών. Μπορεί να χρησιμοποιηθεί για δημιουργία τυχαίων αριθμών, τυχαίων λιστών, για να εκτελεστούν διάφορες λειτουργίες όπως τυχαία αναδιάρθρωση στοιχείων (ανακάτεμα), επιλογή τυχαίων στοιχείων και προσομοίωση πιθανοτήτων. Αυτό το άρθρωμα είναι χρήσιμο για παιχνίδια, προσομοιώσεις, τυχαίες επιλογές και παραγωγή τυχαίων δεδομένων για δοκιμές ελέγχου στον κώδικα του προγραμματιστή.

Παράδειγμα 4.21

Συντάξτε και εκτελέστε τις παρακάτω εντολές για τη φόρτωση και τη χρήση του αρθρώματος `random`

```
>>> import random
# Επιστροφή ακεραίου από κλειστό διάστημα
>>> dice = random.randint(1, 6)
>>> print(dice)
2

# Επιστροφή πραγματικού αριθμού στο διάστημα [0, 1)
>>> number = random.random()
>>> print(number)
0.4115632908214132

# Επιστροφή τυχαίου στοιχείου από μία λίστα στοιχείων
>>> rgb_colors = ['Red', 'Green', 'Blue']
>>> color = random.choice(rgb_colors)
>>> print(color)
Red
```

Το άρθρωμα datetime:

Το άρθρωμα `datetime` παρέχει λειτουργίες για το χειρισμό και την εργασία με ημερομηνίες και ώρες. Μπορεί να δημιουργήσει αντικείμενα που αναπαριστούν ημερομηνίες, ώρες, χρονικά διαστήματα και να εκτελέσει διάφορες λειτουργίες όπως προσθήκη, αφαίρεση, σύγκριση και μορφοποίηση. Αυτό το άρθρωμα είναι ιδιαίτερα χρήσιμο για εφαρμογές που απαιτούν τον υπολογισμό του χρόνου, όπως συστήματα κρατήσεων, καταγραφή δεδομένων και αναφορές.

Παράδειγμα 4.22

Συντάξτε και εκτελέστε τις παρακάτω εντολές για φόρτωση και χρήση του αρθρώματος `datetime`

```
# Εισαγωγή αρθρώματος datetime
import datetime

# Εμφάνιση τρέχουσας ημ/νιας και ώρας
now = datetime.datetime.now()
print(now)
2023-06-15 14:40:53.640659

# Εμφάνιση τρέχουσας ημ/νίας
date = datetime.date.today()
print(date)
2023-06-15

# Εμφάνιση τρέχοντος Έτους
year = datetime.date.today().year
print(year)
2023
```

```
# Εμφάνιση τρέχοντος μήνα
month = datetime.date.today().month
print(month)
6

# Εμφάνιση τρέχουσας ημέρας
day = datetime.datetime.now().day
print(day)
15

# Εμφάνιση τρέχουσας ώρας
hour = datetime.datetime.now().hour
print(hour)
14

# Εμφάνιση τρέχοντος λεπτού
minute = datetime.datetime.now().minute
print(minute)
48

# Εμφάνιση τρέχοντος δευτερόλεπτου
second = datetime.datetime.now().second
print(second)
34
```

Το άρθρωμα time:

Το ενσωματωμένο άρθρωμα `time` παρέχει λειτουργίες για τη χρονομέτρηση και γενικά το χειρισμό του χρόνου. Δίνει τη δυνατότητα προσθήκης καθυστέρησης στην εκτέλεση του κώδικα, μέτρησης της διάρκειας εκτέλεσής του, όπως και τοποθέτησης χρονικών σφραγίδων (timestamp).

Παράδειγμα 4.23

Συντάξτε και εκτελέστε τις παρακάτω εντολές οι οποίες υπολογίζουν το χρόνο εκτέλεσης ενός τμήματος κώδικα στη γλώσσα Python

```
import time
start = time.time() # Καταγραφή της ακριβούς ώρας εκκίνησης
# Εδώ τοποθετείται ο κώδικας προς εκτέλεση
print('Code running ...')
end = time.time()   # Καταγραφή της ώρας τερματισμού
execution_time = end - start  # Διάρκεια εκτέλεσης του κώδικα
print('Ο κώδικας εκτελέστηκε σε', execution_time, 'δευτερόλεπτα')
```

Παράδειγμα 4.24

Συντάξτε και εκτελέστε τις παρακάτω εντολές οι οποίες προσθέτουν καθυστέρηση χρόνου στη διάρκεια εκτέλεσής τους.

```
import time
print('Ένα')
```

```
print('Δύο')
time.sleep(3)   # Καθυστέρησε 3 δευτερόλεπτα (secs)
print('Τρία')
time.sleep(0.100) # Καθυστέρησε 100 χιλιοστά του δευτερολέπτου (100 msecs)
print('Τέσσερα')
```

Παράδειγμα 4.25

Συντάξτε και εκτελέστε τις παρακάτω εντολές οι οποίες εμφανίζουν τον τρέχοντα χρόνο (timestamp) σε μορφή αριθμού και σε αναγνώσιμη μορφή από το χρήστη.

```
>>> import time
>>> timestamp = time.time()
>>> print(timestamp)
1687002162.368047
>>> local_time = time.ctime(time_stamp)
>>> print(local_time)
Sat Jun 17 14:42:42 2023
```

Το άρθρωμα os:

Το άρθρωμα `os` παρέχει λειτουργίες για το σύστημα λειτουργίας του ΗΥ. Με τη χρήση του είναι εφικτή η αλληλεπίδραση με το σύστημα αρχείων του λειτουργικού συστήματος (ΛΣ). Έτσι μπορούν να δοθούν εντολές Python για: δημιουργία, μετονομασία, αντιγραφή φακέλων και αρχείων όπως και εκτέλεσης εντολών του κελύφους του ΛΣ. Περισσότερα για το άρθρωμα αυτό θα βρείτε και στο κεφάλαιο διαχείρισης αρχείων. Αυτό το άρθρωμα είναι ιδιαίτερα χρήσιμο για εφαρμογές που απαιτούν τη διαχείριση αρχείων, την εκτέλεση εντολών και την αλληλεπίδραση με το λειτουργικό σύστημα.

Παράδειγμα 4.26

Συντάξτε και εκτελέστε τις παρακάτω εντολές για τη φόρτωση και τη χρήση του αρθρώματος `os`

```
# Εισαγωγή αρθρώματος os
import os
# Εμφάνιση ονόματος ΛΣ
print(os.name)
posix

# Εμφάνιση τρέχοντος καταλόγου εργασίας
print(os.getcwd())
/home/teacher

# Διαγραφή αρχείου από τον τρέχοντα κατάλογο
os.remove('file.txt')
# Εμφάνιση περιεχομένων τρέχοντος καταλόγου
```

```
os.listdir()

# Δημιουργία καταλόγου

os.mkdir('temp')

# Αλλαγή καταλόγου

os.chdir('/home')
```

4.4.2 Δραστηριότητα

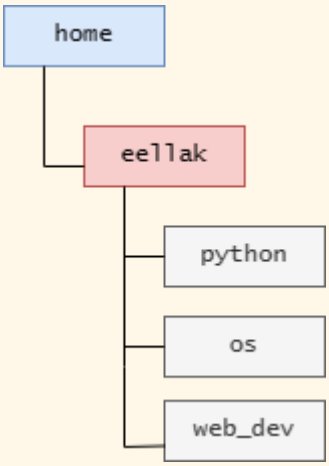
Δώστε τις κατάλληλες εντολές στο κέλυφος ή υλοποιήστε τα κατάλληλα προγράμματα ώστε:

- *Να εμφανίσετε ένα τυχαίο πραγματικό αριθμό στο διάστημα [0, 20]*
- *Να δέχεστε τα όρια ενός διαστήματος ακραίων και να εμφανίζετε ένα τυχαίο αριθμό στο διάστημα αυτό.*
- *Να εμφανίζετε μία τυχαία επιλογή κάθε φορά που θα τρέχει το πρόγραμμα από τις λέξεις: ΠΕΤΡΑ, ΨΑΛΙΔΙ, ΧΑΡΤΙ*

4.4.3 Δραστηριότητα

Δώστε τις κατάλληλες εντολές στο κέλυφος ώστε:

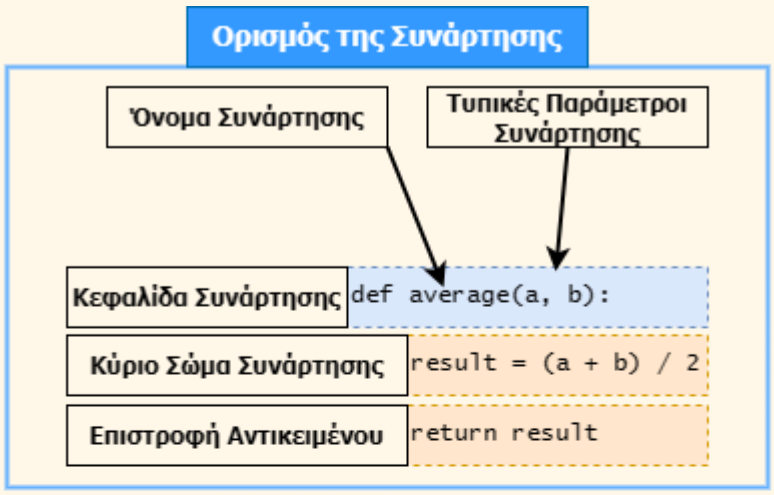
- *να εμφανίσετε τα περιεχόμενα του τρέχοντος καταλόγου*
- *να εμφανίσετε μία λίστα των αρχείων και των φακέλων του τρέχοντος καταλόγου*
- *να εμφανίσετε ένα τυχαίο αρχείο από τη λίστα των παραπάνω αρχείων*
- *να εμφανίσετε το πλήθος των αρχείων και των φακέλων του τρέχοντος καταλόγου*
- *να εμφανίσετε το πλήθος των αρχείων και των φακέλων του γονικού καταλόγου*
- *να δημιουργήστε στον τρέχοντα κατάλογο την παρακάτω δομή καταλόγων*

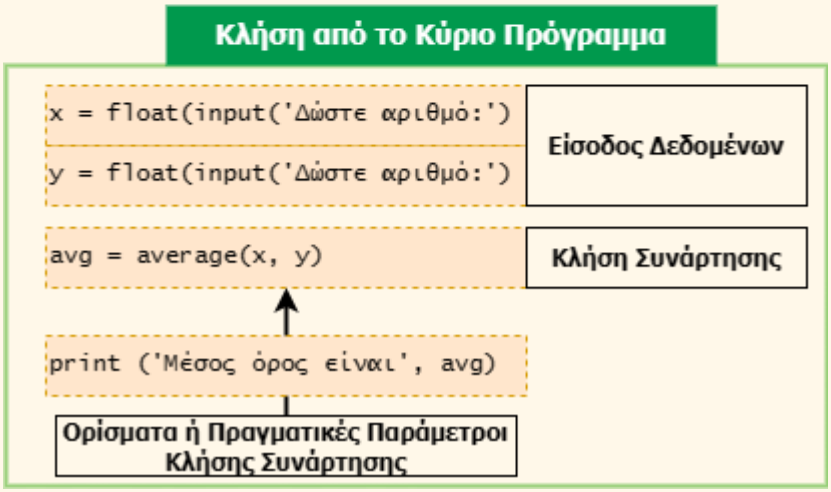


4.5 Δημιουργία, Δομή και λειτουργία συναρτήσεων χρήστη

Οι συναρτήσεις όπως αναφέρθηκε αποτελούν ένα θεμελιώδες κομμάτι της Python και επιτρέπουν τη δημιουργία επαναχρησιμοποιήσιμου κώδικα. Μια συνάρτηση είναι ένα τμήμα κώδικα που εκτελεί μια συγκεκριμένη εργασία και μπορεί να καλείται από διάφορα σημεία του προγράμματος. Σε αυτό το κεφάλαιο, θα εξεταστεί η δομή και η λειτουργία των συναρτήσεων στην Python και θα υλοποιηθούν παραδείγματα **συναρτήσεων χρήστη** για να κατανοηθεί καλύτερα η λειτουργία τους.

Μία συνάρτηση έχει δύο βασικά μέρη, την κεφαλίδα και το κύριο σώμα της. Η κεφαλίδα αρχίζει όπως αναφέρθηκε με τη δεσμευμένη εντολή(keyword) `def` (define: ορίζω) στη συνέχεια ακολουθεί το όνομα της συνάρτησης ακολουθούμενο από ένα ζεύγος παρενθέσεων μέσα στο οποίο τοποθετούνται εάν υπάρχουν οι τυπικές παράμετροι (formal parameters) της συνάρτησης διαχωρισμένες με κόμμα(,).





Κάποιες συναρτήσεις επιστρέφουν μία τιμή με την εντολή `return`, άλλες μπορεί να εκτελούν λειτουργίες χωρίς να επιστρέφουν κάποια τιμή.

Στη συνέχεια αφού οριστεί μία συνάρτηση χρήστη, αυτή πρέπει να κληθεί από το κύριο πρόγραμμα ώστε να εκτελεστούν οι λειτουργίες της. Εάν διαθέτει δε και παραμέτρους τότε κατά την κλήση πρέπει να δοθούν σε αυτές και πραγματικές τιμές.

4.5.1 Δημιουργία συναρτήσεων χωρίς επιστροφή και χωρίς παραμέτρους

Στο επόμενο παράδειγμα που ακολουθεί θα δημιουργηθεί **η πιο απλή μορφή συνάρτησης** η οποία **δεν διαθέτει παραμέτρους εισόδου** και **δεν επιστρέφει κάποια τιμή**, απλά εμφανίζει μηνύματα κειμένου στην οθόνη.

Παράδειγμα 4.27

Συντάξτε και εκτελέστε τις παρακάτω εντολές για δημιουργία συνάρτησης χρήστη χωρίς παραμέτρους

```
# Η δημιουργία μίας νέας συνάρτησης με το όνομα about
# γίνεται με την εντολή def και την ακόλουθη σύνταξη

>>> def about():
...     print('Το πρόγραμμα αυτό υλοποιήθηκε από')
...     print('    Αναστάσιο Αναστασίου')
...     print('    Χρησιμοποιήθηκε η γλώσσα')
...     print('    Python 3.11.4')
...     print('All rights reserved 2023')
...
...
```

η κλήση της συνάρτησης γίνεται καλώντας το όνομά της

```
>>> about()
Το πρόγραμμα αυτό υλοποιήθηκε από
    Αναστάσιο Αναστασίου
    Χρησιμοποιήθηκε η γλώσσα
    Python 3.11.4
All rights reserved 2023
```

Παρατηρήσετε ότι η κλήση της συνάρτησης γίνεται χρησιμοποιώντας το όνομά της ακολουθούμενο από παρενθέσεις. Η συνάρτηση αυτή δεν δέχεται κάποια παράμετρο ως είσοδο ούτε επιστρέφει κάποιο αντικείμενο (αυτό δεν είναι απόλυτα αληθές όπως θα αναφερθεί στην πορεία) απλά εμφανίζει κάποια μηνύματα στην οθόνη.

4.5.2 Δημιουργία συναρτήσεων χωρίς επιστροφή με παραμέτρους

Στο επόμενο παράδειγμα η συνάρτηση που δημιουργείται **έχει ως είσοδο μία παράμετρο** χωρίς να επιστρέφει κάποια τιμή κατά την κλήση της.

Παράδειγμα 4.28

Συντάξτε και εκτελέστε τις παρακάτω εντολές για δημιουργία συνάρτησης χρήστη με παράμετρο

```
# Τροποποίηση της συνάρτηση με όνομα about και προσθήκη παραμέτρου name
>>> def about(name):
...     print('Το πρόγραμμα αυτό υλοποιήθηκε από')
...     print(name)
```

```
...     print('    Χρησιμοποιήθηκε η γλώσσα')
...     print('    Python 3.11.4')
...     print('All rights reserved 2023')
```

Η κλήση της συνάρτησης about γίνεται από το κύριο πρόγραμμα (ή και από άλλη συνάρτηση) με χρήση του ορίσματος ονόματος

```
>>> about('Άννα Παπαγεωργίου')
Το πρόγραμμα αυτό υλοποιήθηκε από
Άννα Παπαγεωργίου
    Χρησιμοποιήθηκε η γλώσσα
    Python 3.11.4
All rights reserved 2023
```

4.5.3 Δραστηριότητα

Στο προηγούμενο παράδειγμα το όνομα του προγραμματιστή ακολουθεί αριστερή στοίχιση και όχι κεντρική. Να τροποποιηθεί η συνάρτηση ώστε ανάλογα με το μήκος του ονόματος του προγραμματιστή να εκτελεί κεντρική στοίχιση εμφάνισης της 2ης γραμμής σε σχέση με την 1η πρώτη γραμμή ώστε το αποτέλεσμα μετά την κλήση της να είναι το παρακάτω:

```
>>> about('Άννα Παπαγεωργίου')
Το πρόγραμμα αυτό υλοποιήθηκε από
       Άννα Παπαγεωργίου
    Χρησιμοποιήθηκε η γλώσσα
    Python 3.11.4
All rights reserved 2023
```

4.5.4 Δημιουργία συνάρτησης με παραμέτρους και επιστροφή τιμής

Μία συνάρτηση ορίζεται συνήθως εντός ενός σεναρίου της Python. Στη συνέχεια η συνάρτηση καλείται μέσα από το σενάριο με το όνομά της και πραγματικά δεδομένα τα οποία περνούν στις τυπικές παραμέτρους. Στη συνέχεια ξεκινά η εκτέλεση του τμήματος κώδικα που περιέχεται στη συνάρτηση. Όταν ολοκληρωθεί η εκτέλεσή του η **εντολή return που περιέχεται σε αυτή επιστρέφει τα αποτελέσματα** των υπολογισμών στο κύριο πρόγραμμα. Τα παραπάνω φαίνονται στο παράδειγμα που ακολουθεί.

Παράδειγμα 4.29

Συντάξτε και εκτελέστε τις παρακάτω εντολές για δημιουργία συνάρτησης χρήστη με παραμέτρους και επιστροφή τιμής από τη συνάρτηση

```
# Ορίζεται η συνάρτηση average η οποία δέχεται 2 αριθμούς
# Υπολογίζει το μέσο όρο τους και τον επιστρέφει
def average(a, b):          # Ορισμός συνάρτησης
    result = (a + b) / 2    # Υπολογισμός Μέσου όρου

    return result           # Επιστροφή Μέσου όρου
```

```
# ΚΥΡΙΟ ΠΡΟΓΡΑΜΜΑ
# Εισαγωγή δεδομένων από το χρήστη
x = float(input('Δώσε αριθμό:'))
y = float(input('Δώσε αριθμό:'))

# Κλήση της συνάρτησης με ορίσματα τις τιμές των x, y
# Οι τιμές των ορισμάτων περνούν στις τυπικές μεταβλητές a, b της
συνάρτησης
avg = average(x, y) # Το αποτέλεσμα των υπολογισμών της συνάρτησης
επιστρέφεται στη μεταβλητή avg

print('Ο μέσος όρος των αριθμών που δόθηκαν είναι',avg)
```

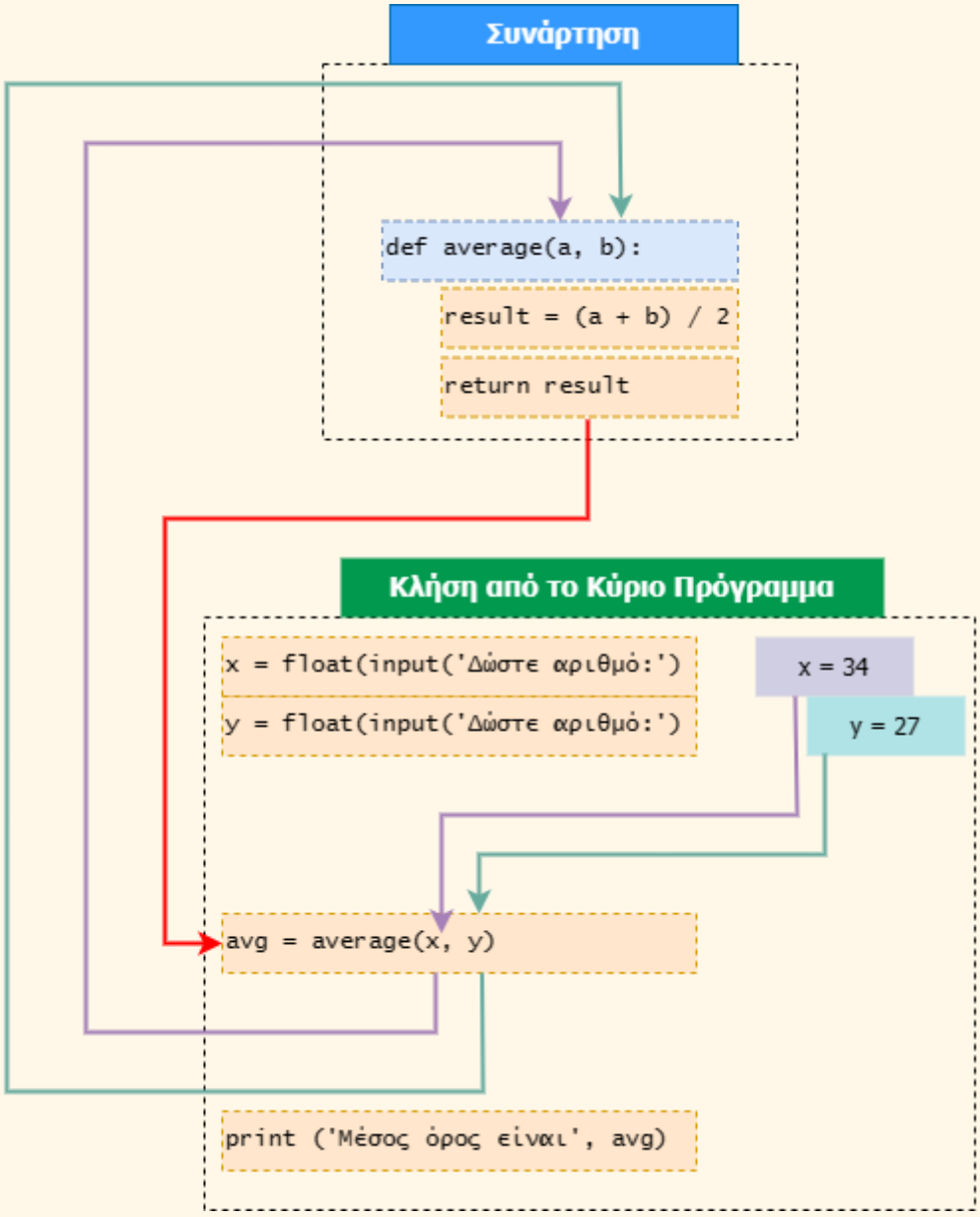
Το αποτέλεσμα στην οθόνη θα είναι

```
Δώσε αριθμό:34
Δώσε αριθμό:27

Ο μέσος όρος των αριθμών που δόθηκαν είναι 30.5
```

Σειρά εκτέλεσης εντολών στο προηγούμενο παράδειγμα κλήσης συνάρτησης σε πρόγραμμα

1. Ορίζεται η συνάρτηση average με την εντολή `def` στη μνήμη
2. Αρχίζει η εκτέλεση του κύριου προγράμματος
3. Εισάγεται το 34 στη πραγματική μεταβλητή `x` και το 27 στη πραγματική μεταβλητή `y`
4. Γίνεται η κλήση της συνάρτησης με τις τιμές των `x, y` και ο έλεγχος περνά στη συνάρτηση
5. Περνά το 34 στην τυπική παράμετρο `a` και το 27 στην τυπική παράμετρο `b`
6. Εκτελούνται οι εντολές στο σώμα της συνάρτησης με τις παραπάνω τιμές (34, 27) και υπολογίζεται η τιμή της μεταβλητής `result`
7. Ο κώδικας της συνάρτησης εκτελείται μέχρι την εντολή `return` και επιστρέφεται η τιμή της result μαζί με τον έλεγχο στο κύριο πρόγραμμα
8. Η επιστρεφόμενη τιμή της `result` τοποθετείται στη μεταβλητή `avg`
9. Γίνεται η εκτύπωση της `avg` και τελειώνει το πρόγραμμα

Κλήση συνάρτησης από το κύριο πρόγραμμα, πέρασμα παραμέτρων και επιστροφή τιμής



Σημειώσεις στις συναρτήσεις

- *Εάν δεν υπάρχει εντολή return σε μία συνάρτηση, τελειώνει η εκτέλεσή της με την τελευταία εντολή και επιστρέφεται στο κύριο πρόγραμμα ένα αντικείμενο ειδικού τύπου με τιμή* `None`.
- *Συνήθως αναφέρεται ότι η συνάρτηση επιστρέφει μία τιμή ή τιμές. Το ορθό όμως είναι ότι μία συνάρτηση στην γλώσσα Python επιστρέφει ΠΑΝΤΑ ένα αντικείμενο κάποιου τύπου πχ,* `int`, `float`, `str`, `bool`, `list`, `tuple`, `dict`, `set`, `None` *κα. Γενικά ο όρος τιμή που χρησιμοποιείται σε άλλες γλώσσες στην Python όπως θα αναλυθεί και σε επόμενα κεφάλαια είναι ταυτόσημος με τον όρο αντικείμενο.*
- *Όταν υπάρχει η εντολή* `return` *σε μία συνάρτηση, γίνεται αποτίμηση της έκφρασης που ακολουθεί την εντολή return και γίνεται επιστροφή του αντικειμένου εξόδου που παράγεται από την έκφραση.*
- *Η εντολή* `return` *σταματά την εκτέλεση του σώματος της συνάρτησης, οπότε εάν υπάρχει εντολή μετά από αυτήν απλά δεν εκτελείται.*

- *Εάν μία συνάρτηση κληθεί με πραγματικές παραμέτρους εκφράσεις, γίνεται πρώτα η αποτίμηση των εκφράσεων, υπολογίζεται η τιμή τους και περνά αυτή η τιμή (αντικείμενο) στην τυπική παράμετρο της συνάρτησης.*
- *Οι συναρτήσεις που δημιουργούνται από το χρήστη είναι σκόπιμο να συνοδεύονται από βοήθεια σχετική με τη λειτουργία και τη χρήση τους. Αυτό γίνεται τοποθετώντας ένα multiline string κάτω από την κεφαλίδα τους το οποίο αποτελεί μέρος της τεκμηρίωσης της συνάρτησης όπως φαίνεται και στο επόμενο παράδειγμα.*

4.5.5 Κλήση συνάρτησης με ορίσματα εκφράσεις ή συναρτήσεις

Όπως επισημάνθηκε στις παραπάνω σημειώσεις:

- Εάν μία συνάρτηση κληθεί με ορίσματα (πραγματικές παραμέτρους) εκφράσεις, γίνεται πρώτα η αποτίμηση των εκφράσεων, υπολογίζεται η τιμή τους και περνά αυτή η τιμή (αντικείμενο) στην τυπική παράμετρο της συνάρτησης.

Παράδειγμα 4.30

```
# Ορίζεται η συνάρτηση add η οποία δέχεται 2 αριθμούς
# Υπολογίζει το άθροισμά τους και το επιστρέφει
def add(a, b):
    """Επιστρέφει το άθροισμα των παραμέτρων
    a και b"""
    result = a + b
    return result

# Κλήση της συνάρτησης με ορίσματα ακέραιες τιμές
r1 = add(12, 13)
# Κλήση της συνάρτησης με ορίσματα αριθμητικές εκφράσεις
# Αποτιμώνται πρώτα οι εκφράσεις
# και στη συνέχεια περνούν οι τιμές στις τυπικές παραμέτρους της συνάρτησης
r2 = add(30 / 5, 2 ** 3)
print(r1, r2)
```

```
25 14.0
```

- Εάν μία συνάρτηση κληθεί με ορίσματα άλλες συναρτήσεις, γίνεται πρώτα η αποτίμηση των συναρτήσεων αυτών, υπολογίζεται η τιμή τους και περνά αυτή η τιμή (αντικείμενο) στην τυπική παράμετρο της αρχικής συνάρτησης.

4.5.6 Δραστηριότητα

Χρησιμοποιώντας τη συνάρτηση `add()` *του προηγούμενου παραδείγματος χωρίς να χρησιμοποιήσετε τον τελεστή της πρόσθεσης* `(+)` *να συμπληρώσετε τα κενά στις παρακάτω εντολές ώστε να βρείτε το άθροισμα των αριθμών*:

α. 1 + 2 + 3

β. 4 + 5 + 6 + 7
γ. 1 + 3 + 5 + 7 + 9

```
>>> add(add(1, 2), _)
6
>>> add(add(4, _), __(_, _))
22
>>> ___(add(1, _), ___(5, ___(_,_)))
25
```

4.5.7 Δραστηριότητα

Συντάξτε και εκτελέστε για να υπολογίσετε με τη χρήση συνάρτησης το μέσο όρο 2 αριθμών

```
def average(a, b):
    """Επιστρέφει το Μέσο Όρο των a και b"""
    avg = (a + b) / 2
    return avg

final = average(10, 14)
print(final)
12.0
```

Στη συνέχεια:

Συμπληρώστε τα κενά ώστε χρησιμοποιώντας και θεωρώντας γνωστή τη συνάρτηση `average()` *του προηγούμενου παραδείγματος ώστε να βρείτε τους μέσους όρους που ζητούνται στα τέσσερα επόμενα τμήματα κώδικα:*

```
# Ζητείται ο μέσος όρων των αριθμών 10, 11, 12, 13
avg1 = average(__, __)
avg2 = average(__, __)
avg = _______(____, ____)
print('Ο μέσος όρος που ζητήθηκε είναι ο', avg)
```

```
# Ζητείται ο μέσος όρων των αριθμών 10, 11, 12, 13
avg = ________(average(10, 11), average(__, __))
print('Ο μέσος όρος που ζητήθηκε είναι ο', avg)
```

Δώστε τις κατάλληλες εντολές ώστε να:

```
# Βρείτε το μέσο όρο των αριθμών 10, 11, 12
```

```
# Βρείτε το μέσο όρο των αριθμών 10, 11, 12, 13, 14, 15
```

4.5.8 Δραστηριότητα

Συμπληρώστε τον παρακάτω κώδικα ώστε να δημιουργήσετε μία συνάρτηση με όνομα `calculate()` *η οποία θα δέχεται 2 αριθμούς και θα εμφανίζει τα αποτελέσματα των 4 βασικών αριθμητικών πράξεων μεταξύ τους.*

```
def ____________(a, b):
    x = a + _
    y = _ - b
    z = _ * _
    w = _ _ _
    print('Το άθροισμα', a, '+', b, '=', _)
    print('Η διαφορά',_ , '-', _, '=', _)
    print('Το γινόμενο',_ ,'_' ,_, '_', _)
    print('Το πηλίκο', _, '_', _, '_', _)
```

Στη συνέχεια καλέστε τη συνάρτηση με ορίσματα τα ζεύγη των αριθμών

- 2, 5
- 8, 6
- 4, 14 / 2 - 7

Γράψτε τις παρατηρήσεις σας για τα αποτελέσματα της κλήσης της συνάρτησης με το 3ο ζεύγος αριθμών.

4.5.9 Κλήση συναρτήσεων με προεπιλεγμένες παραμέτρους

Στη γλώσσα προγραμματισμού Python, μπορούν να οριστούν συναρτήσεις με προεπιλεγμένες παραμέτρους. Αυτό σημαίνει ότι μπορεί να οριστεί μια τιμή για μια παράμετρο, η οποία θα χρησιμοποιηθεί αν δεν δοθεί τιμή κατά την κλήση της συνάρτησης. Αυτή η δυνατότητα επιτρέπει την ευελιξία στη χρήση της συνάρτησης και την απλοποίηση της κλήσης της με λιγότερες πραγματικές παραμέτρους.

Παράδειγμα 4.31

Στο παρακάτω παράδειγμα υλοποιήστε μία συνάρτηση η οποία θα επιστρέφει την τιμή ρίψης ενός ζαριού με όρια που θα καθορίζονται από τον χρήστη. Εάν δε δοθούν όρια θα χρησιμοποιούνται τα τυπικά όρια ενός ζαριού [1, 6].

```
def dice(start=1, end=6):
    from random import randint
    return randint(start, end)
```

```
# Κλήση συνάρτησης χωρίς πραγματικές παραμέτρους
# χρησιμοποιούνται οι προεπιλεγμένες τιμές 1, 6
dice()
3
```

```
# Κλήση συνάρτησης με πραγματικές παραμέτρους 1, 10
dice(1, 10)
8
```

Είναι εφικτό στην κεφαλίδα της συνάρτησης να συνδυάζονται τυπικές παράμετροι με προεπιλεγμένες τιμές με τυπικές παραμέτρους χωρίς προεπιλεγμένες τιμές. Όμως από την ώρα που χρησιμοποιηθεί μία τυπική παράμετρος με προεπιλεγμένη τιμή δεν μπορεί να τοποθετηθεί στη συνέχεια τυπική παράμετρος χωρίς προεπιλεγμένη τιμή. Όπως φαίνεται στο παρακάτω παράδειγμα:

Παράδειγμα 4.32

Αποδεκτοί τρόποι χρήσης τυπικών παραμέτρων με προεπιλεγμένες τιμές:

```
def func1(a=3, b=4):
    return a + b

def func2(a, b=4):
    return a + b
```

ΜΗ αποδεκτός τρόπος χρήσης τυπικών παραμέτρων με προεπιλεγμένες τιμές:

```
def func3(a=3, b):
    return a + b
SyntaxError: non-default argument follows default argument
```

4.5.10 Κλήση συνάρτησης από πρόγραμμα ή από άλλη συνάρτηση

Σε προηγούμενη παράγραφο χρησιμοποιήθηκαν σε παραδείγματα κλήσεων συναρτήσεων είτε από το κύριο πρόγραμμα είτε από τις πραγματικές παραμέτρους μίας συνάρτησης. Είναι εφικτό όπως αναφέρθηκε κατά την υλοποίηση ενός προγράμματος ή μίας συνάρτησης χρήστη να γίνει κλήση μίας συνάρτησης από μία άλλη συνάρτηση καθώς και μία άλλη κλήση συνάρτησης μέσα από αυτήν κοκ.

Παράδειγμα 4.33

Φανταστείτε ότι ένα Σάββατο πρωί αποφασίσατε να τακτοποιήσετε και να καθαρίσετε το δωμάτιο σας. Πριν τελειώσετε την εργασία αυτή ένα μέλος της οικογένειάς σας καλεί να το βοηθήσετε να μεταφέρει τα ψώνια από το αυτοκίνητο γιατί έχει χαλάσει ο ανελκυστήρας. Σταματάτε λοιπόν την τακτοποίηση του δωματίου σας για να βοηθήσετε στη μεταφορά. Την ώρα όμως που βοηθάτε στη μεταφορά χτυπάει το κινητό σας τηλέφωνο με κλήση από την εργασία σας. Σταματάτε προσωρινά τη μεταφορά και απαντάτε στην κλήση. Όμως κατά τη διάρκεια της ομιλίας με την εργασία σας ακούτε να χτυπάει ο συναγερμός φωτιάς της οικίας σας. Αφήνετε το τηλέφωνο χωρίς να τερματίσετε την κλήση και τρέχετε να δείτε τι συμβαίνει. Βλέπετε ότι στην κουζίνα είχε ξεχαστεί ένα σκεύος με ανοιχτό το μάτι και βγάζει καπνούς χωρίς ευτυχώς να πάρει φωτιά. Τακτοποιείτε το θέμα αυτό και κάθεστε σε μία πολυθρόνα να ηρεμήσετε από την ταραχή. Αφού ηρεμήσετε, σκέφτεστε τώρα ότι έχετε αφήσει πολλές εκκρεμότητες στη μέση και πρέπει να τις διεκπεραιώσετε. Δυσκολεύεστε όμως να θυμηθείτε όχι μόνο ποιες είναι αυτές οι εκκρεμότητες αυτές αλλά και με ποια σειρά πρέπει να τις κάνετε.

Αυτές οι καταστάσεις που περιεγράφηκαν παραπάνω και είναι ίσως λίγο απίθανο αλλά όχι αδύνατον να συμβούν στην πραγματικότητα **είναι πολύ συχνό φαινόμενο στον τμηματικό προγραμματισμό** και βέβαια με πολύ μεγαλύτερη πολυπλοκότητα.

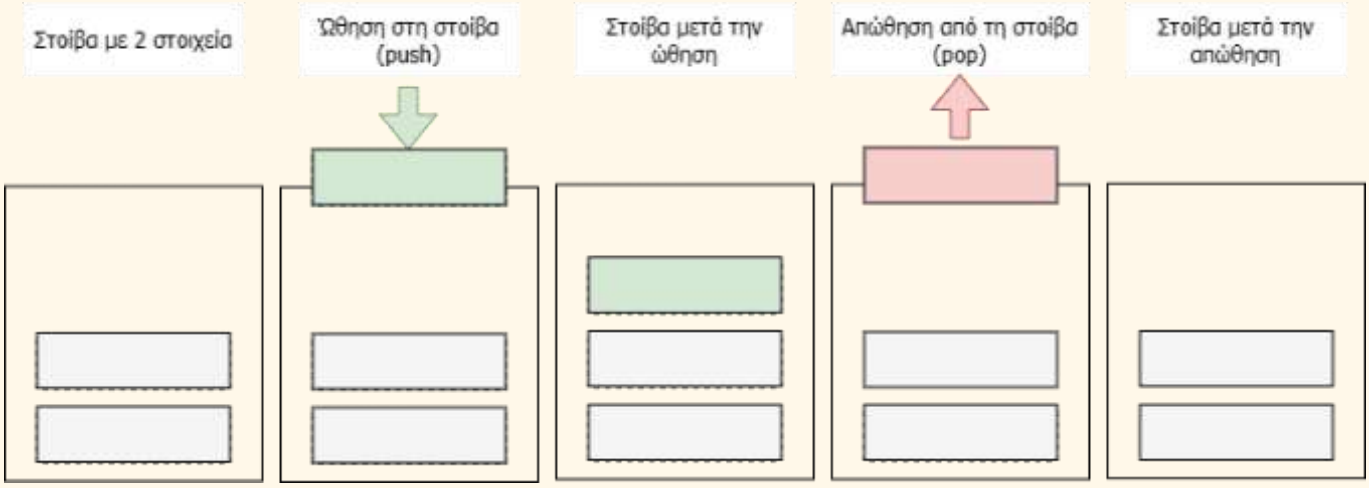
Έτσι λοιπόν ένα ερώτημα το οποίο τίθεται είναι πως διατηρείται ο έλεγχος της σειράς των εντολών που πρέπει να εκτελεστούν, εφόσον κάθε φορά που καλείται μία συνάρτηση από κάποιο σημείο ενός προγράμματος ή από μία άλλη συνάρτηση ο έλεγχος εκτέλεσης της σειράς εντολών του προγράμματος, διακόπτεται και μεταφέρεται στη συνάρτηση. Μετά το πέρας της συνάρτησης ο έλεγχος εκτέλεσης μεταφέρεται και πάλι πίσω ακριβώς μετά την εντολή που κλήθηκε αρχικά η συνάρτηση.

Η λειτουργία αυτή μπορεί να συμβαίνει με μεγαλύτερη πολυπλοκότητα πχ ένα πρόγραμμα στο σώμα του να καλεί μία συνάρτηση αυτή η συνάρτηση από το σώμα της να καλεί μία άλλη συνάρτηση κοκ.

Να σημειωθεί ότι η λειτουργία αυτή αν και γίνεται αυτοματοποιημένα από τους μηχανισμούς της γλώσσας, πρέπει να έχει κατανοηθεί ο μηχανισμός της από τον προγραμματιστή ώστε να μπορεί να παρακολουθεί τη ροή εκτέλεσης των εντολών στα προγράμματα που αναπτύσσει ή συντηρεί.

Για να διατηρείται στον προγραμματισμό η σειρά ελέγχου εκτέλεσης των εντολών ακολουθείται η τακτική breadcrumb navigation (πλοήγηση με ψίχουλα ψωμιού εμπνευσμένο από το παιδικό παραμύθι Χάνσελ και Γκρέτελ).

Στον προγραμματισμό για υλοποιηθεί η παραπάνω λειτουργία χρησιμοποιείται μία **δομή δεδομένων που ονομάζεται στοίβα** (stack) (η στοίβα θα μελετηθεί ως δομή δεδομένων και κλάση αργότερα). Η στοίβα αυτή μοιάζει σαν ένα κατακόρυφο δοχείο το οποίο έχει άνοιγμα μόνο στο επάνω μέρος του (στην κορυφή του). Ότι στοιχείο τοποθετηθεί ή ότι στοιχείο ληφθεί από τη στοίβα μπορεί να γίνει μόνο από το άνοιγμα αυτό. Η τοποθέτηση ενός στοιχείου στην κορυφή της στοίβας ονομάζεται πιο τυπικά ώθηση (push), ενώ η λήψη ενός στοιχείου από την κορυφή της στοίβας ονομάζεται απώθηση (pop).

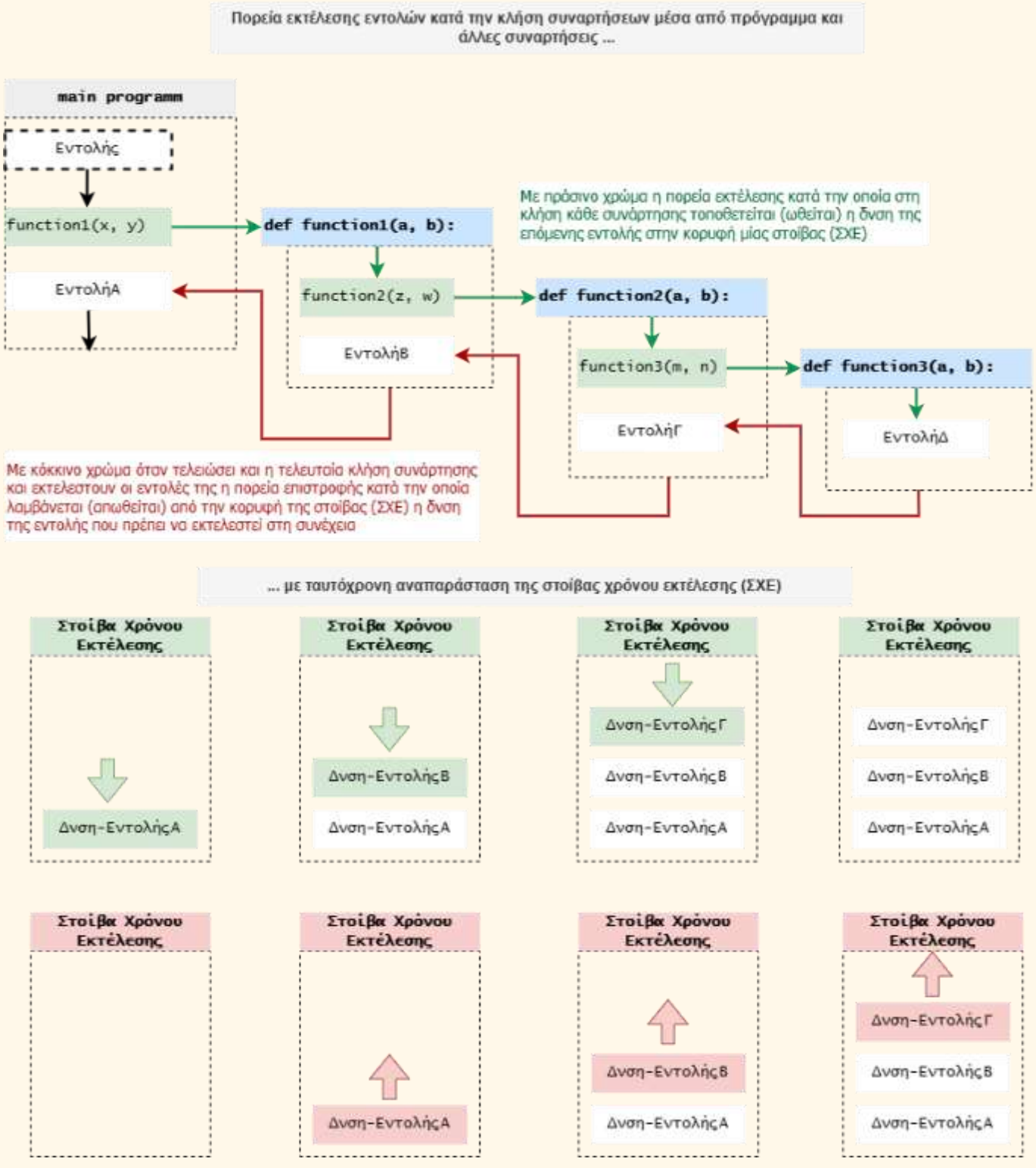


Κάθε φορά που καλείται μία συνάρτηση ανεξάρτητα από που καλείται αυτή (από το πρόγραμμα ή από άλλη συνάρτηση) διακόπτεται η εκτέλεση του προγράμματος και η δνση μνήμης της επόμενης εντολής ωθείται σε μία στοίβα (ρίχνουμε δηλαδή ένα ψίχουλο: breadcrumb) η οποία ονομάζεται Στοίβα Χρόνου Εκτέλεσης (ΣΧΕ) αφήνοντας τις επόμενες εντολές του προγράμματος σε αναμονή.

Εάν τώρα αυτή η συνάρτηση καλεί και μία άλλη συνάρτηση διακόπτεται η εκτέλεση των δικών της εντολών και ωθείται στη ΣΧΕ η δνση της επόμενης εντολής της και ο έλεγχος μεταφέρεται στην 2η συνάρτηση κοκ.

Όταν τελειώσει η εκτέλεση όλων των εντολών της τελευταίας συνάρτησης (όταν σβήσουμε το μάτι της κουζίνας στο αρχικό παράδειγμα) απωθείται από τη ΣΧΕ η δνση μνήμης που βρίσκεται στην κορυφή της και συνεχίζει η εκτέλεση των εντολών της συνάρτησης από εκεί.

Όταν τελειώσουν οι εντολές της συνάρτησης αυτής απωθείται η επόμενη δνση από τη στοίβα και συνεχίζεται η εκτέλεση από εκεί κοκ. (Η αναπαράσταση της λειτουργίας των πολλαπλών κλήσεων για 3 συναρτήσεις με την κατάσταση της αντίστοιχης ΣΧΕ φαίνεται και στο παρακάτω σχήμα)


Πορεία εκτέλεσης εντολών κατά την κλήση συναρτήσεων μέσα από πρόγραμμα και άλλες συναρτήσεις ...
main programm
Εντολής
function1(x, y)
def function1(a, b):
ΕντολήΑ
function2(z, w)
def function2(a, b):
ΕντολήΒ
function3(m, n)
def function3(a, b):
ΕντολήΓ
ΕντολήΔ
Με πράσινο χρώμα η πορεία εκτέλεσης κατά την οποία στη κλήση κάθε συνάρτησης τοποθετείται (ωθείται) η δνση της επόμενης εντολής στην κορυφή μίας στοίβας (ΣΧΕ)
Με κόκκινο χρώμα όταν τελειώσει και η τελευταία κλήση συνάρτησης και εκτελεστουν οι εντολές της η πορεία επιστροφής κατά την οποία λαμβάνεται (απωθείται) από την κορυφή της στοίβας (ΣΧΕ) η δνση της εντολής που πρέπει να εκτελεστεί στη συνέχεια
... με ταυτόχρονη αναπαράσταση της στοίβας χρόνου εκτέλεσης (ΣΧΕ)
Στοίβα Χρόνου Εκτέλεσης
Δνση-ΕντολήςΑ
Δνση-ΕντολήςΒ
Δνση-ΕντολήςΓ

4.6 Δημιουργία και φόρτωση αρθρώματος λογισμικού του χρήστη

Εκτός από τα ενσωματωμένα αρθρώματα λογισμικού (built-in modules) και τις ενσωματωμένες συναρτήσεις της γλώσσας (built-in functions), όπως υπάρχει η δυνατότητα δημιουργίας συναρτήσεων χρήστη (user defined functions) υπάρχει και η **δυνατότητα δημιουργίας αρθρωμάτων χρήστη** (user defined modules).

Η δημιουργία αρθρωμάτων από το χρήστη παρουσιάζει χρησιμότητα:

- είτε σε περιπτώσεις επαναχρησιμοποίησης ολόκληρων λειτουργιών και τμημάτων του κώδικα σε άλλα προγράμματα ώστε να μειώνεται ο χρόνος συγγραφής τους και να απλοποιείται ο κώδικάς τους,
- είτε σε περιπτώσεις σχεδίασης μέσω του τμηματικού προγραμματισμού όπου ο κώδικας μπορεί να διαχωρίζεται ομαδοποιώντας τις συναφείς λειτουργίες του σε μικρότερα αρχεία που θα καλούνται τελικά από κάποιο κύριο αρχείο διευκολύνοντας έτσι τον καταμερισμό εργασιών, τη συντήρηση και τη διόρθωσή τους.

Τα αρθρώματα λογισμικού δεν είναι τίποτα άλλο από αρχεία κώδικα σε γλώσσα Python με επέκταση .py το όνομα των οποίων πρέπει να αρχίζει από πεζό λατινικό γράμμα. Συνήθως περιέχουν συναρτήσεις, μεταβλητές (αντικείμενα διαφόρων τύπων), ή και αργότερα όπως θα συναντήσουμε σε επόμενο κεφάλαιο κλάσεις.

- Η φόρτωση των αρθρωμάτων του χρήστη γίνεται με παρόμοιους τρόπους όπως και των ενσωματωμένων αρθρωμάτων της γλώσσας

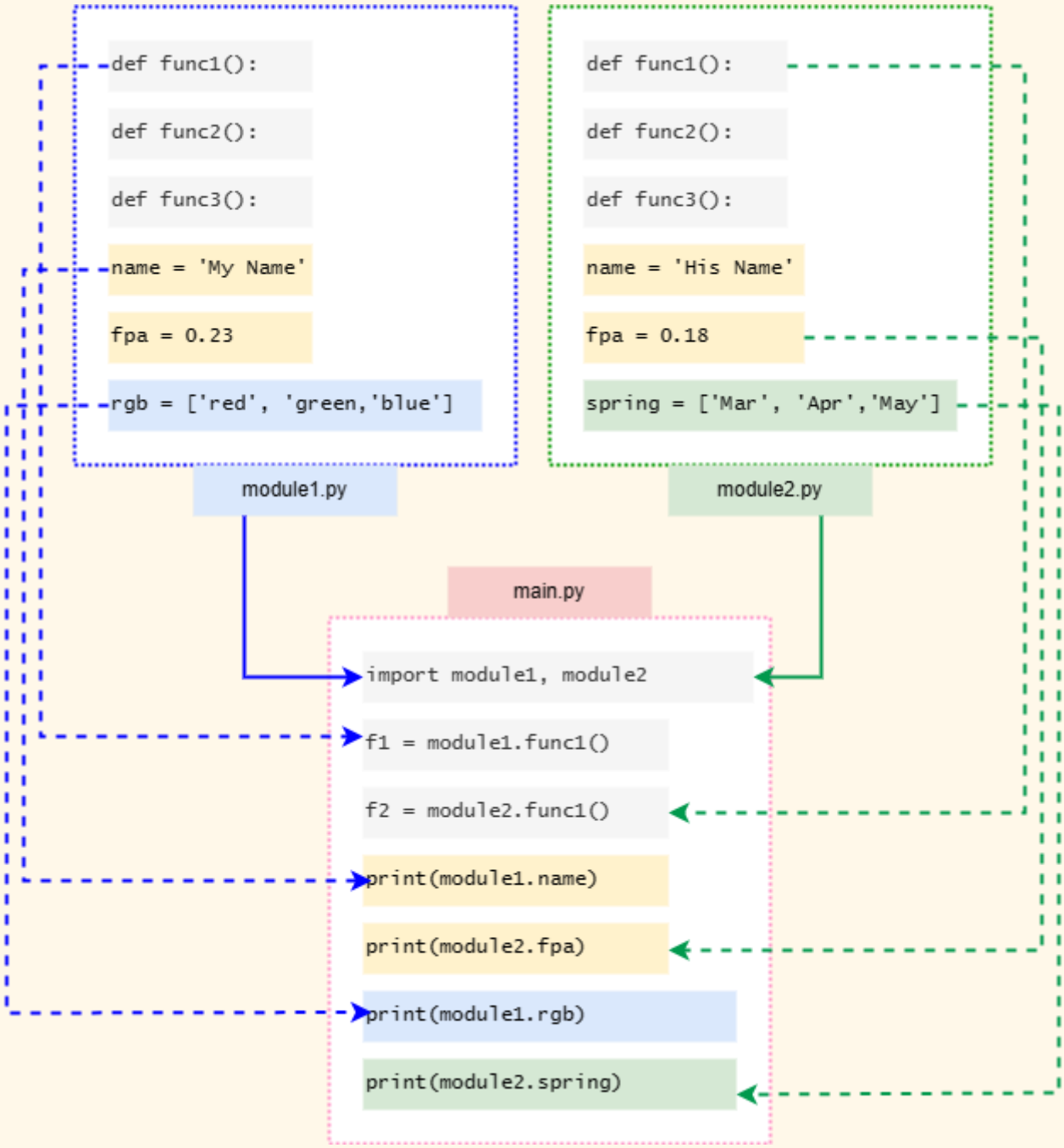
```
1. import module_name
2. from module_name import function1, function2
3. from module_name import *
4. from module_name import my_module as mm
```

- Η φόρτωση ενός αρθρώματος που περιέχει κάποιες μεμονωμένες εντολές προκαλεί αυτόματα και εκτέλεσή τους.
- Σε περιπτώσεις όμως κοινής ονομασίας συναρτήσεων ή μεταβλητών εντός των αρθρωμάτων πρέπει να χρησιμοποιείται ο 1ος τρόπος κλήσης όπως φαίνεται στο παρακάτω σχήμα ώστε να μην υπάρχει σύγχυση από πιο άρθρωμα καλείται ένα αντικείμενο ή μία συνάρτηση με κοινό όνομα.
- Τα αρθρώματα είναι σκόπιμο για ευκολία να αποθηκεύονται σε κοινή τοποθεσία (φάκελο) με το κύριο πρόγραμμα, σε διαφορετική περίπτωση χρειάζεται να χρησιμοποιηθεί πρώτα αλλαγή του τρέχοντος φακέλου στο φάκελο που περιέχει το άρθρωμα:
  - `os.chdir('c:\\python311\\my_scripts\'`
  - ή χρησιμοποιώντας άλλες τεχνικές με το άρθρωμα sys για τις οποίες όμως δεν θα επεκταθούμε περαιτέρω

Ένα αρχείο .py μπορεί να περιέχει διάφορες εντολές, συναρτήσεις που εκτελούν κάποιες λειτουργίες ή κλάσεις ή μεταβλητές και μπορεί είτε να:

- εκτελεστεί ως σενάριο (script) είτε να,
- φορτωθεί ως άρθρωμα (module)

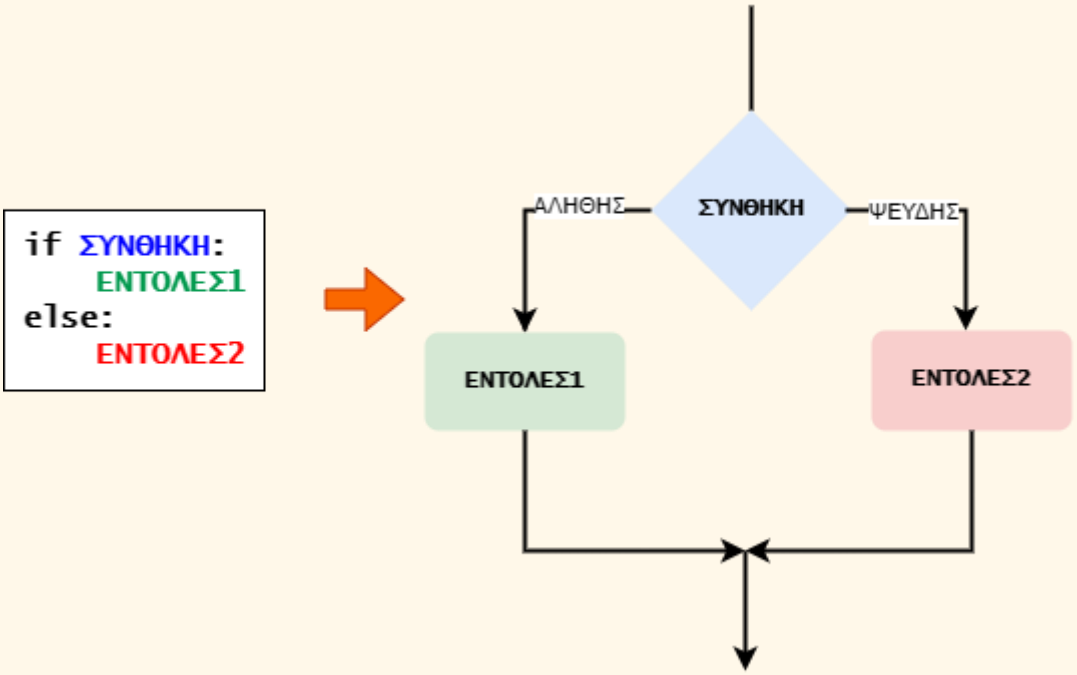
Φόρτωση αρθρωμάτων μέσα στο κύριο πρόγραμμα



Σε περίπτωση που ο προγραμματιστής επιθυμεί να χρησιμοποιήσει κάποιες συναρτήσεις ή μεταβλητές που περιέχει το αρχείο αυτό και σε άλλα προγράμματα (χωρίς να τις ξαναγράψει) χρειάζεται να φορτώσει το αρχείο αυτό μέσα στο νέο πρόγραμμά του. Για να μην εκτελεστούν όμως οι μεμονωμένες εντολές που περιέχει κατά τη φόρτωση αφού αφορούν στο αρχικό πρόγραμμα και όχι στο νέο χρειάζεται η εντολή `if __name__ == '__main__'`. Η λειτουργία της θα αναλυθεί παρακάτω.

Στο παράδειγμα που ακολουθεί θα χρησιμοποιηθεί μία νέα εντολή `if - else` που θα μελετηθεί αναλυτικά σε επόμενο μάθημα εδώ θα εξηγηθεί συνοπτικά η λειτουργία της:

Η εντολή `if` ελέγχει τη ΣΥΝΘΗΚΗ που ακολουθεί, σε περίπτωση που αυτή βρεθεί αληθής εκτελούνται οι ΕΝΤΟΛΕΣ1 που περιέχει, σε περίπτωση που η ΣΥΝΘΗΚΗ βρεθεί ψευδής εκτελούνται οι ΕΝΤΟΛΕΣ2 που περιέχονται στην εντολή `else`.



Έχει αναφερθεί ότι όταν φορτώνεται ένα module με την εντολή `import` οι μεμονωμένες εντολές που περιέχει εκτελούνται ως εάν είχε επιλεχθεί από το menu του editor η εντολή run.

Παράδειγμα 4.34

Το παρακάτω πρόγραμμα περιέχει 3 συναρτήσεις υπολογισμού μέσου όρου 2, 3 και 4 αριθμών αντίστοιχα, 1 συνάρτηση `test_data()` με δοκιμαστικά δεδομένα και τη νέα εντολή:

```
if __name__ == '__main__'
```

Αποθηκεύστε τον παρακάτω κώδικα σε ένα αρχείο με όνομα demo_module.py

```
"""Άρθρωμα λογισμικού επίδειξης λειτουργιών"""


def avg2(a, b):
    """Υπολογισμός ΜΟ 2 αριθμών"""
    return (a + b) / 2


def avg3(a, b, c):
    """Υπολογισμός ΜΟ 3 αριθμών"""
    return (a + b + c) / 3


def avg4(a, b, c, d):
    """Υπολογισμός ΜΟ 4 αριθμών"""
    return (a + b + c + d) / 4
```

```
def test_data():
    """ Δοκιμαστικά δεδομένα για έλεγχο καλής λειτουργίας
    των συναρτήσεων avg2, avg3, avg4"""
    x = 3
    y = 4
    z = 5
    w = 6
    print('Average of', x, y, 'is', avg2(x, y))
    print('Average of', x, y, z, 'is', avg3(x, y, z))
    print('Average of', x, y, z, w, 'is', avg4(x, y, z, w))


if __name__ == '__main__':
    test_data()
else:
    print('Το άρθρωμα', __name__, 'φορτώθηκε με επιτυχία')
```

Κάντε τώρα τις παρακάτω ενέργειες:

- Εκτελέστε το αρχείο με τη γνωστή επιλογή RUN από το μενού επιλογών του IDLE. Θα εμφανιστούν τα ακόλουθα αποτελέσματα από όπου είναι εμφανές ότι έγινε κλήση της συνάρτησης `test_data()` άρα η ΣΥΝΘΗΚΗ της εντολής `if` ήταν αληθής. Αυτό έγινε διότι η μεταβλητή `__name__` πήρε ως τιμή το όνομα του τρέχοντος προγράμματος, έτσι αντιλαμβάνεται η Python ότι το αρχείο .py έγινε RUN και όχι `import`.

```
Average of 3 4 is 3.5
Average of 3 4 5 is 4.0
Average of 3 4 5 6 is 4.5
```

- Έπειτα επιλέξτε Shell / Restart Shell από το μενού επιλογών του κελύφους και στη συνέχεια εφόσον ο τρέχον φάκελος είναι αυτός του αρθρώματος (εάν δεν είναι χρησιμοποιήστε το άρθρωμα και τη λειτουργία `os.chdir`) φορτώστε το άρθρωμα λογισμικού με την εντολή `import demo_module`. Από τα αποτελέσματα που ακολουθούν φαίνεται ότι δεν κλήθηκε η συνάρτηση `test_data()` συνεπώς η ΣΥΝΘΗΚΗ της εντολής `if` ήταν ψευδής και απλά φορτώθηκαν οι συναρτήσεις του αρθρώματος εμφανίζοντας και το σχετικό μήνυμα.

```
# Προαιρετικό τμήμα
import os
os.chdir('c:\\python311\\my_scripts\\diek\\kef_04')
# Φόρτωση αρθρώματος χρήστη
import demo_module
```

```
Το άρθρωμα demo_module φορτώθηκε με επιτυχία
```

- Δοκιμάστε τώρα και το αποτέλεσμα της συνάρτησης `help()` στο άρθρωμα που υλοποιήσατε.

```
help(demo_module)
Help on module demo_module:

NAME
    demo_module - Άρθρωμα λογισμικού επίδειξης λειτουργιών

FUNCTIONS
    avg2(a, b)
        Υπολογισμός ΜΟ 2 αριθμών

    avg3(a, b, c)
        Υπολογισμός ΜΟ 3 αριθμών

    avg4(a, b, c, d)
        Υπολογισμός ΜΟ 4 αριθμών

    test_data()
        Δοκιμαστικά δεδομένα για έλεγχο καλής λειτουργίας
        των συναρτήσεων avg2, avg3, avg4

FILE
    c:\python311\my_scripts\diek\kef_04\demo_module.py
```

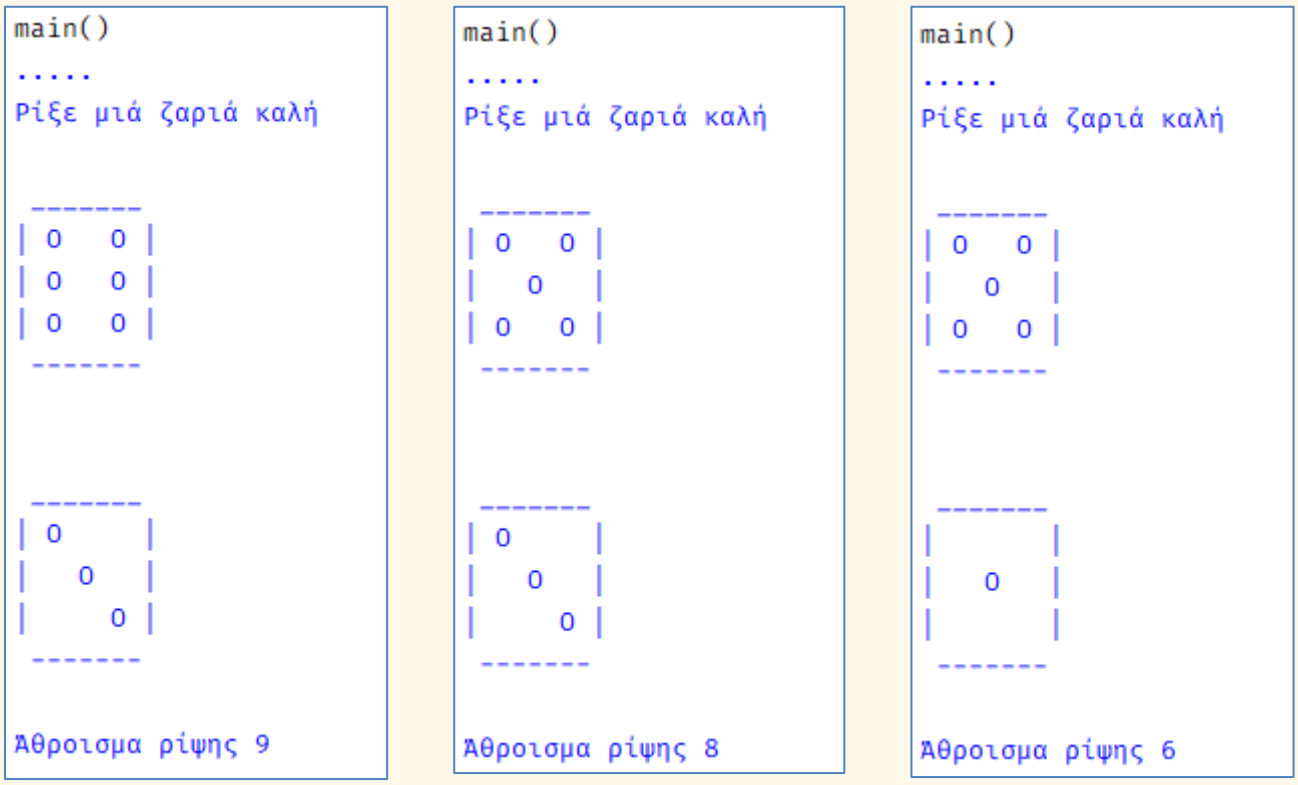
Συμπερασματικά

- Η εντολή `if __name__ == '__main__'` είναι ένας τρόπος να ελεγχθεί αν το αρχείο Python εκτελείται ως κύριο πρόγραμμα ή ως άρθρωμα (module) που εισάγεται από άλλο πρόγραμμα. Αυτό επιτρέπει να ορίζεται κώδικας που θα εκτελεστεί μόνο όταν το αρχείο εκτελείται ως κύριο πρόγραμμα και να αποφευχθεί η εκτέλεση μη επιθυμητού κώδικα όταν το αρχείο εισάγεται ως μονάδα σε άλλα προγράμματα.
- Τα αρθρώματα λογισμικού είναι επιβεβλημένο να περιέχουν πλήρη τεκμηρίωση αφού μπορεί να χρησιμοποιηθούν σε πολλά προγράμματα από διάφορους χρήστες.

Παράδειγμα 4.35

Στο επόμενο παράδειγμα καλείστε να υλοποιήσετε ένα ηλεκτρονικό διπλό ζάρι με «γραφική απεικόνιση» των αποτελεσμάτων της ρίψης. Έτσι κάθε φορά που θα καλείται το πρόγραμμα ζάρι:

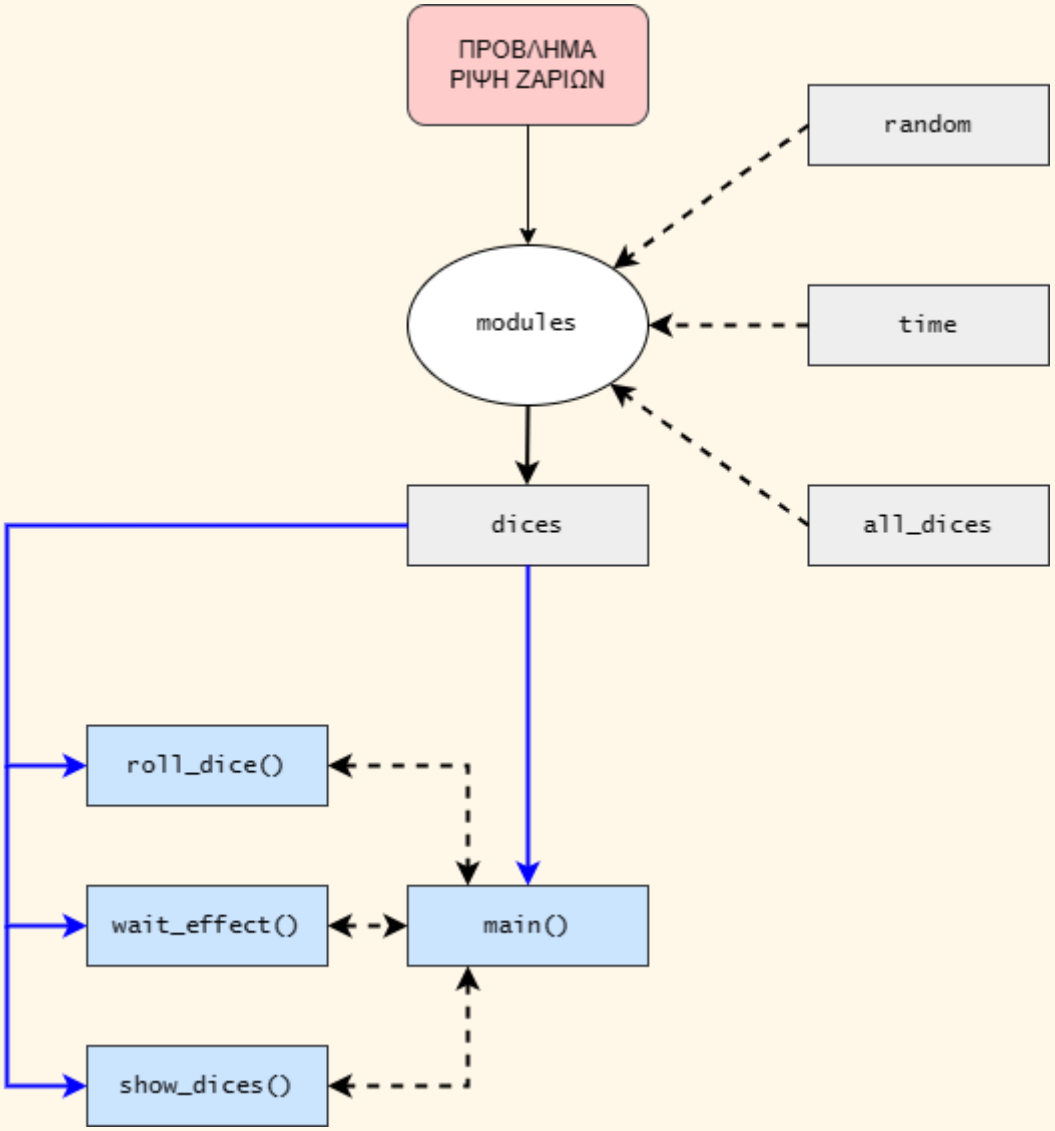
- *θα υπάρχει ένα εφέ καθυστέρησης του αποτελέσματος (για σασπένς) και στη συνέχεια*
- *θα εμφανίζεται το αποτέλεσμα της κάθε ρίψης ανάλογο με τις ακόλουθες εικόνες:*



Βήματα επίλυσης του προβλήματος

1. Δημιουργία των εικόνων των ζαριών με συμβολικούς χαρακτήρες (ascii art) μία εικόνα για κάθε ρίψη σε μορφή multiline string και αποθήκευσή τους σε ξεχωριστό αρχείο (.py άρθρωμα) με όνομα all_dices.py, όπως φαίνεται παρακάτω.
2. Εισαγωγή των ενσωματωμένων αρθρωμάτων λογισμικού της Python: `random, time`
3. Υλοποίηση των ακόλουθων συναρτήσεων στο αρχείο προγράμματος dices.py:
   α. `roll_dice()`, η οποία θα επιστρέφει ένα τυχαίο ακέραιο στο διάστημα [1, 6]
   β. `wait_effect()`, η οποία θα δημιουργεί το εφέ καθυστέρησης εμφανίζοντας μία τελεία(.) κάθε 200 msecs
   γ. `show_dices(d1, d2)`, η οποία θα δέχεται 2 ακέραιες τιμές στο διάστημα [1, 6] και εμφανίζει τις αντίστοιχες εικόνες από το άρθρωμα all_dices
   δ. `main()`, η οποία θα καλεί κατάλληλα τις παραπάνω συναρτήσεις και θα εμφανίζει τα αποτελέσματα στην οθόνη

Δομή υλοποίησης της εφαρμογής των ηλεκτρονικών ζαριών με χρήση συναρτήσεων, ενσωματωμένων αρθρωμάτων και αρθρωμάτων χρήστη



περιεχόμενα αρχείου all_dices.py

```
dice6 = """
 -------
| O   O |
| O   O |
| O   O |
 -------
 """

dice5 = """
 -------
| O   O |
|   O   |
| O   O |
 -------
 """
```

```
dice4 = """
 -------
| O   O |
|       |
| O   O |
 -------
 """

dice3 = """
 -------
| O     |
|   O   |
|     O |
 -------
 """
dice2 = """
 -------
| O     |
|       |
|     O |
 -------
 """
dice1 = """
 -------
|       |
|   O   |
|       |
 -------
 """
```

Δημιουργία του βασικού αρχείου του κώδικα του προγράμματος με όνομα dices.py

```
# Φόρτωση όλων των απαραίτητων αρθρωμάτων
from random import randint  # Λειτουργία τυχαίων ακεραίων
from all_dices import *     # Εικόνες ζαριών dice1, dice2, ... dice6
from time import sleep      # Λειτουργία καθυστέρησης

def roll_dice():
    """Επιστρέφει μία ακέραια τιμή στο διάστημα [1, 6]"""
    d = randint(1, 6)
    return d

def wait_effect():
    """Δημιουργία εφέ καθυστέρησης μίας τελείας x 5 ανά 200 msecs"""
    print('.', end='')  # Τύπωσε μία τελεία και μην αλλάξεις γραμμή
    sleep(0.2)          # Περίμενε 200 msecs
    print('.', end='')
```

```
    sleep(0.2)
    print('.', end='')
    sleep(0.2)
    print('.', end='')
    sleep(0.2)
    print('.', end='')
    sleep(0.2)
    print('\nΡίξε μια ζαριά καλή')

def show_dices(d1, d2):
    """Δέχεται 2 ακέραιες τιμές στο διάστημα [1, 6] και
    εμφανίζει τις αντίστοιχες εικόνες από το άρθρωμα all_dices"""
    # Μετατροπή των ακεραίων σε string
    d1 = str(d1)
    d2 = str(d2)
    # Δημιούργησε τα ονόματα των εικόνων των ζαριών σε μορφή string
    d1 = 'dice' + d1
    d2 = 'dice' + d2
    # Μετατροπή των ονομάτων των εικόνων από string σε ονόματα μεταβλητών
    d1 = eval(d1)
    d2 = eval(d2)
    # Τύπωσε τις εικόνες των ζαριών
    print(d1)
    print(d2)

def main():
    """Ρίψη ζαριών, εφέ καθυστέρησης, εμφάνιση αποτελεσμάτων"""
    wait_effect()   # Κλήση συνάρτησης εφέ καθυστέρησης
    d1 = roll_dice ()     # Κλήση συνάρτησης για ρίψη του 1ου ζαριού
    d2 = roll_dice ()     # Κλήση συνάρτησης για ρίψη του 2ου ζαριού
    d1d2 = d1 + d2  # Υπολόγισε το άθροισμα των τιμών των ζαριών
    show_dices(d1, d2)# Κλήση συνάρτησης εμφάνισης των εικόνων των ζαριών
    print('Άθροισμα ρίψης', d1d2)   # Εμφάνιση αθροίσματος τιμών των ζαριών

main() # Ρίξε τα ζάρια
```

4.6.1 Δραστηριότητα

Συμπληρώστε τα κενά ώστε η παρακάτω συνάρτηση `random_ip()` *να επιστρέφει μία τυχαία ip vs4 δνση. Στη συνέχεια καλέστε την κατάλληλα ώστε να εμφανίσετε 4 τυχαίες ip δνσεις.*

```
from ________ import __________
def random_ip():
```

```
    part1 = ___(randint(0, ____)
    _____ = ___(_______(0, ____)
    _____ = ___(_______(0, ____)
    _____ = ___(_______(0, ____)
    ip = ______ + _____ part2 ______ part3 + ______ part4
    return __
```

4.6.2 Δραστηριότητα

Ένα κατάστημα χονδρικής πώλησης διαθέτει τα προϊόντα του αναγράφοντας τις τιμές πώλησης χωρίς το Φ.Π.Α. ο οποίος μπορεί να είναι είτε 24% είτε 13%.

Να υλοποιήσετε συμπληρώνοντας τα κενά την συνάρτηση `final_price()` *η οποία θα δέχεται ως είσοδο την τιμή του προϊόντος χωρίς Φ.Π.Α. και το ποσοστό του Φ.Π.Α. που επιβαρύνει το προϊόν και θα επιστρέφει την τελική τιμή του προϊόντος .*

Στη συνέχεια καλέστε τη συνάρτηση κατάλληλα ώστε να υπολογίσετε τις τελικές τιμές πώλησης για τα προϊόντα με τις παρακάτω αρχικές τιμές και τα αντίστοιχα ποσοστά Φ.Π.Α.

α. τιμή 80 και ΦΠΑ 24%
β. τιμή 230 και ΦΠΑ 24%
γ. τιμή 24 και ΦΠΑ 13%
δ. τιμή 65 και ΦΠΑ 13%

```
___ final_price(price, pfpa):
    fpa = (______ * _____) / 100
    final = ______ + ______

    return final
```

Υλοποιήστε την αντίστροφη συνάρτηση `starting_price()` *η οποία θα δέχεται την τελική τιμή ενός προϊόντος το ποσοστό Φ.Π.Α. που έχει επιβαρυνθεί και θα επιστρέφει την αρχική τιμή του προϊόντος πριν την επιβάρυνση με το Φ.Π.Α.;*

```
___ starting_price(price, pfpa):
    ___________________________
    ___________________________
    ___________________________
    ___________________________
    ___________________________
```

4.6.3 Δραστηριότητα

Υλοποιήστε τη συνάρτηση `trip_cost()` *η οποία θα δέχεται την κατανάλωση λίτρων καυσίμου ενός οχήματος ανά 100 χιλιόμετρα (lit/100 Km), την τιμή ανά λίτρο του καυσίμου (€ / lit) και την απόσταση σε χιλιόμετρα που θα διανυθεί (KM) και θα επιστρέφει το εκτιμώμενο κόστος της διαδρομής στρογγυλοποιημένο στο 2ο δεκαδικό ψηφίο. Στη συνέχεια καλέστε τη συνάρτηση με τα παρακάτω δεδομένα:*

Κατανάλωση (lit/100KM)	*Τιμή Καυσίμου ανά lit*	*Απόσταση*	*Επιστροφή Κόστους διαδρομής*
8.2	*1.8*	*30*	*4.43*
6	*1.8*	*30*	*3.24*
7.2	*1.45*	*500*	*52.2*

Έπειτα χρησιμοποιήστε τη συνάρτηση σε πρόγραμμα που θα υλοποιήσετε το οποίο θα ζητάει από το χρήστη την απόσταση που επιθυμεί να διανύσει, την τρέχουσα τιμή του καυσίμου καθώς και την κατανάλωση του οχήματός του και θα εμφανίζει μήνυμα που θα περιλαμβάνει το κόστος της διαδρομής καθώς και το χρονικό αποτύπωμα του υπολογισμού (ημερομηνία και ώρα).

4.6.4 Δραστηριότητα

Δημιουργήστε ένα αρχείο με όνομα activities4_6.py και στη συνέχεια :

- *αντιγράψτε σε αυτό όλες τις συναρτήσεις των προηγούμενων δραστηριοτήτων*
- *τοποθετήστε σε αυτές τεκμηρίωση της λειτουργίας τους*
- *υλοποιήστε μία νέα συνάρτηση* `test_data` *όπου θα καλείται με δοκιμαστικά δεδομένα για όλες τις προηγούμενες συναρτήσεις ώστε να εμφανίζουν αποτελέσματα*
- *στο τέλος του αρθρώματος τοποθετήστε συντάσσοντας κατάλληλα την εντολή* `if __name__ == '__main__'`*: ώστε όταν το πρόγραμμα «τρέχει» να εκτελείται η συνάρτηση* `test_data` *εμφανίζοντας τα δοκιμαστικά αποτελέσματα, ενώ όταν το πρόγραμμα φορτώνεται ως άρθρωμα λογισμικού να εμφανίζεται ένα μήνυμα ορθής φόρτωσής του καθώς και βοήθεια για όλες τις συναρτήσεις που περιέχει.*

4.7 Αφαιρετικότητα

Η έννοια της αφαιρετικότητας (abstraction) είναι ιδιαίτερα σημαντική στον προγραμματισμό. Συνίσταται στη δυνατότητα να αποκρύπτονται οι λεπτομέρειες υλοποίησης μίας λειτουργίας από την εφαρμογή αυτής της λειτουργίας, παρέχοντας στον προγραμματιστή έναν απλοποιημένο τρόπο διεπαφής στη χρήση της.

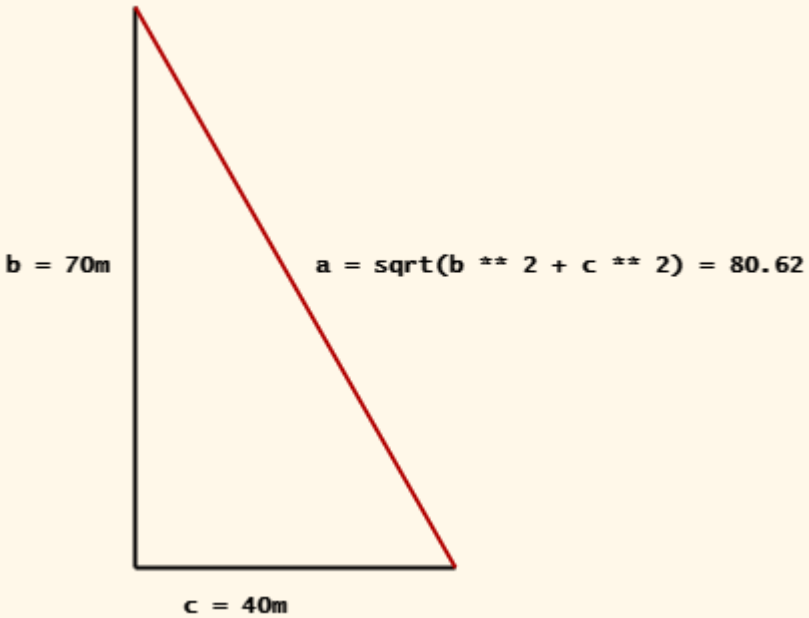
Οι συναρτήσεις παρέχουν αυτή τη δυνατότητα αφού για την χρήση τους αρκεί να είναι γνωστός ο τρόπος κλήσης τους (όνομα + τυπικές παράμετροι = διεπαφή) χωρίς να χρειάζεται να είναι γνωστός ο τρόπος υλοποίησης των λειτουργιών υπολογισμού.

Επιπλέον κάθε φορά που αυξάνεται το πλήθος των παραμέτρων μίας συνάρτησης τόσο αυξάνεται ο βαθμός αφαίρεσής της, γίνεται πιο γενική και μπορεί να χρησιμοποιηθεί σε ευρύτερο πλήθος προβλημάτων.

Παράδειγμα 4.36

Κάθε φορά που χρειάζεται ο υπολογισμός μίας τετραγωνικής ρίζας ενός αριθμού ο προγραμματιστής δεν χρειάζεται να υλοποιήσει κάποιον από τους αλγορίθμους υπολογισμού της τετραγωνικής ρίζας, αλλά αντί αυτού μπορεί να φορτώσει από το άρθρωμα λογισμικού `math` την αντίστοιχη λειτουργία `sqrt` από αυτό και να την καλέσει για τον υπολογισμό της τετραγωνικής ρίζας ενός αριθμού. Αυτό μπορεί να γίνει **χωρίς να τον ενδιαφέρει καθόλου με ποιον τρόπο και με ποιον αλγόριθμο** οι δημιουργοί της γλώσσας Python έχουν υλοποιήσει τον υπολογισμό της τετραγωνικής ρίζας μέσα στη συνάρτηση αυτή.

Έτσι για να υπολογισθεί σε ένα τριγωνικό οικόπεδο της παρακάτω μορφής ορθογωνίου τριγώνου η κόκκινη πλευρά του (υποτείνουσα) χρησιμοποιώντας το πυθαγόρειο θεώρημα ($a^2 = b^2 + c^2$) υλοποιείται η συνάρτηση `hypotenuse()` η οποία δέχεται ως είσοδο τα γνωστά μήκη των 2 καθέτων πλευρών και επιστρέφει το μήκος της υποτείνουσας.




```
def hypotenuse(b, c):
    from math import sqrt
    a = sqrt(b ** 2 + c ** 2)
```

```
    return a

print(hypotenuse(70, 40))

80.62257748298549
```

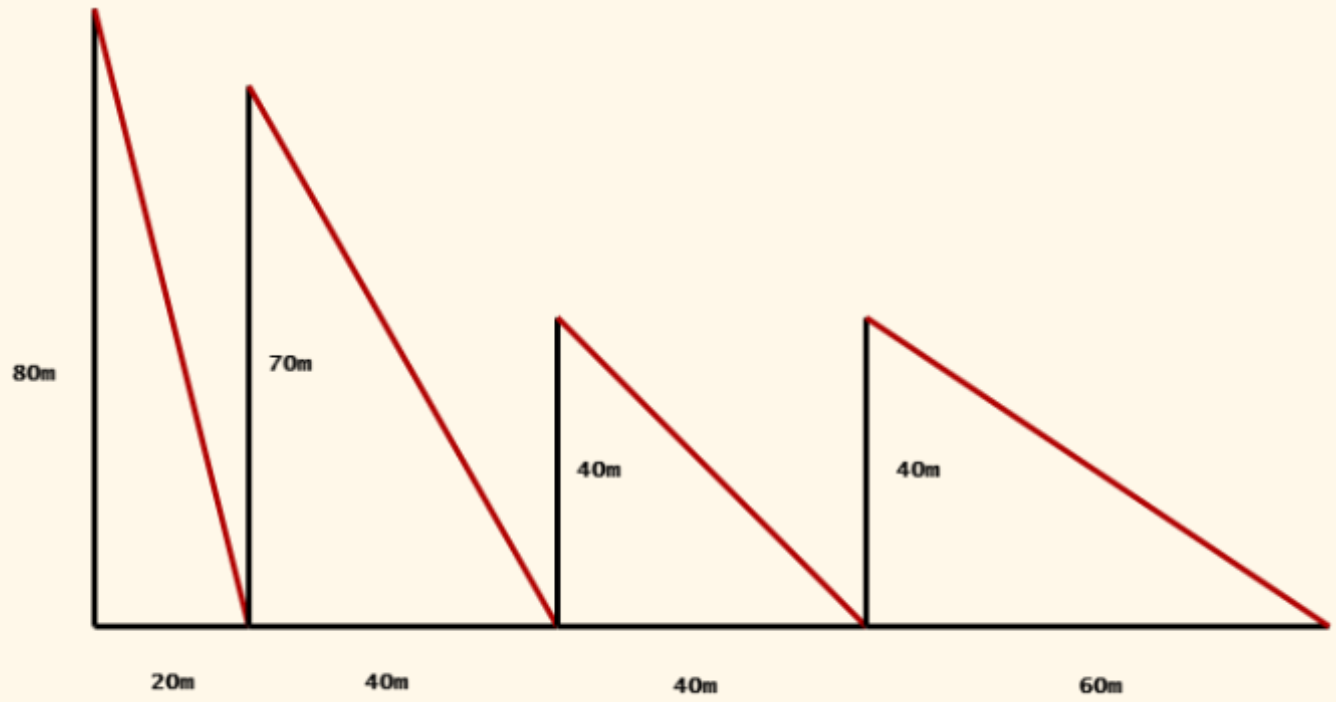
Η προηγούμενη συνάρτηση `hypotenuse()` θα λειτουργεί ακριβώς με τον ίδιο τρόπο **ακόμα και εάν οι δημιουργοί της γλώσσας Python** αποφασίσουν να υλοποιήσουν σε επόμενη έκδοση της γλώσσας διαφορετικό τρόπο, ακρίβεια ή και αλγόριθμο υπολογισμού της τετραγωνικής ρίζας ενός αριθμού.

Εφόσον η διεπαφή παραμένει ίδια `sqrt(x)` η συνάρτηση `hypotenuse()` του προγραμματιστή δε χρειάζεται να αλλάξει.

Παράδειγμα 4.37

Τη συνάρτηση `hypotenuse` που υλοποιήθηκε στο προηγούμενο παράδειγμα τη χρησιμοποιεί ένας πολιτικός μηχανικός για να υπολογίσει όλα τα παρακάτω (κόκκινα μήκη) ενός ακινήτου σε ένα πρόγραμμα που θα φτιάξει.

Για να υλοποιήσει το πρόγραμμά του το μόνο που πρέπει να γνωρίζει είναι η διεπαφή της προηγούμενης συνάρτησης hypotenuse.




```
print(hypotenuse(80, 20))
print(hypotenuse(70, 40))
print(hypotenuse(40, 40))
print(hypotenuse(40, 60))

82.46211251235322
80.62257748298549
```

```
56.568542494923804

72.11102550927978
```

Έτσι με την αρχή της αφαιρετικότητας ακόμα και εάν αλλάξει ο τρόπος υπολογισμού της υποτείνουσας (συνάρτηση `hypotenuse`) χρησιμοποιώντας την ιδιότητα των δυνάμεων από τα μαθηματικά:

$$\alpha^{\frac{\kappa}{\lambda}} = \sqrt[\lambda]{\alpha^{\kappa}}$$

$$\alpha^{\frac{\kappa}{\lambda}} = \sqrt[\lambda]{\alpha^{\kappa}}$$

στην παρακάτω μορφή:

```
def hypotenuse(b, c):
    return (b ** 2 + c ** 2) ** 0.5

print(hypotenuse(70, 40))

80.62257748298549
```

εφόσον διατηρηθεί η διεπαφή της ίδια, το πρόγραμμα του πολιτικού μηχανικού δε θα χρειαστεί καμία αλλαγή.

Οι πολιτικοί μηχανικοί γνωρίζουν βέβαια και μαθηματικά και γεωμετρία και θα μπορούσαν ενδεχομένως να υπολογίσουν μόνοι τους τις πλευρές στα παραπάνω οικόπεδα.

Τη συνάρτηση όμως `hypotenuse` μπορεί να τη χρησιμοποιήσει και κάποιος ο οποίος θέλει απλά να υπολογίσει πόσα μέτρα σύρμα χρειάζονται οι κόκκινες πλευρές για να περιφραχθούν και δε θυμάται πολλά από μαθηματικά και γεωμετρία επιτυγχάνοντας έτσι τον απαιτούμενο βαθμό αφαίρεσης στο πρόβλημά του αφού ασχολείται μόνο με την περίφραξη, την αγορά υλικών και τις εργασίες τοποθέτησης και καθόλου με το μαθηματικό κομμάτι των υπολογισμών.

Συνοψίζοντας οι συναρτήσεις στην Python παρέχουν έναν τρόπο να επιτύχουμε αφαιρετικότητα και να αποκρυφθούν οι λεπτομέρειες υλοποίησης μιας λειτουργίας. Μπορούν να χρησιμοποιηθούν έτοιμες συναρτήσεις που εκτελούν συγκεκριμένες λειτουργίες χωρίς να είναι γνωστός ο κώδικας υλοποίησής τους. Αυτό επιτρέπει την οργάνωση του κώδικα σε λογικές μονάδες την αύξηση της αναγνωσιμότητας, την επαναχρησιμοποίηση, την απλοποίηση και τη πιο εύκολη συντήρησή του.

4.7.1 Δραστηριότητα

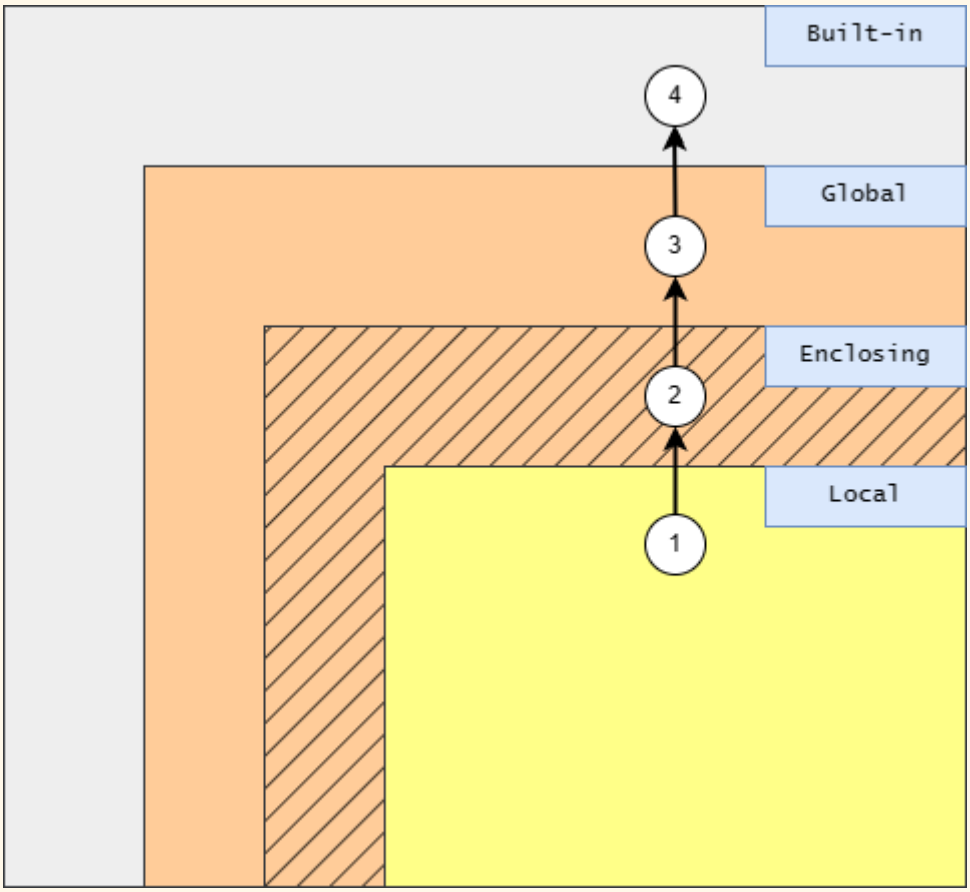
Σε επόμενο κεφάλαιο θα σας ζητηθεί να τροποποιήσετε την εφαρμογή του παραδείγματος dices όπου θα αυξήσετε το βαθμό αφαίρεσής του τροποποιώντας τον κώδικά του ώστε να επιλέγει ο χρήστης το πλήθος των ζαριών που θα εμφανίζονται στην οθόνη για να μπορεί να χρησιμοποιηθεί το ίδιο πρόγραμμα για να φτιάξετε διάφορα παιχνίδια που χρησιμοποιούν 1 ή 2 ή 3 ή 4 ζάρια.

4.8 Εμβέλεια μεταβλητών

Η εμβέλεια των μεταβλητών (variable scope) καθορίζει το χώρο και τον τρόπο με τον οποίο μπορούν οι μεταβλητές να προσπελαθούν σε ένα πρόγραμμα ή μία συνάρτηση. Η εμβέλεια μίας μεταβλητής σε ένα πρόγραμμα Python καθορίζεται από τη θέση της δήλωσής της (χρήσης της). Στη γλώσσα Python μία μεταβλητή μπορεί να είναι :

- **Τοπικής Εμβέλειας** (Local Scope): Οι μεταβλητές που ορίζονται εντός μίας συνάρτησης και έχουν τοπική εμβέλεια δηλαδή είναι προσβάσιμες μόνο εντός της συνάρτησης που ορίζονται.
- **Εμβέλειας ενθυλακωμένων μεταβλητών** (Enclosing Scope): Αφορά στις μεταβλητές οι οποίες ορίζονται σε εμφωλευμένες συναρτήσεις (δεν θα γίνει περαιτέρω αναφορά σε αυτό το κεφάλαιο).
- **Καθολικής Εμβέλειας** (Global Scope): Οι μεταβλητές που ορίζονται στο πρόγραμμα και είναι διαθέσιμες και προσβάσιμες από όλα τα σημεία του προγράμματος.
- **Ενσωματωμένης εμβέλειας** (Built-In Scope): Περιλαμβάνει τις ενσωματωμένες συναρτήσεις της γλώσσας που είναι διαθέσιμες σε όλα τα σημεία ενός προγράμματος.

Η προτεραιότητα πρόσβασης στα ονόματα μεταβλητών, συναρτήσεων, (γενικά αντικειμένων) στην Python ακολουθεί κατά προτεραιότητα τη λογική εκτέλεσης του παρακάτω σχήματος δηλαδή τη λογική εκτέλεσης (L→ E → G → B)



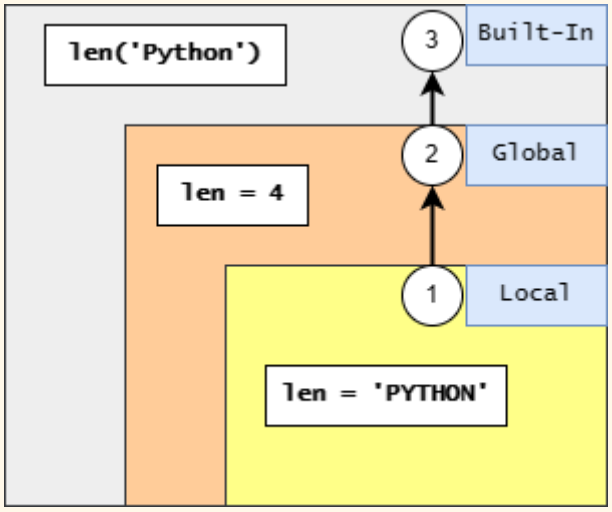
Σημείωση

Η εμβέλεια μεταβλητών (variable scope) έχει συναφή σημασία με το χώρο ονομάτων (namespaces).

Παράδειγμα 4.38

Στο παρακάτω παράδειγμα χρησιμοποιείται το όνομα της συνάρτησης len:

α. Ως ενσωματωμένη συνάρτηση
β. Ως καθολική μεταβλητή εντός του προγράμματος
γ. Ως τοπική μεταβλητή εντός της συνάρτησης func




```
# Δεν έχει οριστεί ακόμα άλλο αντικείμενο με το όνομα len
# οπότε καλείται η ενσωματωμένη συνάρτηση με αυτό το όνομα
print(len('PYTHON'), 'Built In')

# Ορίζεται η καθολική μεταβλητή len οπότε καλείται αυτή κατά
# προτεραιότητα έναντι της ενσωματωμένης συνάρτησης με το ίδιο όνομα
len = 4
print(len, 'Global')

# Εντός της συνάρτησης ορίζεται η τοπική μεταβλητή len οπότε
# καλείται αυτή κατά προτεραιότητα έναντι της καθολικής με το ίδιο όνομα
# αλλά και της ενσωματωμένης συνάρτησης με το ίδιο όνομα
def func():
    len = 'PYTHON'
    local_var = 'in function'
    print (len * 2, 'Local')


func() # Κλήση της συνάρτησης

# Η καθολική μεταβλητή έχει προτεραιότητα έναντι της ενσωματωμένης
συνάρτησης
print (len, 'Global')

# Ενώ η τοπική μεταβλητή local_var της συνάρτησης func είναι άγνωστη εδώ
```

```
print(local_var, 'local var of Func')
```

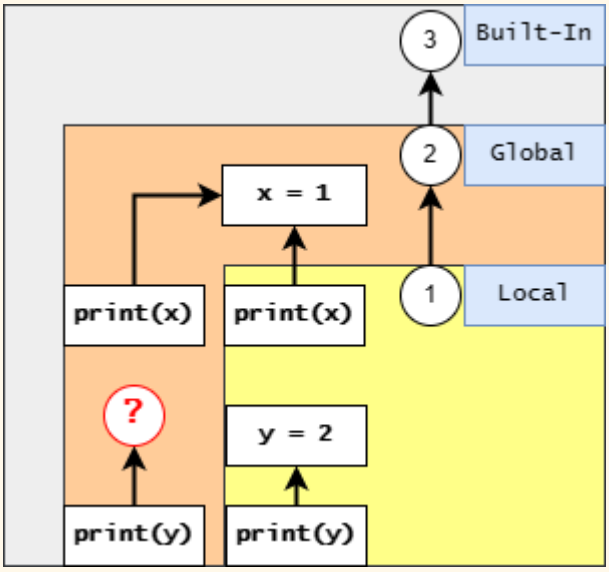
Το αποτέλεσμα της εκτέλεσης του παραπάνω κώδικα

```
6 Built In
4 Global
PYTHONPYTHON Local
4 Global
Traceback (most recent call last):
  File "C:/Python311/my_scripts/DIEK/kef_04/scope1.py", line 24, in
<module>
    print(local_var, 'local var of Func')
NameError: name 'local_var' is not defined
```

Παράδειγμα 4.39

Συντάξτε και εκτελέστε το παρακάτω πρόγραμμα για να δείτε τη λειτουργία της καθολικής μεταβλητής x και της τοπικής μεταβλητής y.

```
x = 1 # Καθολική Μεταβλητή


def func():
    y = 2   # Τοπική Μεταβλητή
    print(x, 'Καθολική Μεταβλητή εντός συνάρτησης')
    print(y, 'Τοπική Μεταβλητή εντός της συνάρτησης')

func()
print(x, 'Καθολική μεταβλητή εκτός συνάρτησης')
print(y, 'Τοπική Μεταβλητή εκτός συνάρτησης (θα δώσει λάθος)')
```



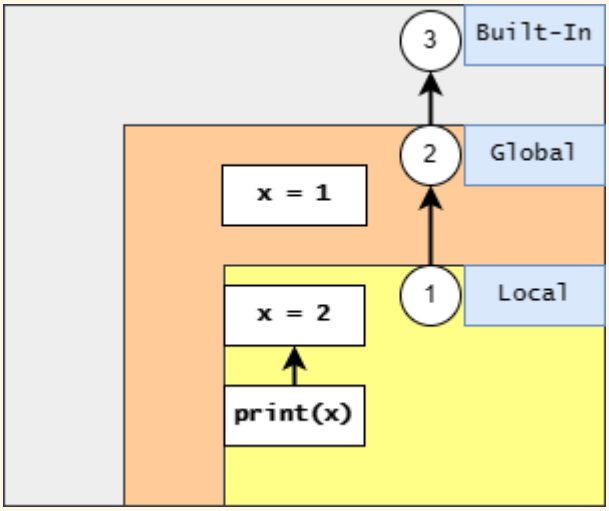

```
1 Καθολική Μεταβλητή εντός συνάρτησης
```

```
2 Τοπική Μεταβλητή εντός της συνάρτησης
1 Καθολική μεταβλητή εκτός συνάρτησης
Traceback (most recent call last):
  File "C:/Python311/my_scripts/DIEK/kef_04/scope2.py", line 11, in
<module>
    print(y, 'Τοπική Μεταβλητή εκτός συνάρτησης (θα δώσει λάθος)')
NameError: name 'y' is not defined
```

Παράδειγμα 4.40

Συντάξτε και εκτελέστε το παρακάτω πρόγραμμα για να δείτε τη λειτουργία της καθολικής μεταβλητής x και της τοπικής μεταβλητής x, μπορεί να έχουν το ίδιο όνομα είναι όμως διαφορετικά αντικείμενα. Η εντολή `x = 2` *εντός της συνάρτησης ορίζει ένα νέο αντικείμενο x το οποίο «ζει», έχει εμβέλεια μόνο εντός της συνάρτησης ως τοπική μεταβλητή.*

```
# Καθολική μεταβλητή
x = 1


def func():
    x = 2   # Τοπική Μεταβλητή
    print(x, 'Τοπική Μεταβλητή εντός της συνάρτησης')

func()
print(x, 'Καθολική μεταβλητή εκτός συνάρτησης')
```




```
2 Τοπική Μεταβλητή εντός της συνάρτησης
1 Καθολική μεταβλητή εκτός συνάρτησης
```

4.8.1 Δραστηριότητες

α. Μπορείτε να προβλέψετε τα αποτελέσματα εκτέλεσης του παρακάτω κώδικα:

```
# Καθολική μεταβλητή
x = 1
```

```
def func():
    x = x + 2
    print(x)

func()
print(x)
```

β. Μπορείτε να προβλέψετε τα αποτελέσματα εκτέλεσης του παρακάτω κώδικα:

```
# Καθολική μεταβλητή
x = 1


def func():
    print(x)
    x = x + 2
    print(x)

func()
print(x)
```

γ. Μπορείτε να προβλέψετε τα αποτελέσματα εκτέλεσης του παρακάτω κώδικα:

```
# Καθολική μεταβλητή
x = 1


def func():
    print(x)

func()
print(x)
```

δ. Μπορείτε να προβλέψετε τα αποτελέσματα εκτέλεσης του παρακάτω κώδικα:

```
# Καθολική μεταβλητή
x = 1


def func():
    global x
    print(x)
    x = x + 2
    print(x)

func()
print(x)
```

4.9 Να θυμάμαι

Αντικείμενα

Στη γλώσσα προγραμματισμού Python, τα αντικείμενα (objects) αποτελούν τον πυρήνα της γλώσσας και αναπαριστούν τα δομικά στοιχεία του προγραμματισμού. Ένα αντικείμενο είναι μια συλλογή δεδομένων (μεταβλητές) και λειτουργιών (μεθόδων) που σχετίζονται με αυτά τα δεδομένα. Στη γλώσσα Python, όλα είναι αντικείμενα. Ακόμη και οι απλές μορφές δεδομένων, όπως οι αριθμοί (ακέραιοι ή κινητής υποδιαστολής) και οι συμβολοσειρές, είναι αντικείμενα.

Συνάρτηση

(Function) στη γλώσσα Python είναι ένα τμήμα κώδικα το οποίο εκτελεί μία συγκεκριμένη λειτουργία. Μπορεί να διαθέτει είσοδο δεδομένων με τη χρήση μεταβλητών στην κεφαλίδα της οι οποίες ονομάζονται τυπικές παράμετροι (formal parameters), διαθέτει κύριο σώμα όπου υλοποιούνται οι βασικές υπολογιστικές λειτουργίες και να επιστρέφει ένα αντικείμενο ή να εκτυπώνει αποτελέσματα ή να τροποποιεί κάποιο αντικείμενο. Ο ορισμός μίας συνάρτησης χρήστη μπορεί να γίνει μέσω της ειδικής εντολής `def` (ορίζω).

Κλήση της συνάρτησης

Η διαδικασία κατά την οποία συντάσσεται το όνομα της συνάρτησης με ή χωρίς παραμέτρους ανάλογα τις απαιτήσεις της και στη συνέχεια εκτελείται ονομάζεται κλήση της συνάρτησης. Κατά την κλήση της συνάρτησης από το πρόγραμμα καθορίζονται οι τιμές που θα λάβουν οι παράμετροι αυτοί, οι τιμές αυτές ονομάζονται ορίσματα (arguments) ή πραγματικές παράμετροι (actual parameters).

Χρησιμότητα Συναρτήσεων

Οι συναρτήσεις χρησιμοποιούνται στην υλοποίηση για την οργάνωση του κώδικα σε μικρότερα, αυτόνομα τμήματα τα οποία μπορούν να κληθούν από άλλα προγράμματα ή και από άλλες συναρτήσεις.

Η χρήση συναρτήσεων κατά τη συγγραφή κώδικα παρουσιάζει πολλά πλεονεκτήματα:

- **Επαναχρησιμοποίηση κώδικα** (code reuse): Μια συνάρτηση ορίζεται μία φορά και μπορεί να κληθεί πολλές φορές σε διάφορα μέρη του προγράμματος. Αυτό επιτρέπει την αποφυγή της επανάληψης συγγραφής του ίδιου κώδικα και διευκολύνει τη συντήρηση και αναβάθμιση του προγράμματος. Επίσης συναρτήσεις που επιτελούν συχνές (δημοφιλείς) λειτουργίες μπορούν να αποτελέσουν μέρη αρθρωμάτων όπως θα μελετηθεί παρακάτω και να χρησιμοποιηθούν και από άλλους προγραμματιστές.
- **Δομή και οργάνωση του κώδικα** (code structure): Οι συναρτήσεις βοηθούν στον διαχωρισμό του κώδικα σε λογικά αυτόνομες μονάδες. Αυτό καθιστά τον κώδικα πιο οργανωμένο, πιο ευανάγνωστο και πιο εύκολο στη συντήρηση.
- **Αφαίρεση πολυπλοκότητας** (abstraction): Με τη χρήση συναρτήσεων, μειώνεται η πολυπλοκότητα του κύριου κώδικα. Ενώ μία συνάρτηση μπορεί να έχει σύνθετη λειτουργικότητα η απλή κλήση της από τον κύριο κώδικα, κάνει το κύριο πρόγραμμα πιο κατανοητό και ευκολότερο στη διόρθωση και στη συντήρηση.

Οι συναρτήσεις γενικά στις γλώσσες προγραμματισμού ανήκουν στα ΥΠΟ-προγράμματα και μαζί με τις διαδικασίες (procedures) και αποτελούν τη βάση του τμηματικού προγραμματισμού και εργαλείο της ιεραρχικής σχεδίασης προγραμμάτων. Στη γλώσσα Python δεν υπάρχουν διαδικασίες αλλά μόνο συναρτήσεις.

Ενσωματωμένες Συναρτήσεις

Η γλώσσα προγραμματισμού Python προσφέρει μια πληθώρα ενσωματωμένων συναρτήσεων (άμεσα διαθέσιμες στον προγραμματιστή), εκτελώντας βασικές λειτουργίες που παρέχονται **απευθείας** από τη γλώσσα χωρίς την ανάγκη εισαγωγής κάποιας επιπλέον βιβλιοθήκης. Αυτές οι ενσωματωμένες συναρτήσεις παρέχουν έτοιμες λειτουργίες και χρησιμοποιούνται ευρέως για την εκτέλεση διαφόρων εργασιών και επεξεργασία δεδομένων. Αναφέρθηκαν οι παρακάτω ενσωματωμένες συναρτήσεις και οι λειτουργίες τους:

`print()`, `input()`, `len()`, `abs()`, `int()`, `float()`, `str()`, `bool()`, `type()`, `pow()`, `divmod()`, `round()`, `bin()`, `hex()`, `eval`, `help()`

Αρθρώματα Λογισμικού (modules)

Τα αρθρώματα λογισμικού γενικά είναι κομμάτια λογισμικού που παρέχουν έτοιμες λειτουργίες και λύσεις για διάφορες προγραμματιστικές ανάγκες. Τα αρθρώματα λογισμικού που έχει ο χρήστης τη δυνατότητα να χρησιμοποιήσει στο πρόγραμμά του μπορεί να είναι:

- ενσωματωμένα στη γλώσσα (built-in) ή
- σε εξωτερικές βιβλιοθήκες λογισμικού τις οποίες πρέπει να κατεβάσει και να εγκαταστήσει πριν τις χρησιμοποιήσει από τα αποθετήρια λογισμικού
- δικής του υλοποίησης που θα ενσωματώνει στο πρόγραμμά του

Αναφέρθηκαν τα παρακάτω ενσωματωμένα αρθρώματα και κάποιες από τις λειτουργίες τους:

`math()`, `random()`, `datetime()`, `time()`, `os()`

Δομή συναρτήσεων

Μία συνάρτηση έχει δύο βασικά μέρη, την κεφαλίδα και το κυρίως σώμα της. Η κεφαλίδα αρχίζει όπως αναφέρθηκε με τη δεσμευμένη εντολή(keyword) `def` (define: ορίζω) στη συνέχεια ακολουθεί το όνομα της συνάρτησης ακολουθούμενο από ένα ζεύγος παρενθέσεων μέσα στο οποίο τοποθετούνται εάν υπάρχουν οι τυπικές παράμετροι (formal parameters) της συνάρτησης διαχωρισμένες με κόμμα(,).

Κάποιες συναρτήσεις επιστρέφουν μία τιμή με την εντολή return άλλες μπορεί να εκτελούν λειτουργίες χωρίς να επιστρέφουν κάποια τιμή.

Στη συνέχεια αφού οριστεί μία συνάρτηση χρήστη, αυτή πρέπει να κληθεί από το κύριο πρόγραμμα ώστε να εκτελεστούν οι λειτουργίες της. Εάν διαθέτει δε και παραμέτρους σε αυτές πρέπει κατά την κλήση της να δοθούν και πραγματικές τιμές. Όταν ολοκληρωθεί η εκτέλεσή της συνάρτησης η εντολή `return` επιστρέφει τα αποτελέσματα των υπολογισμών στο κύριο πρόγραμμα.

Σημειώσεις στις συναρτήσεις

- *Εάν δεν υπάρχει εντολή return σε μία συνάρτηση, τελειώνει η εκτέλεσή της με την τελευταία εντολή και επιστρέφεται στο κύριο πρόγραμμα ένα αντικείμενο ειδικού τύπου με τιμή* `None`.
- *Συνήθως αναφέρεται ότι η συνάρτηση επιστρέφει μία τιμή ή τιμές. Το ορθό όμως είναι ότι μία συνάρτηση στην γλώσσα Python επιστρέφει ΠΑΝΤΑ ένα αντικείμενο κάποιου τύπου πχ, int, float, str, bool, list, tuple, dict, set, None κα. Γενικά ο όρος τιμή σε άλλες γλώσσες στην Python όπως θα αναλυθεί και σε επόμενα κεφάλαια είναι ταυτόσημος με τον όρο αντικείμενο.*
- *Όταν υπάρχει η εντολή return σε μία συνάρτηση, γίνεται αποτίμηση της έκφρασης που ακολουθεί την εντολή return και γίνεται επιστροφή του αντικειμένου εξόδου που παράγεται από την έκφραση.*
- *Η εντολή return σταματά την εκτέλεση του σώματος της συνάρτησης, οπότε εάν υπάρχει εντολή μετά από αυτήν απλά δεν εκτελείται.*

- *Εάν μία συνάρτηση κληθεί με πραγματικές παραμέτρους εκφράσεις, γίνεται πρώτα η αποτίμηση των εκφράσεων, υπολογίζεται η τιμή τους και περνά αυτή η τιμή (αντικείμενο) στην τυπική παράμετρο της συνάρτησης.*
- *Οι συναρτήσεις που δημιουργούνται από το χρήστη είναι σκόπιμο να συνοδεύονται από βοήθεια σχετική με τη λειτουργία και τη χρήση τους. Αυτό γίνεται τοποθετώντας ένα multiline string κάτω από την κεφαλίδα τους το οποίο αποτελεί μέρος της τεκμηρίωσης της συνάρτησης.*
- *Είναι εφικτό κατά την υλοποίηση ενός προγράμματος ή μίας συνάρτησης χρήστη να γίνει κλήση μίας συνάρτησης από μία άλλη συνάρτηση καθώς και μία άλλη κλήση συνάρτησης μέσα από αυτήν κοκ για να διατηρείται μία καταγραφή της σειράς εκτέλεσης των εντολών χρησιμοποιείται η Στοίβα Χρόνου Εκτέλεσης.*

Ιεραρχική Σχεδίαση και Τμηματικός Προγραμματισμός

Η ιεραρχική σχεδίαση επίλυσης ενός σύνθετου συνήθως προβλήματος περιλαμβάνει τη διαδικασίας διάσπασής του σε μικρότερα (και άρα απλούστερα) προβλήματα τα οποία είναι πιο εύκολο να επιλυθούν και η λύση των οποίων επιλύει και το αρχικό πρόβλημα. Η τεχνική αυτή σχεδίασης ονομάζεται και από επάνω προς τα κάτω (top down design). Υλοποιείται με τον τμηματικό προγραμματισμό όπου επιλύεται κάθε υποπρόβλημα με ένα υποπρόγραμμα (στην Python με συνάρτηση). Η σύνθεση των επιμέρους υποπρογραμμάτων οδηγεί και στη λύση του αρχικού προβλήματος.

Ο τμηματικός προγραμματισμός μειώνει τα λάθη, επιτρέπει την ευκολότερη παρακολούθηση, κατανόηση και συντήρηση του προγράμματος από τρίτους.

Αρθρώματα Λογισμικού Χρήστη

Τα αρθρώματα λογισμικού δεν είναι τίποτα άλλο από αρχεία κώδικα σε γλώσσα Python με επέκταση .py το όνομα των οποίων πρέπει να αρχίζει από λατινικό γράμμα. Συνήθως περιέχουν συναρτήσεις, μεταβλητές (αντικείμενα διαφόρων τύπων), ή και αργότερα όπως θα συναντήσουμε σε επόμενο κεφάλαιο κλάσεις.

Η δημιουργία αρθρωμάτων από το χρήστη παρουσιάζει **χρησιμότητα**:

- είτε σε περιπτώσεις επαναχρησιμοποίησης ολόκληρων λειτουργιών και τμημάτων του κώδικα σε άλλα προγράμματα ώστε να μειώνεται ο χρόνος συγγραφής τους και να απλοποιείται ο κώδικάς τους,
- είτε σε περιπτώσεις σχεδίασης μέσω του τμηματικού προγραμματισμού όπου ο κώδικας μπορεί να διαχωρίζεται ομαδοποιώντας τις συναφείς λειτουργίες του σε μικρότερα αρχεία που θα καλούνται τελικά από κάποιο κύριο αρχείο διευκολύνοντας έτσι τον καταμερισμό εργασιών, τη συντήρηση και τη διόρθωσή τους.

Η **φόρτωση** των αρθρωμάτων του χρήστη γίνεται με παρόμοιους τρόπους όπως και των ενσωματωμένων αρθρωμάτων της γλώσσας

```
α. import module_name
β. from module_name import function1, function2
γ. from module_name import *
```

Αφαιρετικότητα

Η έννοια της αφαιρετικότητας (abstraction) είναι ιδιαίτερα σημαντική στον προγραμματισμό. Συνίσταται στη δυνατότητα να αποκρύπτονται οι λεπτομέρειες υλοποίησης μίας λειτουργίας από την εφαρμογή αυτής της λειτουργίας, παρέχοντας στον προγραμματιστή έναν απλοποιημένο τρόπο διεπαφής στη χρήση της.

Οι συναρτήσεις παρέχουν αυτή τη δυνατότητα αφού για την χρήση τους αρκεί να είναι γνωστός ο τρόπος κλήσης τους (όνομα + τυπικές παράμετροι = διεπαφή) χωρίς να χρειάζεται να είναι γνωστός ο τρόπος υλοποίησής των λειτουργιών υπολογισμού.

Εμβέλεια Μεταβλητών

Η εμβέλεια των μεταβλητών (variable scope) καθορίζει το πού και πότε μπορούν οι μεταβλητές να προσπελαθούν σε ένα πρόγραμμα ή μία συνάρτηση. Η εμβέλεια μία μεταβλητής σε ένα πρόγραμμα Python καθορίζεται από τη θέση της δήλωσής της (χρήσης της). Στη γλώσσα Python μία μεταβλητή μπορεί να είναι :

- **Τοπικής Εμβέλειας** (Local Scope): Οι μεταβλητές που ορίζονται εντός μίας συνάρτησης και έχουν τοπική εμβέλεια δηλαδή είναι προσβάσιμες μόνο εντός της συνάρτησης που ορίζονται.
- **Εμβέλειας ενθυλακωμένων μεταβλητών** (Enclosing Scope): Αφορά στις μεταβλητές οι οποίες ορίζονται σε εμφωλευμένες συναρτήσεις (δεν θα γίνει περαιτέρω αναφορά σε αυτό το κεφάλαιο).
- **Καθολικής Εμβέλειας** (Global Scope): Οι μεταβλητές που ορίζονται στο πρόγραμμα και είναι διαθέσιμες και προσβάσιμες από όλα τα σημεία του προγράμματος.
- **Ενσωματωμένης εμβέλειας** (Built-In Scope): Περιλαμβάνει τις ενσωματωμένες συναρτήσεις της γλώσσας που είναι διαθέσιμες σε όλα τα σημεία ενός προγράμματος.

Η προτεραιότητα πρόσβασης στα ονόματα μεταβλητών, συναρτήσεων, (γενικά αντικειμένων) στην Python ακολουθεί κατά προτεραιότητα τη λογική εκτέλεσης του παρακάτω σχήματος δηλαδή τη λογική εκτέλεσης (L→ E → G → B)

4.10 Ερωτήσεις Κατανόησης

1. Μόνο κάποια από τα δεδομένα της γλώσσας Python αποτελούν αντικείμενα.

   ΝΑΙ ΟΧΙ

2. Η συνάρτηση είναι ένα τμήμα κώδικα το οποίο εκτελεί μία συγκεκριμένη λειτουργία.

   ΝΑΙ ΟΧΙ

3. Μία συνάρτηση υλοποιείται/δηλώνεται χρησιμοποιώντας την εντολή `define`.

   ΝΑΙ ΟΧΙ

4. Η εκτεταμένη χρήση συναρτήσεων αυξάνει την πολυπλοκότητα στον κώδικα του προγράμματος.

   ΝΑΙ ΟΧΙ

5. Η γλώσσα Python διαθέτει συναρτήσεις (Functions) και διαδικασίες (Procedures).

   ΝΑΙ ΟΧΙ

6. Οι ενσωματωμένες συναρτήσεις στην Python δεν χρειάζεται να φορτωθούν πριν χρησιμοποιηθούν.

   ΝΑΙ ΟΧΙ

7. Η χρήση συναρτήσεων αυξάνει την πολυπλοκότητα ενός προγράμματος.

   ΝΑΙ ΟΧΙ

8. Η ιεραρχική σχεδίαση επίλυσης ενός σύνθετου συνήθως προβλήματος περιλαμβάνει τη διαδικασίας διάσπασής του σε μικρότερα

   ΝΑΙ ΟΧΙ

9. Η συνάρτηση `abs` υπολογίζει την τετραγωνική ρίζα ενός αριθμού.

   ΝΑΙ ΟΧΙ

10. Η συνάρτηση `len` υπολογίζει την απόσταση ενός αντικειμένου από το μηδέν.

    ΝΑΙ ΟΧΙ

11. Η συνάρτηση `pow` εκτελεί παρόμοια λειτουργία με τον τελεστή **.

    ΝΑΙ ΟΧΙ

12. Η συνάρτηση `divmod` εκτελεί παρόμοια λειτουργία με τους τελεστές //, &.

    ΝΑΙ ΟΧΙ

13. Η συνάρτηση `round` χρησιμοποιείται για την μετατροπή ενός πραγματικού αριθμού σε ακέραιο.

ΝΑΙ ΟΧΙ

14. Η συνάρτηση `bin` μετακινεί ένα αρχείο στον κάδο ανακύκλωσης (διαγράφει δηλαδή).

ΝΑΙ ΟΧΙ

15. Η συνάρτηση `type` αλλάζει τον τύπο ενός αντικειμένου σε έναν άλλο.

ΝΑΙ ΟΧΙ

16. Η συνάρτηση `eval` αποτιμάει μία έκφραση στην αντίστοιχη εντολή της Python.

ΝΑΙ ΟΧΙ

17. Το άρθρωμα, το module, η βιβλιοθήκη είναι έννοιες ταυτόσημες.

ΝΑΙ ΟΧΙ

18. Τα αρθρώματα `math`, `random`, πρέπει να κατέβουν από κάποιο αποθετήριο λογισμικού προκειμένου να χρησιμοποιηθούν.

ΝΑΙ ΟΧΙ

19. Τα αρθρώματα `time`, `os` είναι ενσωματωμένα και μπορούν να φορτωθούν απευθείας στο πρόγραμμα χωρίς κάποια επιπλέον ενέργεια.

ΝΑΙ ΟΧΙ

20. Για να ορισθεί μία συνάρτηση χρήστη χρησιμοποιείται η εντολή `define`.

ΝΑΙ ΟΧΙ

21. Μία συνάρτηση μπορεί να καλέσει μία άλλη συνάρτηση.

ΝΑΙ ΟΧΙ

22. Η Στοίβα Χρόνου Εκτέλεσης χρησιμοποιείται για να διατηρείται στον προγραμματισμό η σειρά ελέγχου εκτέλεσης των εντολών

ΝΑΙ ΟΧΙ

23. Η εντολή `if __name__ == '__main__'` χρησιμοποιείται στο τέλος ενός αρθρώματος για να είναι δυνατή η φόρτωσή του από ένα πρόγραμμα.

ΝΑΙ ΟΧΙ

24. Η έννοια της αφαιρετικότητα στον προγραμματισμό σχετίζεται άμεσα μα τη μαθηματική πράξη της αφαίρεσης.

ΝΑΙ ΟΧΙ

25. Η έννοια της εμβέλειας των μεταβλητών σχετίζεται άμεσα με τον τύπο του κάθε αντικειμένου.

ΝΑΙ ΟΧΙ

26. Η έννοια της εμβέλειας των μεταβλητών σχετίζεται άμεσα με τον τύπο του κάθε αντικειμένου.

ΝΑΙ ΟΧΙ

27. Η προτεραιότητα πρόσβασης στα ονόματα μεταβλητών, συναρτήσεων, (γενικά αντικειμένων) στην Python ακολουθεί κατά προτεραιότητα τη λογική εκτέλεσης του L→ E→ G→ O

ΝΑΙ ΟΧΙ

28. Κάθε συνάρτηση πρέπει να έχει τυπικές παραμέτρους.

ΝΑΙ ΟΧΙ

29. Κάθε φορά που καλείται μία συνάρτηση πρέπει να δίνονται ορίσματα.

ΝΑΙ ΟΧΙ

30. Οι έννοιες ορίσματα και πραγματικές παράμετροι είναι ταυτόσημες.

ΝΑΙ ΟΧΙ

31. Όλες οι συναρτήσεις επιστρέφουν ένα αντικείμενο κάποιου τύπου.

ΝΑΙ ΟΧΙ

4.11 Ερωτήσεις Ανάπτυξης

1. Τι είναι το Αντικείμενο σε μία γλώσσα προγραμματισμού;
2. Τι ονομάζουμε κλήση συνάρτησης;
3. Να αναφέρετε τα πλεονεκτήματα συγγραφής κώδικα με χρήση συναρτήσεων;
4. Τι ενέργειες περιλαμβάνει η ιεραρχική σχεδίαση ενός προγράμματος και πως υλοποιείται με τη χρήση του τμηματικού προγραμματισμού;
5. Να αναφέρετε 5 ενσωματωμένες συναρτήσεις της γλώσσας Python;
6. Τι ονομάζουμε άρθρωμα λογισμικού (module) και τι περιέχει αυτό;
7. Να αναφέρετε 3 ενσωματωμένα αρθρώματα λογισμικού της γλώσσας Python;
8. Περιγράψτε τη δομή μίας συνάρτησης χρήστη στη γλώσσα Python;
9. Ποιος ο ρόλος της στοίβας χρόνου εκτέλεσης και πως λειτουργεί;
10. Ποια η χρησιμότητα των αρθρωμάτων λογισμικού στην γλώσσα Python;
11. Περιγράψτε το ρόλο της εντολής `if __name__ == '__main__'` στα αρθρώματα λογισμικού;
12. Εξηγήστε την έννοια της αφαιρετικότητας (abstraction) στον προγραμματισμό;
13. Τι ονομάζουμε εμβέλεια μεταβλητών;
14. Ποιους τύπους εμβέλειας μεταβλητών συναντάμε στη γλώσσα Python;
15. Τι σημαίνει η σειρά L→E→G→B στην εμβέλεια μεταβλητών;

4.12 Ασκήσεις

Να λυθούν οι δραστηριότητες:

4.4.2

4.4.3

4.5.2

4.5.5

4.5.6

4.5.7

4.6.1

4.6.2

4.6.3

4.6.4

4.8.1

Κεφάλαιο 5

5. Υλοποίηση Βασικών προγραμματιστικών Δομών

Τελειώνοντας αυτό το μάθημα θα έχεις μάθει

- Να σχηματίζεις λογικές εκφράσεις
- Να αποτιμάς λογικές εκφράσεις
- Να διατυπώνεις τις μορφές των τριών βασικών αλγοριθμικών δομών.
- Να διατυπώνεις τις μορφές της δομής επιλογής (if)
- Να επιλέγεις τη σωστή δομής επιλογής για την επίλυση ενός συγκεκριμένου προβλήματος
- Να διατυπώνεις τις μορφές δομής επανάληψης
- Να επιλέγεις τη σωστή δομή επανάληψης ανάλογα με τις ανάγκες του προβλήματος.
- Να χρησιμοποιείς τις εντολές break και continue για τη διαφοροποίηση εκτέλεσης των δομών επανάληψης.

5.1 Εισαγωγή

Στο κεφάλαιο αυτό θα γνωρίσουμε τις αλγοριθμικές δομές της Python, η οποία θα οδηγήσει σε πιο σύνθετα προγράμματα. Ένα πρόγραμμα λαμβάνει δεδομένα ως είσοδο, τα επεξεργάζεται σύμφωνα με τις δομές και τις εντολές που περιέχει και στη τέλος εξάγει τα αποτελέσματα.

Η είσοδος των δεδομένων γίνεται συνήθως από το πληκτρολόγιο (η από το ποντίκι σε γραφικά περιβάλλοντα) και από αποθηκευμένα αρχεία. Σε προηγούμενα κεφάλαια έχει αναφερθεί η συνάρτηση `input(),` η οποία διαβάζει δεδομένα από το πληκτρολόγιο.

Η επεξεργασία των δεδομένων γίνεται από εντολές και δομές που καθορίζουν τη σειρά με την οποία θα γίνει η επεξεργασία (ροή εκτέλεσης). Η ροή εκτέλεσης επηρεάζεται από τρείς βασικές αλγοριθμικές δομές, τη δομή ακολουθίας, τη δομή επιλογής και τη δομή επανάληψης που θα παρουσιαστούν στο κεφάλαιο αυτό.

Η έξοδος εμφανίζει η τυπώνει τα τελικά αποτελέσματα. Σε προηγούμενο κεφάλαιο γνωρίσαμε τη βασική εντολή εμφάνισης αποτελεσμάτων στην οθόνη, την `print().`

5.2 Λογικές εκφράσεις ή λογικές συνθήκες

Λογική έκφραση ή λογική συνθήκη είναι μια έκφρασή η οποία μπορεί να βρεθεί η τιμή της (μπορεί να αποτιμηθεί) και η τιμή αυτή να είναι **Αληθής** ή **Ψευδής**, δηλαδή να είναι μια από τις δύο λογικές τιμές.

Ένα απλό παράδειγμα είναι η μαθηματική έκφραση x = 3 (στην Python x == 3). Η έκφραση αυτή είναι **Αληθής** αν πράγματι η μεταβλητή x έχει την τιμή 3 (οπότε η έκφραση είναι 3 =3) ή είναι **Ψευδής** αν η μεταβλητή x έχει μια άλλη τιμή (αν η μεταβλητή χ έχει τιμή 5 τότε η έκφρασή είναι 5 = 3 και φυσικά είναι Ψευδής)

Μια λογική έκφραση μπορεί να είναι σύνθετη και να αποτελείται από σταθερές, μεταβλητές, αριθμητικές παραστάσεις με κάθε είδους τελεστές, συγκριτικούς τελεστές,, λογικούς τελεστές (AND, OR) καθώς και παρενθέσεις.

Η εύρεση της τελικής τιμής μιας σύνθετης λογικής έκφρασης γίνεται σταδιακά στα παρακάτω βήματα:

1. Αντικαθιστούμε τις τιμές των μεταβλητών
2. Πραγματοποιούμε τις αριθμητικές πράξεις ακολουθώντας την προτεραιότητα των τελεστών και την παρουσία παρενθέσεων.
3. Υπολογίζουμε τη λογική τιμή των εκφράσεων που περιέχουν συγκριτικούς τελεστές.
4. Υπολογίζουμε τη λογική τιμή των εκφράσεων που περιέχουν λογικούς τελεστές.

Θυμίζουμε: Οι αριθμητικοί τελετές εκτελούνται πρώτοι ακολουθούν οι συγκριτικοί τελεστές και τέλος οι λογικοί τελεστές

Οι λογικές εκφράσεις που χρησιμοποιούνται σε προγραμματιστικές δομές ονομάζονται λογικές συνθήκες ή απλά συνθήκες.

Παράδειγμα: Η φυσιολογική θερμοκρασία λειτουργίας μια συσκευής είναι από 5 βαθμούς μέχρι 40 βαθμούς Κελσίου. Η σύνθετη λογική συνθήκη ελέγχου της στην Python είναι

```
if therm > 5 and therm < 40:
```

5.3 Αλγοριθμικές δομές

Στον δομημένο προγραμματισμό υπάρχουν τρείς κύριες αλγοριθμικές δομές

- Η δομή ακολουθίας
- Η δομή επιλογής
- Η δομή επανάληψης

Νεότερες τεχνικές προγραμματισμού όπως ο αντικειμενοστραφής προγραμματισμός έχουν εισαγάγει νέες δομές όπως τα αντικείμενα ή έχουν εισάγει τροποποιήσεις στη λειτουργία αυτών των δομών. Αλλά σε κάθε περίπτωση οι αλγοριθμικές δομές εξακολουθούν να χρησιμοποιούνται σε κάθε γλώσσα προγραμματισμού.

5.4 Δομή ακολουθίας

Μια εταιρία burger προσφέρει στους πελάτες που παραγγέλνουν σπέσιαλ burger σε κάθε τρία burger ένα επιπλέον δώρο. Για παράδειγμα αν είμαστε 4 άτομα και παραγγείλουμε 4 burger θα πληρώσουμε τα 3 γιατί το τέταρτο θα είναι δώρο. Να γράψουμε ένα πρόγραμμα το οποίο θα διαβάζει τον αριθμό των burger που θα παραγγείλουμε και την τιμή του και θα υπολογίζει το κόστος της παραγγελίας.

```
# Πρόγραμμα παραγγελίας burger
burger = input("Πόσα Burger θέλετε να παραγγείλετε; ")
timi_enos = input("πόσο κοστίζει το ένα; ")
free_burger = burger //3      # Πόσες τριάδες burger υπάρχουν
kostos = (burger - free-burger)* timi_enos
print ("Κόστος παραγγελίας ", kostos
```

Παρατηρούμε ότι η λύση στο παραπάνω πρόβλημα γίνεται μια σειρά εντολών που το χαρακτηριστικό της εκτέλεσής τους είναι ότι εκτελούνται στη σειρά η μια μετά την άλλη χωρίς να παραλείπεται καμιά.

Αυτή η σειρά εκτέλεσης ονομάζεται δομή ακολουθίας. Η δομή αυτή ουσιαστικά χρησιμοποιείται για την επίλυση απλών προβλημάτων. Τα χαρακτηριστικά της δομής είναι:

- Η σειρά των εντολών (βημάτων) του προγράμματος είναι καθορισμένη.
- Όλες οι εντολές (βήματα) εκτελούνται σειριακά από την αρχή μέχρι το τέλος.

5.5 Δομές επιλογής

Η δομή επιλογής προσφέρει τη δυνατότητα να εκτελούνται (ή να μην εκτελούνται) τμήματα εντολών ανάλογα με το αποτέλεσμα μιας συνθήκης (ενός ελέγχου). Η δομή αυτή χρησιμοποιείται όταν το πρόβλημα που λύνουμε έχει διαφορετικές λύσεις που εξαρτώνται από διάφορα κριτήρια.

Υπάρχουν τρείς κύριες μορφές της δομής επιλογής

- Η απλή δομή `if`
- Η σύνθετη δομή `if … else`
- Η πολλαπλή επιλογή `if .. elif … else`

5.5.1 Η απλή δομή if

Η δομή αυτή χρησιμοποιείται αν θέλουμε κάποιο τμήμα εντολών να εκτελείται ή να μην εκτελείται ανάλογα με το αποτέλεσμα μίας συνθήκης.

Για παράδειγμα αν η θερμοκρασία του χώρου είναι πάνω από 30 βαθμούς κελσίου τότε άναψε τον κλιματισμό. Αυτός το τμήμα αλγορίθμου σε πολύ απλοποιημένη μορφή στην python είναι:

```
#Πρόγραμμα θερμοκρασία
therm = input("Δωσε θερμοκρασία περιβάλλοντος ")
if therm > 30:
    print ("Άνοιξε το κλιματιστικό")
```

Τι θα γίνει αν η θερμοκρασία είναι χαμηλότερη από 30 βαθμούς; Η απάντηση είναι ότι το τμήμα αυτό του αλγορίθμου δεν περιλαμβάνει τις άλλες εντολές που ακολουθούν τη δομή επιλογής και έτσι δεν μπορούμε να απαντήσουμε. Το σίγουρο είναι ότι δεν θα εμφανιστεί το μήνυμα "Άνοιξε το κλιματιστικό".

Σημείωση: Προσέξτε στη γραμμή 4 ότι η εντολή έχει ορισμένα κενά στην αρχή της που δημιουργούν μια εσοχή. Συγκεκριμένα η εσοχή είναι 4 κενοί χαρακτήρες. Οι εσοχές είναι σημαντικές στην python γιατί έτσι ορίζεται το τμήμα (block) εντολών της δομής `if`.

Αν μια εντολή if τοποθετηθεί μέσα σε μια άλλη εντολή if τότε το δεύτεροι μπλοκ εντολών έχουν μια επιπλέον εσοχή 4 χαρακτήρων, το τρίτο μπλοκ θα έχει εσοχή επιπλέον 4 χαρακτήρες κλπ.

Παράδειγμα 5.1

Θέλουμε να γράψουμε ένα τμήμα προγράμματος το οποίο να διαβάζει έναν αριθμό και αν αυτός είναι αρνητικός να τον μετατρέπει σε θετικό και να τον εμφανίζει. Αυτό στα μαθηματικά ονομάζεται εύρεση της απόλυτης τιμής του αριθμού.

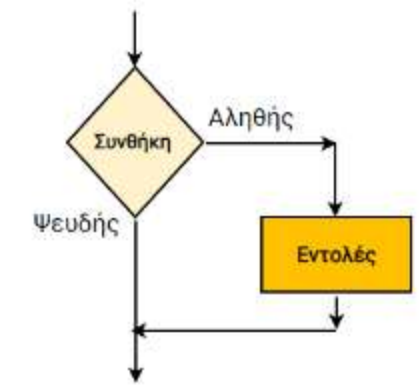
```
# Υπολογισμός απόλυτης τιμής
x = input("Δώσε έναν αριθμό ")
```

```
if x <0:
    x = -1 * x
print ("Η απόλυτη τιμή του αριθμού είναι ", x)
```

Η εντολή στη γραμμή 4: x = -1 * x, θα εκτελεστεί μόνο αν η συνθήκη x < 0 της γραμμής 3 είναι Αληθής . Σε κάθε περίπτωση στη συνέχεια θα εκτελεστεί η εντολή print ("Η απόλυτη τιμή του αριθμού είναι ", x) της γραμμής 5.

Η σύνταξη της εντολής if είναι:

```
if  Συνθήκη:
    Εντολές
```



Η λειτουργία της είναι: Αν η συνθήκη είναι αληθής τότε το σύνολο των εντολών που περιέχονται στην δομή if θα εκτελεστούν, αλλιώς η ροή του προγράμματος θα προσπεράσει όλη τη δομή if και θα συνεχίσει στην επόμενη εντολή.

5.5.2 Σύνθετη δομή if (if ...else)

Η σύνθετη εντολή if, επιπλέον από την λειτουργία της απλής εντολής if, καθορίζει τι θα γίνει (ποιες εντολές θα εκτελεστούν) εάν δεν ισχύει η συνθήκη ελέγχου.

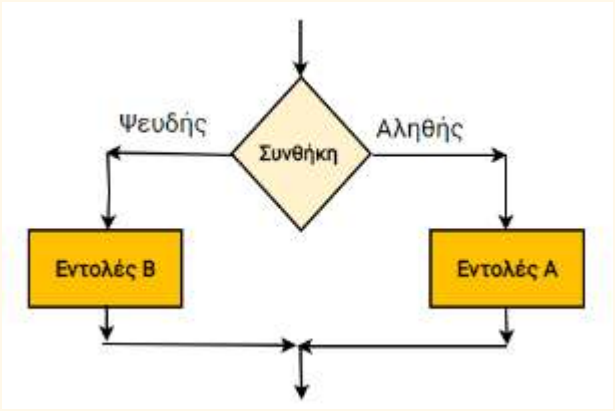
Παράδειγμα 5.2

Σε μια εξέταση ενός πτυχίου ξένης γλώσσας οι μαθητές εξετάζονται προφορικά και γραπτά και βαθμολογούνται με ακέραια βαθμολογία στην κλίμακα 0-20. Αν ο μέσος όρος της βαθμολογίας ενός μαθητή είναι από 10 και πάνω ο μαθητής έχει επιτύχει στην εξέταση, διαφορετικά έχει αποτύχει. Να γραφεί τμήμα προγράμματος το οποίο να διαβάζει για ένα μαθητή τον προφορικό και γραπτό βαθμό του και να εμφανίζει κατάλληλο μήνυμα αν ο μαθητής έχει επιτύχει ή έχει αποτύχει. Αν ο μαθητής έχει επιτύχει θα εμφανίζεται και η βαθμολογία του.

```
# Αποτέλεσμα εξέτασης
prof = int(input("Δώστε τον προφορικό βαθμό "))
grapto = int(input("Δώστε τον γραπτό βαθμό "))
bath = (prof + grapto)/2
# Σύνθετη δομή If
if bath >= 10:
    print("Ο μαθητής επέτυχε με βαθμό ", bath)
else:
    print("Ο μαθητής απέτυχε")
```

Η σύνταξη της σύνθετης εντολής if είναι:

```
if  Συνθήκη_ελέγχου:
    Εντολές_Α   #  αν ισχύει η συνθήκη
else:
    Εντολές_Β  # αν δεν ισχύει η συνθήκη
```



Παρατηρούμε ότι στη δομή `if` έχει προστεθεί ένα ακόμα τμήμα, το τμήμα `else`. Στο τμήμα αυτό γράφονται οι εντολές που θα εκτελεστούν όταν η Συνθήκη_ελέγχου θα είναι ψευδής (δεν θα ισχύει).

Δηλαδή στη δομή αυτή υπάρχουν δύο τμήματα εντολών από τα οποία ανάλογα με την τιμή της συνθήκης ελέγχου θα εκτελεστεί μόνο το ένα.

5.5.3 Πολλαπλή if (if...elif...elif..else)

Η πολλαπλή εντολή `if` αυξάνει την δυνατότητα της σύνθετης εντολής if και δίνει την δυνατότητα να οριστούν ενέργειες για περισσότερες από μια συνθήκες.

Παράδειγμα 5.3

Κατάστημα για ένα συγκεκριμένο προϊόν του εφαρμόζει την παρακάτω πολιτική εκπτώσεων. Για παραγγελίες από 100 τεμάχια και πάνω η έκπτωση τιμής είναι 20%, για παραγγελίες από 50 τεμάχια και πάνω (και μέχρι 100) η έκπτωση είναι 10%, για παραγγελίες από 20 τεμάχια και πάνω (και μέχρι 50) η έκπτωση είναι 5%, ενώ για παραγγελίες κάτω των 20 δεν υπάρχει έκπτωση. Να γραφεί πρόγραμμα το οποίο να διαβάζει το πλήθος τεμαχίων της παραγγελίας, την τιμή ενός τεμαχίου και να εμφανίζει το τελικό κόστος της παραγγελίας

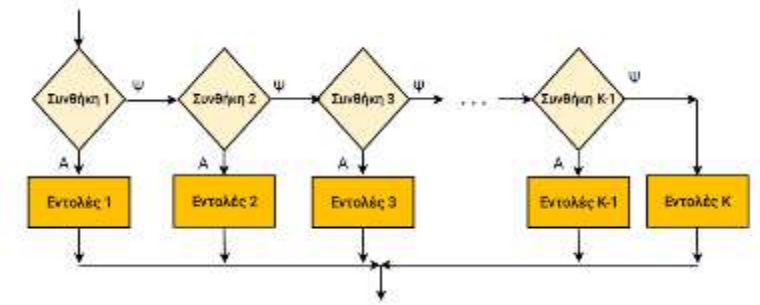
```
# Πρόγραμμα υπολογισμού έκπτωσης - τελικής τιμής

amount = int(input("Αριθμός τεμαχίων: "))
price_pp = float(input("Αρχική τιμή τεμαχίου: "))
orig_cost = amount * price_pp

if  amount >= 100:
    discountp = 20
elif amount >= 50:
    discountp = 10
elif amount >= 20:
```

```
    discountp = 5
else:
    discount = 0

# Υπολογισμός έκπτωσης
discount = (discountp / 100) * orig_cost
# Τελικό κόστος
final_cost = orig_cost - discount

print ("Τελικό κόστος: ", orig_cost)
```

Παρατηρούμε σε αυτό το παράδειγμα η δομή `if` περιέχει 4 τμήματα εντολών. Η συνθήκη ελέγχου που υπάρχει ένα τμήμα `elif: (else-if)` εξετάζεται όταν όλες οι προηγούμενες συνθήκες της δομής είναι ψευδείς. Αν η συνθήκη της `elif:` είναι αληθής εκτελούνται οι εντολές που περιέχει. Και η ροή εκτέλεσης μεταφέρεται στο τέλος της δομής. Το τμήμα `else` εκτελείται μόνο όταν όλες οι προηγούμενες συνθήκες της δομής είναι ψευδείς.

Η σύνταξη της πολλαπλής δομή if είναι

```
if  συνθήκη_ελέγχου_1:
    Εντολές_1
elif συνθήκη_ελέγχου_2:
    Εντολές_2
…..
else:
    Εντολές_κ
```



Τυπικά δεν υπάρχει όριο στο πλήθος των περιπτώσεων `elif` που μπορούν να εισαχθούν σε μια δομή `if`.

5.5.4 Εμφωλευμένα if (nested if)

Παράδειγμα 5.4

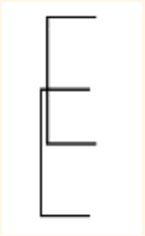
Να γραφεί ένα πρόγραμμα το όποιο να ελέγχει αν ένας ακέραιος αριθμός είναι το μηδέν, είναι αρνητικός ή θετικός και επιπλέον να ελέγχει αν αυτός είναι μονοψήφιος ή πολυψήφιος.

```
# Πρόγραμμα έλεγχος αριθμού
```

```
a = int(input('Δώσε έναν αριθμό: '))
if a > 0:
    if a < 10:
        print ('Θετικός και μονοψήφιος')
    else:
        print ('Θετικός και πολυψήφιος')
elif a < 0:
    if a > -10:
        print ('Αρνητικός και μονοψήφιος')
    else:
        print ('Αρνητικός και πολυψήφιος')
else:
    print ('Είναι το μηδέν')
```

Παρατηρήστε τις εσοχές που καθορίζουν τις διαφορετικές δομές και την ενσωμάτωση των δομών. Ένα λάθος στις εσοχές μπορεί να οδηγήσει σε συντακτικά λάθη ή απρόσμενα αποτελέσματα.

Στην εμφώλευση δομών επιλογής μια δομή μπορεί να καλύπτει μόνο ένα τμήμα (μπλοκ) εντολών μιας άλλης δομής. Δεν επιτρέπεται η επικάλυψη διαφορετικών τμημάτων.



Σχηματικά δεν επιτρέπεται δομές να έχουν τη μορφή του διπλανού σχήματος (το σχήμα "[" αντιπροσωπεύει την αρχή και το τέλος μια δομής)

5.5.5 Δραστηριότητα

Να βρεθεί ο μέγιστος από τρείς αριθμούς a, b, c

Προσέγγιση 1η : Με λογικές συνθήκες του τύπου

```
if  a >b  and a > c :
    print ("Μεγαλύτερος είναι ο a")
```

ελέγχουμε όλους τους αριθμούς και βρίσκουμε τον μεγαλύτερο.

Προσέγγιση 2η :

Χρησιμοποιούμε μια βοηθητική μεταβλητή max η οποία στην αρχή παίρνει την τιμή του a. Στη συνέχεια συγκρίνουμε τη max με το b και αν το b είναι μεγαλύτερος ο max παίρνει την τιμή του b. Τέλος συγκρίνεται η max με το c και αν το c είναι μεγαλύτερος ο max παίρνει την τιμή του c.

Μετά τη διαδικασία αυτή η max περιέχει τη μέγιστη τιμή.

Συζήτηση: Να γίνει συζήτηση σχετικά με το ποια προσέγγιση είναι κατάλληλη αν θα πρέπει να βρεθεί η μέγιστη τιμή από περισσότερες μεταβλητές.

5.6 Δομές επανάληψης (βρόχοι - loops)

Συχνά σε μια υπολογιστική διαδικασία πρέπει ορισμένες ενέργειες να γίνονται ξανά και ξανά. Δηλαδή να επαναλαμβάνονται. Όπως για παράδειγμα:

- Ο έλεγχος της θερμοκρασίας ενός μηχανήματος πρέπει να γίνεται συνεχώς κάθε 1 δευτερόλεπτο για όλη τη διάρκεια λειτουργίας του.
- Ο έλεγχος των φαναριών κυκλοφορίας ενός κόμβου (κόκκινο, πράσινο, πορτοκαλί).
- Η επεξεργασία των στοιχείων 5.000 υποψηφίων για ένα διαγωνισμό του Δημοσίου.

Είναι προφανές ότι αυτές οι ενέργειες στον προγραμματισμό πρέπει να ομαδοποιηθούν και να αντιμετωπιστούν συνολικά και όχι κάθε μια χωριστά.

Αυτό υλοποιούν οι δομές επανάληψης.

5.6.1 Δομή επανάληψης ορισμένου πλήθους επαναλήψεων (for)

Να επικεντρωθούμε περισσότερο στο παράδειγμα δημιουργίας ενός προγράμματος για την επεξεργασία των στοιχείων 5.000 υποψηφίων για ένα διαγωνισμό του δημοσίου. Είναι προφανές ότι δεν είναι λογικό να γράψουμε 5.000 εντολές για να διαβάσουμε τα στοιχεία κάθε υποψήφιου/ας χωριστά..

Η Python για εργασίες στις οποίες είναι καθορισμένος και γνωστός από την αρχή ο αριθμός των επαναλήψεων διαθέτει την δομή `for`.

Παράδειγμα 5.5

```
#Loop1
L = ['Μαρια', 'Κώστας', 'Γιάννης']
for item in L:
    print(item)
Με αποτέλεσμα
Μαρία
Κώστας
Γιάννης
```

```
#Loop2
L = [2, 4, 6, 8, 10]
for item in L:
    print(item)
Με αποτέλεσμα
2
4
6
8
10
```

Παρατηρούμε ότι στα δύο ανωτέρω παραδείγματα χρησιμοποιείται μια λίστα η οποία την πρώτη φορά περιέχει τρία ονόματα (L = ['Μαρια', 'Κώστας', 'Γιάννης']) και τη δεύτερη περιέχει πέντε αριθμούς (L = [2, 4, 6, 8, 10]). Στην εντολή `for item in L:` η μεταβλητή item παίρνει διαδοχικά τιμές από τα στοιχεία της λίστας L. Οι τιμές αυτές τυπώνονται από την εντολή `print`.

Η σύνταξη της δομής for είναι

`for` *μεταβλητή* `in` *αντικείμενο* **:**

εντολές

Στα προηγούμενα παραδείγματα το *αντικείμενο* ήταν μια λίστα. Η μεταβλητή παίρνει διαδοχικά τις τιμές που βρίσκονται μέσα στο *αντικείμενο*.

Περισσότερα για τις λίστες θα αναφερθούν στο κεφάλαιο 7.

Η συνάρτηση range( )

Η `range( )` σε παλαιότερες εκδόσεις της Python ήταν μια ενσωματωμένη συνάρτηση μέσω της οποίας μπορούσε να δημιουργηθεί άμεσα μια λίστα. Στις νεότερες εκδόσεις της Python η `range( )` είναι ένα αντικείμενο το οποίο έμμεσα πάλι μπορεί να χρησιμοποιηθεί για την δημιουργία μιας λίστας.

Παράδειγμα 5.6

Έμμεση δημιουργία μιας λίστας,

```
>>> list(range(4))
[0, 1, 2, 3]
>>>
>>> list(range(2,10))
[2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(2,10,2))
[2, 4, 6, 8]
```

Παρατηρούμε ότι:

Όταν η `range( )` περιέχει μόνο έναν ακέραιο αριθμό π.χ `range(n)` τότε η λίστα που παράγεται περιέχει όλους τους ακεραίους αριθμούς από το 0 μέχρι το n (προσοχή το n δεν περιέχεται).

Όταν η `range( )` περιέχει δύο αριθμούς π.χ `range(a, b)`, οι δύο αυτοί αριθμοί έχουν την έννοια (αρχή, τέλος) και η λίστα που παράγεται περιέχει τους αριθμούς από το a μέχρι το b (εκτός του b).

Τέλος όταν η `range( )` περιέχει τρείς αριθμούς π.χ `range(a, b, c)`, αυτοί οι αριθμοί έχουν την έννοια (*αρχή, τέλος, με_βήμα*) και οι αριθμοί που περιέχει η λίστα δεν είναι συνεχόμενοι αλλά αρχίζουν από την *αρχή* και κάθε επόμενος, μέχρι το *τέλος* , υπολογίζεται από τον προηγούμενο προσθέτοντας το *με_βήμα*.

Μέσα στη δομή `for` η συνάρτηση `range( )` μπορεί να χρησιμοποιηθεί χωρίς την `list( )`

Παράδειγμα 5.7

```
#Loop1
for it in range(10):
    print(it)
```

```
#Loop2
for it in range(2,10):
    print(it)
```

```
#Loop3
for it in range(2,10,2):
    print(it)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
2
3
4
5
6
7
8
9
```

```
2
4
6
8
```

Τα παρακάτω δύο μικρά προγράμματα παράγουν το ίδιο αποτέλεσμα:

```
# prog1
 A = [4, 9, 16, 25, 36]
for i in A:
    print(i)

Αποτέλεσμα
4
6
16
25
36
```

```
# prog2
A = [4, 9, 16, 25, 36]
for i in range(5):
    print A(i)

Αποτέλεσμα
4
6
16
25
36
```

5.6.2 Δραστηριότητα

Δοκιμάστε στις εντολές:

`list(range(a))` `list(range(a,b))` και `list(range(a,b,c))`

να χρησιμοποιήσετε ως τιμές των a, b, c αρνητικούς ή να κάνετε μίξη αρνητικών και θετικών ακεραίων, παρατηρήστε τα αποτελέσματα.

5.6.3 Δραστηριότητα

Να γραφεί πρόγραμμα το οποίο να διαβάζει 20 θετικούς ακεραίους αριθμούς και να βρίσκει και να εμφανίζει τον μεγαλύτερο (μέγιστο) από αυτούς.

```
#  Πρόγραμμα μέγιστος


megistos = 0          # Ορίζουμε αυθαίρετα ένα μικρό αριθμό
for it in range(20):
    ar = int(input('Δώσε έναν αριθμό : '))
    if ar > megistos:
        megistos = ar


print('Ο μεγαλύτερος αριθμός είναι ο ', megistos)
```

Παρατήρηση: Στον ορισμό `megistos = 0` ορίζεται ένας αριθμός μικρότερος από τους αριθμούς που θα δοθούν. Θα μπορούσε να είναι το `megistos = -1` ή οποιασδήποτε άλλος αρνητικός αριθμός.

Επέκταση προβλήματος

Να γραφεί πρόγραμμα το οποίο να διαβάζει 20 (οποιοσδήποτε) αριθμούς και να βρίσκει τον μεγαλύτερο.

Το πρόβλημα εδώ είναι στην αρχική τιμή του megistos. Αν οι αριθμοί είναι τυχαίοι δεν μπορεί να τεθεί megistos = 0 γιατί μπορεί όλοι οι αριθμοί να είναι αρνητικοί και μικρότεροι ή ίσοι π.χ του -5, άρα το μέγιστο μπορεί να είναι το -5.

Η λύση στο πρόβλημα αυτό είναι η αρχική τιμή του megistos να είναι μια από τις τιμές που θα δοθούν (συνήθως η πρώτη). Για το λόγο αυτό διαβάζουμε πρώτα ένα αριθμό τον αποδίδουμε στη μεταβλητή megistos και στη συνέχεια διαβάζουμε τους υπόλοιπους 19 και γίνεται η σύγκριση με τη megistos.

```
#  Πρόγραμμα μέγιστος2
ar = float(input('Δώσε έναν αριθμό : '))
megistos = ar
for it in range(19):
    ar = int(input('Δώσε έναν αριθμό : '))
    if ar > megistos:
        megistos = ar
print('Ο μεγαλύτερος αριθμός είναι ο ', megistos)
```

5.6.4 Δομή επανάληψης υπό προϋπόθεση (while)

Σε πραγματικές συνθήκες επαναλήψεων, η ποιο συνηθισμένη περίπτωση είναι να μην είναι γνωστός ο ακριβής αριθμός των επαναλήψεων αλλά να είναι γνωστή μια συνθήκη τερματισμού.

Για παράδειγμα σε μια εκδήλωση πληκτρολογούνται (διαβάζονται) τα ονόματα των ατόμων που επιθυμούν να συμμετέχουν μέχρι να μην υπάρχει άλλος/η ενδιαφερόμενος/η.

Παράδειγμα 5.8

Να γραφεί πρόγραμμα το οποίο να διαβάζει διαδοχικά αριθμούς και υπολογίζει το τρέχον άθροισμα τους τους. Η διαδικασία εισαγωγής αριθμών τελειώνει όταν δοθεί ο αριθμός 0.

```
# While1
x = 1
suma = 0
while x != 0:
    x = float(input('Δώστε αριθμό'))
        suma = suma + x

#Αποτελέσματα
print(suma)
```

Σύνταξη της δομής while

Η σύνταξη της δομής while είναι:

`while` <Συνθήκη>:

Τμήμα_εντολών

Οι εντολές που περιλαμβάνονται στο Τμήμα_εντολών επαναλαμβάνονται όσο η <Συνθήκη> είναι αληθής.

Σε μια δομή while συναντάμε πολύ συχνά δύο δομές εντολών τους Αθροιστές και τους Μετρητές. Όπως είναι φανερό από το όνομά τους ένας αθροιστής προσθέτει μια ποσότητα σε κάθε επανάληψη και στο τέλος περιέχει το συνολικό άθροισμα. Ένας μετρητής μετρά πόσες φορές συνέβη "κάτι" π.χ πόσες επαναλήψεις έγιναν και στο τέλος περιέχει ένα πλήθος.

Αθροιστής

Η αρχικοποίηση της μεταβλητής `suma = 0` πριν την έναρξη της επανάληψης και η εντολή `suma = suma + x` μέσα στην επανάληψη υλοποιούν ένα **αθροιστή**. Δηλαδή μια δομή η οποία θα προσθέτει κάθε φορά τον αριθμό που εισάγεται στη μεταβλητή `suma` και στο τέλος της διαδικασίας η `suma` θα έχει το άθροισμα όλων των αριθμών.

Υπενθυμίζεται ότι η εντολή `suma = suma + x` μπορεί να γραφεί `suma += x`.

Παράδειγμα 5.9

Να γραφεί πρόγραμμα το όποιο να διαβάζει αριθμούς και υπολογίζει το πλήθος των αριθμών που δόθηκαν. Η διαδικασία εισαγωγής αριθμών τελειώνει όταν δοθεί ο αριθμός 0. Το 0 δεν πρέπει να υπολογίζεται στο πλήθος των αριθμών που δόθηκαν.

Λύση (με πρόβλημα)

```
# While2
x = 1
counter = 0

while x != 0:
    x = float(input('dose arithmo'))
    counter = counter + 1

#Αποτελέσματα
print(counter)
```

Η λύση αυτή έχει ένα πρόβλημα. Το πρόβλημα είναι ότι στο πλήθος των αριθμών που δόθηκαν υπολογίζει και το μηδέν.

Υπάρχουν δύο λύσεις :

Η πρώτη είναι απλή αλλά μη επαγγελματική. Από το πλήθος που εμφανίζουμε αφαιρούμε 1, δηλαδή `print(counter -1).`

Η επαγγελματική λύση είναι η διόρθωση να γίνει στο σημείο που εμφανίζεται το λάθος και ότι στην εκτύπωση μειωμένης τιμής του μετρητή. Ο μετρητής πρέπει να αυξάνεται μόνο όταν ο αριθμός που διαβάστηκε δεν είναι το 0.

```
…
while x != 0:
    x = float(input('dose arithmo'))
    if x != 0:
        counter = counter + 1
…
```

Μετρητής

Η αρχικοποίηση της μεταβλητής `counter = 0` πριν την έναρξη της επανάληψης και η εντολή `counter = counter + 1` μέσα στην επανάληψη υλοποιούν ένα **μετρητή**. Δηλαδή μια δομή η

οποία θα προσθέτει +1 για κάθε αριθμό που διαβάζεται στη μεταβλητή `counter` και στο τέλος της διαδικασίας η `counter` θα έχει το πλήθος όλων των αριθμών

5.6.5 Εμφώλευση δομών επανάληψης

Όπως και στις δομές επιλογής και στις δομές επανάληψης επιτρέπεται η εμφώλευσή τους. Δηλαδή η τοποθέτηση μια δομής επανάληψης μέσα σε μια άλλη δομή επανάληψης.

Η εμφώλευση μπορεί να γίνει και στις δύο βασικές δομές επανάληψης ή σε συνδυασμό τους.

Παράδειγμα 5.10

Να γραφεί πρόγραμμα το οποίο να εμφανίζει την προπαίδεια των αριθμών από το 1 μέχρι το 10

```
# Nested loops
#propedia 1-10
for x in range(1, 11):
    for y in range(1,11):
        print(x, "*", y, " = ", x*y)
    print("---------------")
```

Παρατηρήστε ότι η εντολή print βρίσκεται στο τέλος της πρώτης επανάληψης και όχι στην ενσωματωμένη.

Εντολή break

Σε πολλές γλώσσες προγραμματισμού που ακολουθούν τον δομημένο προγραμματισμό δεν επιτρέπεται να διακοπεί η εκτέλεση μιας επανάληψης πριν ολοκληρωθεί. Στην Python δεν συμβαίνει αυτό. Υπάρχει η εντολή brake η οποία προκαλεί τη διακοπή εκτέλεσης των επαναλήψεων και τερματισμό της δομής. βρόχου.

Παράδειγμα 5.11

Παράδειγμα για τη `for`:

```
#Loop5
L = ['Μαρια', 'Κώστας', 'Γιάννης']
for item in L:
    if item == 'Κώστας':
        print('!Break!')
        break
    print(item)
```

```
#Loop6
for item in range(2,10):
    if item == 7:
        print('!Break!')
        break
    print(item)
```

```
Αποτέλεσμα
Μαρια
!Break!
```

```
Αποτέλεσμα
2
3
4
5
6
!Break!
```

Παράδειγμα 5.12

Παραδείγματα για την `while` .

Να βρεθεί ακέραιος που διαιρείται ακριβώς με το 28 και το 34

```
# While5
num = 1
while num >1:
    if num % 3 == 0 and num % 4 === 0:
        break
    num += 1
print('Ο αριθμός είναι το:' num)
```

```
# While6
num = 1
while True:
    if num % 3 == 0 and num % 4 == 0:
        break
    num += 1
print('Ο αριθμός είναι το:' num)
```

Προσέξτε τη συνθήκη `while True:` Αυτή δημιουργεί ένα ατέρμονα βρόχο δηλαδή μια επανάληψη που θα εκτελείται συνέχεια.

Εντολή continue

Η εντολή continue είναι αντίστοιχη της `break` αλλά διακόπτει μόνο την τρέχουσα επανάληψη του βρόχου και δίνει εντολή στο βρόχο να συνεχίσει με την επόμενη επανάληψή του

Παράδειγμα 5.13

```
#Loop6
for item in range(2,10):
    if item == 7:
        print('Continue!')
        continue
```

```
#Περιττοί  από 1 έως 100
num = 1
while num <= 100:
    if num % 2 == 0:
        num += 1
```

```
    print(item)


Αποτέλεσμα
2
3
4
5
6
!Continue!
8
9
```

```
        continue
    print(num)
    num += 1
Αποτέλεσμα
1
3
5
7
0
```

5.7 Να θυμάμαι

- **Λογική έκφραση η λογική συνθήκη**

  Αποτελεί μια έκφραση η οποία μπορεί να αποτιμηθεί ως ΨΕΥΔΗΣ ή ΑΛΗΘΗΣ.

- **Αλγοριθμικές δομές**

Υπάρχουν τρείς κύριες αλγοριθμικές δομές

- Η δομή ακολουθίας στην οποία όλες οι εντολές ενός κώδικα εκτελούνται στη σειρά η μια μετά την άλλη
- Η δομή επιλογής στην οποία υπάρχουν τμήματα του κώδικα τα οποία μπορούν να εκτελεστούν ή να μην εκτελεστούν ανάλογα με την τιμή μιας λογικής συνθήκης.
- Η δομή επανάληψης στην οποία υπάρχει ένα τμήμα του κώδικα το οποίο εκτελείται πολλές φορές.

- **Δομές επιλογής**

Υπάρχουν τρείς κύριες μορφές:

- **Η απλή δομή επιλογής**
  Υπάρχει ένα τμήμα κώδικα το οποίο εκτελείται ή δεν εκτελείται ανάλογα με την τιμή μιας λογικής έκφρασης: Σύνταξη
  If <συνθήκη>:
  Τμήμα_εντολών
- Η σύνθετη δομή επιλογής
  Υπάρχουν δύο τμήματα κώδικα από τα οποία εκτελείται μόνο το ένα ανάλογα με την τιμή μιας λογικής έκφρασης: Σύνταξη
  If <συνθήκη>:
  Τμήμα_εντολών_1
  else:
  Τμήμα_εντολών_2
- Η πολλαπλή δομή επιλογής
  Υπάρχουν πολλά τμήματα κώδικα από τα οποία εκτελείται μόνο το ένα ανάλογα με την τιμή της λογικής έκφρασης που αντιστοιχεί σε κάθεε τμήμα: Σύνταξη
  If <συνθήκη1>:
  Τμήμα_εντολών_1
  elif : <συνθήκη2>:
  Τμήμα_εντολών_2
  . . .
  . . .
  else:
  Τμήμα_εντολών_K

- **Δομές επανάληψης**

Υπάρχουν δύο κύριες μορφές:

- Δομή επανάληψης με καθορισμένο από την αρχή πλήθος επαναλήψεων. Υλοποιείται με τη δομή for: Σύνταξη

  for *μεταβλητή* **in** *αντικείμενο* **:**

  εντολές

  Το αντικείμενο μπορεί να είναι μια λίστα ή μια συνάρτηση range()
- Δομή επανάληψης με μη καθορισμένο από την αρχή πλήθος επαναλήψεων. Υλοποιείται με τη δομή while : Σύνταξη

  while <Συνθήκη>:

  Τμήμα_εντολών

- Εντολή break χρησιμοποιείται για την διακοπή εκτέλεσης των επαναλήψεων.
- Εντολή continue χρησιμοποιείται για διακοπή της τρέχουσας επανάληψης και συνέχεια εκτέλεσης στην επόμενη επανάληψη.

- **Σημειώσεις**

  - Οι εντολές που ανήκουν σε μια δομή επιλογής ή επανάληψης καθορίζονται από την εσοχή τους, δηλαδή τα κενά που υπάρχουν στην αρχή κάθε γραμμής. Οι γραμμές εντολών που έχουν την ίδια εσοχή θεωρείται ότι ανήκουν στο ίδιο μπλοκ εντολών.

  - Στον δομημένο προγραμματισμό μια εντολή καθορισμένου αριθμού επαναλήψεων (for) δεν πρέπει να διακόπτεται με την εντολή break αλλά να τερματίζει κανονικά με το τέλος των διαθέσιμων επιλογών της μεταβλητής που χρησιμοποιεί.

  - Αντίθετα με την παραπάνω πρόταση είναι συχνό φαινόμενο να χρησιμοποιείται στη δομή επανάληψης while η σύνταξη while True: για να άπειρος αριθμός επαναλήψεων και οι επαναλήψεις διακόπτονται με την εντολή break.

5.8 Ερωτήσεις Κατανόησης

1. Στις δομές επιλογής if ... χαρακτηρίστε κάθε μια τις παρακάτω προτάσεις σωστή (Σ) η Λάθος (Λ):

    a. Όταν χρειάζεται να υπάρξει απόφαση με βάση κάποιο κριτήριο, τότε χρησιμοποιείται η δομή της επιλογής.

    b. Μια δομή επιλογής μπορεί να εκτελεστεί πολλές φορές.

    c. Στη δομή σύνθετης επιλογής υπάρχει περίπτωση κάποιες εντολές να μην εκτελεστούν ποτέ.

    d. Χρησιμοποιούμε τη δομή επιλογής όταν θέλουμε μια ομάδα εντολών να εκτελεστεί πολλές φορές

    e. Η δομή της επιλογής περιλαμβάνει τον έλεγχο κάποιας συνθήκης που μπορεί να έχει δύο τιμές (Αληθής ή Ψευδής)

2. Στις δομές επανάληψης χαρακτηρίστε κάθε μια τις παρακάτω προτάσεις σωστή (Σ) η Λάθος (Λ)::

    a. Μια δομή επανάληψης μπορεί αν μην τερματίζεται ποτέ (αέναος βρόχος).

    b. Σε περίπτωση εμφωλευμένων βρόχων, ο εσωτερικός βρόχος πρέπει να περικλείεται ολόκληρος στον εξωτερικό.

    c. Μέσα σε μια δομή επανάληψης δεν μπορεί να περιέχεται μια δομή επιλογής.

    d. Όταν το πλήθος των επαναλήψεων είναι γνωστό, δε μπορεί να χρησιμοποιηθεί η δομή επανάληψης while

    e. Μια δομή επανάληψης μπορεί να μην εκτελεστεί καμία φορά.

3. Πόσες φορές θα εκτελεστεί η εντολή print στο παρακάτω τμήμα προγράμματος.

```
for x in range(10):
    for y in range(10):
        for z in range(10)
            print(x, y, z)
```

5.9 Ασκήσεις

1. (*) Να γραφεί πρόγραμμα σε Python το οποίο να διαβάζει διαδοχικά τρείς αριθμούς και να εμφανίζει το άθροισμά τους το γινόμενό τους και το μέσο όρο τους με αντίσοιχα μηνύματα. (Δομή ακολουθίας).

2. (*) Να γραφεί πρόγραμμα σε Python το οποίο να διαβάζει έναν ακέραιο αριθμό και να εμφανίζει μήνυμα αν αυτός ο αριθμός είναι άρτιος (ζυγός) ή περιττός (μονός).

3. (**) Ένα εμπορικό κατάστημα ΑΒΓ κάνει σε πελάτες που διαθέτουν την εκπτωτική κάρτα του έκπτωση 5%. Να γραφεί πρόγραμμα σε Python το οποίο για κάθε πελάτη ρωτάει "Διαθέτετε κάρτα ΑΒΓ; (Ν/Ο)" και διαβάζει την απάντηση. Στη συνέχεια να διαβάζει την τιμή μονάδας ενός προϊόντος και την ποσότητα που θέλει να αγοράσει ο πελάτης. Αν στην ερώτηση κάρτας η απάντηση ήταν "ν" ή "Ν" υπολογίζει το ποσό που πρέπει να πληρώσει ο πελάτης με έκπτωση. Σε διαφορετική απάντηση θα εμφανίζει το ποσό που πρέπει να πληρώσει χωρίς έκπτωση.

4. (**) Γενικεύστε το σενάριο της προηγούμενης άσκησης αλλά για περισσότερα προϊόντα. Δηλαδή όταν διαβάζεται η τιμή μονάδας και η ποσότητα ενός προϊόντος θα υπάρχει ερώτηση που θα ρωτάει "Υπάρχει άλλο προϊόν? (Ν/Ο), αν η απάντηση είναι "ν" ή "Ν" επαναλαμβάνεται της τιμής και της ποσότητας του νέου προϊόντος. Αν η απάντη είανι διαφορετική υπολογίζεται και εμφανίζεται το συνολικό ποσό που πρέπει αν πληρώσει ο πελάτης.

5. (*) (Κλιμακωτή χρέωση1) Μια εταιρία παραγωγής μονάδων μνήμης για την προώθηση των πωλήσεων της εφαρμόζει την εξής πολιτική χρέωσης για ένα USB stick που παράγει. Για ποσότητα αγοράς μέχρι και 10 τεμάχια η τιμή τους είναι 12 ευρώ το ένα. Για ποσότητα πάνω από 10, μέχρι και 50 τεμάχια η τιμή τους είναι 10 ευρώ, ενώ από 50 και πάνω η τιμή τους είναι 8 ευρώ το ένα. Να γραφεί πρόγραμμα σε Python το οποίο να διαβάζει τον αριθμό των τεμαχίων που θέλει να αγοράσει ένας πελάτης και να εμφανίζει το ποσό που πρέπει να πληρώσει.

6. (**) (κλιμακωτή χρέωση2) Μια εταιρία παραγωγής μονάδων μνήμης για την προώθηση των πωλήσεων της εφαρμόζει την εξής πολιτική χρέωσης για ένα USB stick που παράγει. Για τα πρώτα 10 τεμάχια αγοράς η τιμή τους είναι 12 ευρώ το ένα. Για τα τεμάχια πάνω από 10, μέχρι και 50 η τιμή τους είναι 10 ευρώ, ενώ για τα τεμάχια από 50 και πάνω η τιμή τους είναι 8 ευρώ το ένα. Να γραφεί πρόγραμμα σε Python το οποίο να διαβάζει τον αριθμό των τεμαχίων που θέλει να αγοράσει ένας πελάτης και να εμφανίζει το ποσό που πρέπει να πληρώσει.

7. (*) Να γραφούν τμήματα προγραμμάτων σε Python τα οποία με χρήση της δομής επανάληψης while να:
    a. Διαβάζει και εμφανίζει ονόματα μέχρι ως όνομα να διαβαστεί η λέξη "ΤΕΛΟΣ".
    b. Διαβάζει έναν αριθμό (Α) και εμφανίζει όλου τους περιττούς αριθμός από το 1 μέχρι τον αριθμό Α.
    c. Διαβάζει έναν αριθμό Α και ελέγχει αν είναι άρτιος και θετικός. Αν είναι άρτιος και θετικός εμφανίζει όλους τους άρτιους αριθμούς από το 0 μέχρι και το Α. Αν δεν είναι άρτιος ή δεν είναι θετικός να εμφανίζει το μήνυμα "Λάθος εισαγωγή αριθμού¨

8. (**) Σε ένα μετεωρολογικό σταθμό καταγράφονται οι υψηλότερες και οι χαμηλότερες τιμές θερμοκρασίας κάθε ημέρας για κάθε μήνα. Να γραφεί πρόγραμμα σε Python το οποίο να

διαβάζει τον αριθμό των ημέρων του μήνα καθώς επίσης για κάθε ημέρα του μήνα, τη μέγιστη και την ελάχιστη θερμοκρασία της μέρας. Στη συνέχεια να υπολογίζει και να εμφανίζει

α) Τη μέγιστη θερμοκρασία του μήνα καθώς και πόσες μέρες εμφανίστηκε αυτή.

β) Την ελάχιστη θερμοκρασία του μήνα καθώς και πόσες μέρες εμφανίστηκε αυτή.

Γ) Τη μέση θερμοκρασία του μήνα (όλων των ημερών). Υπόδειξη να υπολογιστεί πρώτα η μέση θερμοκρασία κάθε ημέρας του μήνα.

9. (***) Στο σταθμό διοδίων της Θήβας τα οχήματα που διέρχονται θα πρέπει να πληρώσουν διόδια. Τα οχήματα χωρίζονται σε τρείς κατηγορίες τα μεγάλα (Μ) οχήματα (φορτηγά , λεωφορεία) που πληρώνουν διόδια 8 ευρώ, τα επιβατικά (Ε) οχήματα που πληρώνουν διόδια 4 ευρώ και τα δίκυκλα (Δ) που πληρώνουν διόδια 2 ευρώ. Να γραφεί πρόγραμμα σε Python το οποίο για μια ημέρα λειτουργίας του σταθμού διοδίων:

   a. Να διαβάζει την κατηγορία του οχήματος (Μ-μεγάλο όχημα, Ε-Επιβατικό, Δ-Δίκυκλο) και να εμφανίζει το αντίτιμο που θα πρέπει να πληρώσει.

   b. Η διαδικασία σταματάει όταν ως κατηγορία του οχήματος δοθεί η λέξη "ΤΕΛΟΣ".

   c. Να υπολογίζει και να εμφανίζει για την ημέρα αυτή:
      i. Τον συνολικό αριθμό οχημάτων για κάθε κατηγορία οχήματος.
      ii. Το ποσοστό των μεγάλων οχημάτων, των επιβατικών και των δικύκλων.
      iii. Το συνολικό ποσό είσπραξης από τη διέλευση των οχημάτων για την ημέρα αυτή.

10. (***) Σε ένα διαγωνισμό του Δημοσίου οι διαγωνιζόμενοι εξετάζονται σε τρείς θεματικές ενότητες. Κάθε ενότητα βαθμολογείται από το 1 έως το 100. Ένας υποψήφιος θεωρείται επιτυχών αν σε κάθε ενότητα η βαθμολογία του είναι από 50 και πάνω και ο μέσος όρος των τριών βαθμών του στις τρείς ενότητες είναι από 60 και πάνω. Να γραφεί πρόγραμμα σε Python το οποίο για κάθε υποψήφιο:
    a. Να διαβάζει το όνομα και τους βαθμούς του στις τρείς ενότητες.
    b. Να εξετάζει αν είναι επιτυχών ή όχι και να εμφανίζει αντίστοιχα μηνύματα ως εξής: Αν είναι επιτυχόν το μήνυμα "Επιτυχών με βαθμό " και δίπλα το μέσο όρο της βαθμολογίας του. Αν είναι αποτυχών μόνο το μήνυμα "Αποτυχών".
    c. Η διαδικασία εισαγωγής να τερματίζεται όταν ως όνομα υποψηφίου δοθεί η λέξη "ΤΕΛΟΣ".
    d. Στη συνέχεια θα εμφανίζεται το όνομα του επιτυχόντος υποψηφίου με τη μεγαλύτερη βαθμολογία και δίπλα η βαθμολογία του. Επίσης θα εμφανίζεται το όνομα του επιτυχόντος υποψήφιου που πέτυχε με τη μικρότερη βαθμολογία. (θεωρήστε ότι τα άτομα αυτά υπάρχουν και είναι μοναδικά).

Κεφάλαιο 6

6. Δομή Συμβολοσειράς, Αντικείμενα, Αναζήτηση

Τελειώνοντας αυτό το μάθημα θα έχεις μάθει

- Τη χρησιμότητα των συμβολοσειρών
- Τι είναι Δομή Δεδομένων
- Τα βασικά χαρακτηριστικά της συμβολοσειράς ως Δομή Δεδομένων
- Τι είναι τα αντικείμενα και τα βασικά χαρακτηριστικά τους
- Να χειρίζεσαι την εσωτερική δομή των συμβολοσειρών
- Να διασχίζεις τις συμβολοσειρές και να κάνεις πράξεις με τους τελεστές τους
- Τον τρόπο κωδικοποίησης των συμβολοσειρών και τις συναφείς συναρτήσεις
- Πως εκφράζεται προγραμματιστικά η λειτουργικότητα των αντικειμένων
- Να χρησιμοποιείς πολλές από τις μεθόδους των συμβολοσειρών
- Τον αλγόριθμο σειριακής αναζήτησης και τις παραλλαγές του
- Τις τεχνικές ιεραρχικής σχεδίασης και του τμηματικού προγραμματισμού
- Να επιλύεις βασικά προβλήματα που αφορούν σε επεξεργασία κειμένου

6.1 Οι συμβολοσειρές

Η επεξεργασία και ο χειρισμός δεδομένων που έχουν τη μορφή κειμένου είναι από τις πιο συχνές λειτουργίες για τις οποίες χρησιμοποιούνται εφαρμογές πληροφορικής. Τεράστιος όγκος κειμένων με μορφή απλών εγγράφων κειμένου ή περιεχομένου: ηλεκτρονικής αλληλογραφίας, άμεσων μηνυμάτων, αναρτήσεων κοινωνικής δικτύωσης, ιστοσελίδων, βάσεων δεδομένων, και πολλών άλλων εφαρμογών. Το περιεχόμενο αυτό πριν ή μετά την αποθήκευσή του σε κάποιο μέσο υλικού, υπόκεινται σε επεξεργασία που αφορά, είτε στην τροποποίησή του με κάποιους κανόνες, είτε στη μορφοποίησή του, είτε σε αναζήτησή του ανάμεσα σε τεράστιο όγκο δεδομένων με κάποια κριτήρια (λέξεις κλειδιά), είτε σε έλεγχο και ανάλυση του περιεχομένου του με απλούς ή και πιο σύνθετους κανόνες. Μερικά παραδείγματα απλής αλλά και πιο σύνθετης επεξεργασίας κειμένου με χρήση κώδικα:

- Διαχωρισμός από το περιεχόμενο ιστοσελίδων των στοιχείων επικοινωνίας που περιέχει (email, τηλέφωνο).
- Θεματική κατάταξη ενός κειμένου ανάλογα με το περιεχόμενό του και με τη βοήθεια λέξεων κλειδιών (Ενημέρωση, Ψυχαγωγία, Αθλητικά, Οικονομία, Υγεία κλπ.).
- Ορθογραφική ή και συντακτική διόρθωσή του.
- Αισθητικές παρεμβάσεις στον τρόπο εμφάνισής του (πχ στοίχιση, διάταξη κλπ.).
- Αναγνώριση γλώσσας γραφής, ύφους, χρονολογίας, μετάφρασης σε άλλη γλώσσα.

Κάποιες από αυτές τις λειτουργίες ή τμήματα αυτών θα αντιμετωπιστούν και σε αυτό το κεφάλαιο.

Το περιεχόμενο ενός κειμένου μπορεί να αποτελείται από γράμματα πεζά ή κεφαλαία διαφόρων γλωσσών, ψηφία, σύμβολα και διάφορους άλλους χαρακτήρες (πχ χαρακτήρες διαφυγής που αναφέρθηκαν στο 2ο κεφάλαιο). Για το χειρισμό του περιεχομένου κειμένου οι περισσότερες γλώσσες προγραμματισμού διαθέτουν κάποιο τύπο δεδομένων. Η Python χρησιμοποιεί ένα βασικό τύπο δεδομένων που ονομάζεται συμβολοσειρά (string).

Οι συμβολοσειρές στην Python όπως θα αναλυθεί στη συνέχεια διαθέτουν εσωτερική δομή αλλά και ένα μεγάλο πλήθος έτοιμων/ενσωματωμένων (built - in) εργαλείων για τον εύκολο χειρισμό τους και την επεξεργασία του περιεχομένου τους.

Παράδειγμα 6.1

Ακολουθούν μερικά παραδείγματα συμβολοσειρών

```
a_string = 'p'
b_string = 'python'
c_string = 'Beautiful is better than ugly.'
d_string = 'القبيح من خير الجميل'
e_string = 'सुंदर बदसूरत से बेहतर है।'
f_string = '123!@#{}'
```

6.1.1 Δομές Δεδομένων και συμβολοσειρές

Δομή Δεδομένων (Data Structure) στην πληροφορική είναι ένας τρόπος αποθήκευσης και οργάνωσης δεδομένων με τέτοιο τρόπο ώστε να εκτελούνται αποδοτικά διάφορες λειτουργίες σε

αυτά. Στην πληροφορική για τον χειρισμό δεδομένων χρησιμοποιούνται διάφορες δομές δεδομένων και κάθε γλώσσα προγραμματισμού (εκτός από κάποιες κοινές) διαθέτει και τις δικές της. Το ποια ή ποιες δομές δεδομένων θα επιλεχθούν από τον προγραμματιστή στην εφαρμογή του θα επηρεάσει και τις επιτρεπόμενες λειτουργίες, τα εργαλεία και τις τεχνικές που θα μπορεί να χρησιμοποιήσει και τελικά και το πρόγραμμα επίλυσης του προβλήματός του. Έτσι προκύπτει η παρακάτω εξίσωση (σχέση):

Algorithms + Data Structures = Programms

που διατυπώθηκε από τον Niklaus Wirth στο ομώνυμο βιβλίο του για να τονίσει ανάμεσα σε άλλα και τη σημασία των χρησιμοποιούμενων δομών δεδομένων στην επίλυση ενός προβλήματος.

Η γλώσσα Python διαθέτει ενσωματωμένες (Built - in) διάφορες τέτοιες δομές δεδομένων, κάθε μία με τα δικά της χαρακτηριστικά, δυνατότητες και λειτουργίες. Λίστες, Πλειάδες, Λεξικά, Σύνολα (Lists, Tuples, Dictionaries, Sets).

Οι συμβολοσειρές στην Python αποτελούν και αυτές μία δομή δεδομένων.

6.1.2 Γενικά Χαρακτηριστικά των συμβολοσειρών

- Διαθέτουν εσωτερική **ακολουθιακή δομή** (sequence), δηλαδή μπορούν να χρησιμοποιηθούν δείκτες για την αναφορά στα στοιχεία τους (τους χαρακτήρες δηλαδή που τις αποτελούν).
- Είναι **διασχίσιμες** ακολουθίες (iterable sequences) χαρακτήρων κειμένου. Έτσι χρησιμοποιώντας μία δομή επανάληψης μπορεί να γίνει προσπέλαση διαδοχικά όλων των στοιχείων τους ένα προς ένα.
- Είναι **αμετάβλητες** δομές, (μη τροποποιήσιμες/ μη μεταβαλλόμενες: immutable). Δηλαδή δεν μπορούν να τροποποιηθούν οι χαρακτήρες μίας συμβολοσειράς, όπως και να προστεθούν ή να αφαιρεθούν χαρακτήρες από αυτήν (χωρίς να δημιουργηθεί νέα).
- Ο τρόπος χειρισμού των δομών των συμβολοσειρών στην Python έχει σχεδιαστεί και έχει εμπλουτιστεί με χαρακτηριστικά και προγραμματιστικά **εργαλεία** ώστε με τη χρήση τους να επιλύεται ένα μεγάλο πλήθος καθημερινών προβλημάτων.

6.1.3 Εισαγωγή δεδομένων συμβολοσειρών

Αναφέρθηκε σε προηγούμενα κεφάλαια ότι η εισαγωγή δεδομένων με την εντολή εισόδου `input` έχει ως αποτέλεσμα να λαμβάνονται αυτά με τη μορφή κειμένου από τη γλώσσα και στη συνέχεια εάν τα περιεχόμενά τους το επιτρέπουν με κάποια από τις συναρτήσεις μετατροπής τύπων δεδομένων `int` ή `float` να μετατρέπονται πχ σε έναν αριθμό ακεραίου τύπου ή σε έναν αριθμό τύπου κινητής υποδιαστολής. Σε περίπτωση που δεν χρησιμοποιηθεί κάποια τέτοια συνάρτηση μετατροπής δεδομένων τα δεδομένα που θα δώσει ο χρήστης εισάγονται στη μεταβλητή που προηγείται της εντολής `input`, ως συμβολοσειρά (string). Σε περίπτωση τοποθέτησης κειμένου απευθείας σε μεταβλητή αυτό τοποθετείται στη μεταβλητή εντός τόνων (μονών ή διπλών).

Παράδειγμα 6.2

Συντάξτε και εκτελέστε στο κέλυφος τις παρακάτω εντολές για εισαγωγή δεδομένων συμβολοσειράς (τα επεξηγηματικά σχόλια δεν χρειάζεται να γραφούν):

```
# Εισαγωγή δεδομένων συμβολοσειράς
```

```
>>> data = input('Δώστε τα δεδομένα σας :')
    Δώστε τα δεδομένα σας :12345
# Με την εντολή εκτύπωσης print εμφανίζεται μόνο το περιεχόμενο
>>> print(data)
    12345
>>> type(data)
    <class 'str '>
# Με την αναφορά ονόματος εμφανίζονται και οι τόνοι της συμβολοσειράς
>>> data
    '12345'
```

Υπάρχει η δυνατότητα δημιουργίας συμβολοσειρών πολλαπλών γραμμών (multiline string). Η δημιουργία τέτοιων αντικειμένων μπορεί να γίνει απευθείας με εκχώρηση περιεχομένου χρησιμοποιώντας 3πλούς (**διπλούς**) τόνους ή με διάφορους τρόπους με χρήση κώδικα, χρησιμοποιώντας και την εντολή `input`.

Παράδειγμα 6.3

Συντάξτε και εκτελέστε στο συντάκτη τις εντολές του παραδείγματος για εισαγωγή δεδομένων κειμένου πολλαπλών γραμμών:

Σε χρόνο σύνταξης

```
# Εισαγωγή δεδομένων πολλαπλών γραμμών σε χρόνο σύνταξης
text = """
If the implementation
is easy to explain,
it may be a good idea.
"""
```

Σε χρόνο εκτέλεσης

```
# Ενδεικτική εισαγωγή δεδομένων πολλαπλών γραμμών σε χρόνο εκτέλεσης
# Η εισαγωγή σταματά όταν δοθεί γραμμή χωρίς περιεχόμενο και <enter>
text = ''
while True:
    line = input('Εισάγετε μία γραμμή κειμένου :')
    if line == '':
        break
    else:
        text += line + '\n'
```

Το αποτέλεσμα εκτέλεσης του προγράμματος φαίνεται παρακάτω:

```
Εισάγετε μία γραμμή κειμένου :If the implementation
Εισάγετε μία γραμμή κειμένου :is easy to explain,
Εισάγετε μία γραμμή κειμένου :it may be a good idea.
```

```
Εισάγετε μία γραμμή κειμένου :
```

*Εκτελέστε στο κέλυφος την παρακάτω εντολή για να δείτε το **εκτυπώσιμο** περιεχόμενο της συμβολοσειράς πολλαπλών γραμμών:*

```
>>> print(text)
If the implementation
is easy to explain,
it may be a good idea.
```

*Εκτελέστε στο κέλυφος την παρακάτω εντολή για να δείτε το **πραγματικό** περιεχόμενο της συμβολοσειράς πολλαπλών γραμμών:*

```
>>> text
'If the implementation\nis easy to explain,\nit may be a good idea.\n'
```

Παρατηρείστε ότι η συμβολοσειρά σε κάθε αλλαγή γραμμής περιέχει το χαρακτήρα διαφυγής `\n`*, βέβαια και στο προηγούμενο παράδειγμα η συμβολοσειρά περιείχε το χαρακτήρα αλλαγής γραμμής, η συνάρτηση όμως* `print` *αναλάμβανε να εκφράσει(αποτιμήσει: evaluate) το αποτέλεσμά του στην οθόνη και όχι να εμφανίσει αυτούσιο τον ειδικό αυτόν χαρακτήρα.*

6.2 Αντικείμενα

Η φράση Python is an Object Oriented Language, Everything in Python is An Object συναντάται συχνά στο Διαδίκτυο όταν αναζητούνται πληροφορίες για τη γλώσσα αυτή. Δηλαδή: «*Η Python είναι Αντικειμενοστρεφής Γλώσσα Προγραμματισμού*», «*Τα πάντα στην Python είναι αντικείμενα*». Ας θυμίσουμε λίγο πιο αναλυτικά τι είναι τα αντικείμενα στον προγραμματισμό και γιατί κάνουμε αναφορά σε αυτά ξανά εδώ, στο κεφάλαιο των συμβολοσειρών;

Ένα αντικείμενο σε μία γλώσσα προγραμματισμού είναι μία οντότητα η οποία εμπεριέχει **δεδομένα** και **λειτουργικότητα** (συμπεριφορά). Στη γλώσσα Python συγκεκριμένα, όλα τα δεδομένα που χρησιμοποιήθηκαν έως τώρα είτε αφορούσαν σε ακεραίους αριθμούς, είτε σε αριθμούς κινητής υποδιαστολής, είτε σε λογικά δεδομένα, είτε σε συμβολοσειρές, είτε σε συναρτήσεις αποτελούν αντικείμενα (objects).

Κάθε αντικείμενο έχει μία **ταυτότητα** (id), έχει κάποια **τιμή/ κατάσταση** (value/ state) και κάποια **συμπεριφορά** (Behavior) ανάλογα με τις ενέργειες που μπορεί να κάνει. Αποδίδουμε σε αυτό κάποιο όνομα για να είναι εφικτός ο χειρισμός του μέσα από το πρόγραμμα, ανήκει δε σε ένα τύπο/ κλάση/ τάξη (type/ class/ class).

Ένα αντικείμενο είναι μια "οντότητα" που συνδυάζει την κατάσταση (δεδομένα) και τη συμπεριφορά (λειτουργίες) σε ένα πακέτο. Αυτό επιτρέπει την οργάνωση του κώδικα σε πιο δομημένες και αφηρημένες μονάδες, που προσομοιάζουν περισσότερο τον πραγματικό κόσμο.

Για παράδειγμα, έστω ένα αντικείμενο με την ονομασία "αυτοκίνητο". Το αυτοκίνητο μπορεί να έχει δεδομένα όπως το χρώμα, το μοντέλο, την ταχύτητα, απόθεμα καυσίμου κλπ. Μπορούν επίσης να ορισθούν μέθοδοι για το αυτοκίνητο, όπως "επιταχύνει", "φρενάρει", "ανεφοδιάζεται" που θα επηρεάζουν την ταχύτητα του αυτοκινήτου, ή το καύσιμό του.

Οι αντικειμενοστραφείς γλώσσες προγραμματισμού, όπως η Python, επιτρέπουν τη δημιουργία και τη χρήση αντικειμένων για να οργανώσουν τον κώδικα και να αναπαραστήσουν τον πραγματικό κόσμο με πιο αφηρημένο τρόπο. Αυτή η προσέγγιση ονομάζεται αντικειμενοστρεφής προγραμματισμός (object-oriented programming) και παρέχει πλεονεκτήματα όπως την ανακατανομή του κώδικα, την επαναχρησιμοποίηση του, τη βελτίωση της αφαιρετικότητας (abstraction) και την αναλογική αντιμετώπιση προβλημάτων.

Στην Python, ο χρήστης μπορεί να σχεδιάσει και να δημιουργήσει τα δικά του αντικείμενα αλλά και να χρησιμοποιήσει και πολλά αντικείμενα που παρέχονται από τις έτοιμες κλάσεις της γλώσσας και των βιβλιοθηκών της. Αυτά τα ενσωματωμένα αντικείμενα παρέχουν έτοιμες λειτουργίες και μεθόδους που μπορούν να χρησιμοποιηθούν στον κώδικά του χωρίς να χρειάζεται να τα προγραμματίσει από την αρχή.

Στη γλώσσα Python λοιπόν, όλα είναι αντικείμενα. Ακόμη και οι απλές μορφές δεδομένων, όπως οι αριθμοί (ακέραιοι ή κινητής υποδιαστολής) και οι συμβολοσειρές, οι συναρτήσεις είναι αντικείμενα. Επίσης, οι Δομές Δεδομένων (ΔΔ) που θα μελετηθούν στην πορεία όπως οι Λίστες, οι Πλειάδες τα Λεξικά τα Σύνολα είναι αντικείμενα.

6.2.1 Χαρακτηριστικά των Αντικειμένων

Η ταυτότητα ενός αντικειμένου (id) είναι ένας μοναδικός ακέραιος αριθμός ο οποίος αποδίδεται αυτόματα στο αντικείμενο από τη γλώσσα όταν αυτό δημιουργείται με μία εντολή ανάθεσης τιμής `x = 3` ή και χωρίς εντολή ανάθεσης όπως θα δούμε, γράφοντας στο κέλυφος απλά την τιμή του. Η ταυτότητα του αντικειμένου παραμένει ίδια σε όλη τη διάρκεια εκτέλεσης του συγκεκριμένου προγράμματος, το καθορίζει και το διαχωρίζει από τα υπόλοιπα αντικείμενα.

Κατάσταση (State) Η κατάσταση ενός αντικειμένου καθορίζεται από τις τιμές των δεδομένων που περιέχει. Αυτά τα δεδομένα μπορεί να είναι μεταβλητές που αναφέρονται στο αντικείμενο.

Συμπεριφορά/Λειτουργικότητα (Behavior) Η συμπεριφορά ενός αντικειμένου καθορίζεται από τις μεθόδους που σχετίζονται με αυτό. Οι μέθοδοι είναι συναρτήσεις που συνδέονται με ένα αντικείμενο και μπορούν να εκτελεστούν για να αλληλεπιδράσουν με τα δεδομένα του αντικειμένου.

Ο τύπος (κλάση/ τάξη) του αντικειμένου καθορίζεται αυτόματα από τη γλώσσα Python ανάλογα την τιμή που αποδίδεται σε αυτό.

Το όνομα που αποδίδεται στο αντικείμενο που έως τώρα καλούσαμε μεταβλητή είναι στην ουσία μία **αναφορά (reference)** στο αντικείμενο που δημιουργήθηκε για να μπορεί να κληθεί από τον προγραμματιστή μέσα στον κώδικά του.

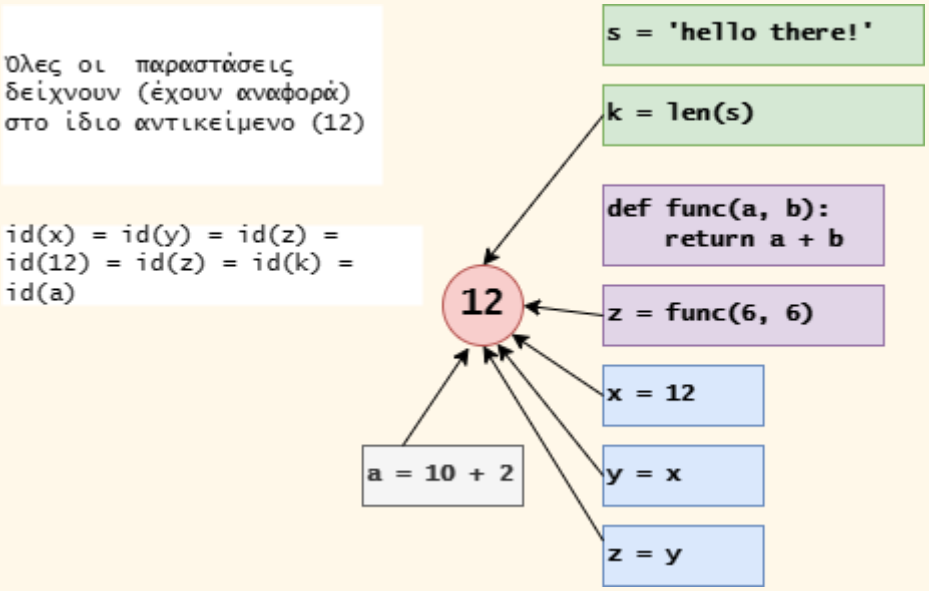
Τα παραπάνω μπορείτε να τα δείτε με αντίστοιχες εντολές από το κέλυφος για όλους τους τύπους δεδομένων που αναφέρθηκαν έως τώρα.

Παράδειγμα 6.4

Συντάξτε και εκτελέστε στο κέλυφος ή στο συντάκτη τις εντολές του παραδείγματος για τις λειτουργίες της ταυτότητας και του ονόματος αναφοράς αντικειμένου:

```
# Δημιουργία αντικειμένου τύπου int με τιμή 12 και αναφορά σε αυτό με
το όνομα της  μεταβλητής x
>>> x = 12
# Εμφάνιση της μοναδικής ταυτότητας του αντικειμένου με τη χρήση της
συνάρτησης id
# ή μέσω της αναφοράς σε αυτό: id(x)
>>> id(x)
140712208233608
# ή μέσω του ίδιου του αντικειμένου: id(12)
>>> id(12)
140712208233608
# Δημιουργία 2ης αναφοράς (y) στο ίδιο αντικείμενο
>>> y = 12
>>> id(12)
140712208233608
# Δημιουργία νέας μεταβλητής (z) στην ίδια αναφορά (y)
>>> z = y
```

```
>>> id(z)
140712208233608
# Παρατηρείστε ότι όλες οι μεταβλητές (αναφορές) x, y, z έχουν το ίδιο
# id 140712208233608 γιατί αναφέρονται στο ίδιο αντικείμενο (12)
```



Παράδειγμα 6.5

Συντάξτε και εκτελέστε στο κέλυφος ή στο συντάκτη τις εντολές του παραδείγματος για τις λειτουργίες εύρεσης τύπων/ κλάσεων αντικειμένων:

```
# Δημιούργησε το αντικείμενο τύπου ακεραίου με τιμή 3 και με αναφορά x
x = 3
# Εμφάνισε την κλάση του αντικειμένου που αναφέρεται η μεταβλητή x
>>> type(x)
<class 'int'>
# Δημιούργησε το αντικείμενο τύπου συμβολοσειράς με τιμή 'Guido' και με
αναφορά σε αυτό name
name = 'Guido '
# Εμφάνισε την κλάση του αντικειμένου που αναφέρεται η μεταβλητή name
>>> type(name)
<class 'str'>
# Εμφάνισε την τιμή του αντικειμένου στο οποίο αναφέρεται η μεταβλητή x
# και η μεταβλητή name
>>> print(x, name)
12, Guido
```

Είναι εφικτή η δημιουργία πληθώρας αντικειμένων από κάθε κλάση των απλών τύπων δεδομένων της Python (`int`, `float`, `str`, `bool`). Επίσης υπάρχει η δυνατότητα δημιουργίας κλάσεων σύμφωνα με τις ανάγκες του χρήστη οι οποίες μπορούν να παράγουν τα αντίστοιχα αντικείμενα στο πρόγραμμά του. Αυτά όμως θα αναλυθούν με λεπτομέρεια σε επόμενο κεφάλαιο.

Σημείωση

Εφόσον παρουσιάστηκαν οι έννοιες αντικείμενο, αναφορά, τύπος. Οι ορολογίες που ακολουθούν για την παρακάτω εντολή είναι αποδεκτές:

x = 3

- *Η μεταβλητή **x** με τιμή 3*
- *Το αντικείμενο (ακεραίου τύπου) με τιμή 3 και όνομα **x***
- *Η αναφορά με όνομα **x** στο αντικείμενο (ακεραίου τύπου) 3*

6.2.2 Δραστηριότητα, ταυτότητα & τύπος αντικειμένων

(*)Στο κέλυφος της γλώσσας Python εκτελέστε τις εντολές και συμπληρώστε στα κενά τα αποτελέσματα των παρακάτω εντολών που εμφανίζουν την ταυτότητα και τους τύπους των αντικειμένων.

Συγκρίνετε τα αποτελέσματα των εντολών στο κέλυφος με αυτά των συμμαθητών σας ή κάντε restart το κέλυφος και εκτελέστε πάλι τις ίδιες εντολές.

α. Ποια αποτελέσματα είναι ίδια και ποια όχι ?
β. Μπορείτε να εξηγήσετε τους λόγους ?

```
>>> x = 12
>>> pi = 3.14
>>> flag = True
>>> name = 'Guido'
>>> id(x)

_________________
>>> id(pi)

_________________
>>> id(flag)

_________________
>>> id(name)

_________________
>>> type(x)

_________________
>>> type(pi)

_________________
>>> type(flag)

_________________
>>> type(name)

_________________
>>> print (x, pi, flag, name)

_________________
```

6.2.3 Αντικείμενα και συναρτήσεις μετατροπής τύπου str, int, float, bool

Εκτός από τη δυνατότητα δημιουργίας ενός αντικειμένου κάποιου συγκεκριμένου τύπου υπάρχει όπως έχει αναφερθεί σε προηγούμενο κεφάλαιο η δυνατότητα μετατροπής από έναν τύπο (κλάση)

σε έναν άλλο (υπό προϋποθέσεις βέβαια). Αυτό γίνεται με τις ενσωματωμένες συναρτήσεις `str()`, `int()`, `float()`, `bool()`, η σύνταξη τους είναι:

```
<συνάρτηση μετατροπής>(<αντικείμενο>)
```

- `str(object)` : Συνάρτηση η οποία μετατρέπει ένα αντικείμενο από έναν άλλο τύπο σε αντικείμενο τύπου συμβολοσειράς

Παράδειγμα 6.6

```
>>> str(123)
'123'
>>> str(12.34)
'12.34'
>>> str(True)
'True'
```

- `int(object)` : Συνάρτηση η οποία μετατρέπει ένα αντικείμενο από έναν άλλο τύπο σε αντικείμενο τύπου ακεραίων

Παράδειγμα 6.7

```
>>> int('123')
123
>>> int(12.34)
12
>>> int(True)
1
>>> int('False')
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    int('False')
ValueError: invalid literal for int() with base 10: 'False'
```

- `float(object)` : Συνάρτηση η οποία μετατρέπει ένα αντικείμενο από έναν άλλο τύπο σε αντικείμενο τύπου κινητής υποδιαστολής

Παράδειγμα 6.8

```
>>> float(123)
123.0
>>> float('12.34')
12.34
>>> float(True)
1.0
>>> float('False')
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    float('False')
ValueError: could not convert string to float: 'False'
```

- `bool(object)` : Συνάρτηση η οποία μετατρέπει ένα αντικείμενο από έναν άλλο τύπο σε αντικείμενο λογικού τύπου

Παράδειγμα 6.9

```
>>> bool(0)
False
>>> bool('something')
True
>>> bool('')
False
>>> bool(' ')
True
```

Σημείωση

Όταν το περιεχόμενο των αντικειμένων των παραπάνω συναρτήσεων δεν είναι συμβατό με τον τύπο μετατροπής επιστρέφεται μήνυμα λάθους.

6.2.4 Δραστηριότητα, συναρτήσεις μετατροπής τύπων

(*)Συντάξτε και συμπληρώστε τα κενά που εμφανίζονται τα αντίστοιχα αποτελέσματα, στη συνέχεια εκτελέστε και ελέγξτε τις απαντήσεις σας:

```
>>> str(true)

_________________
>>> str(False)

_________________
>>> int(123.34)

_________________
>>> int('123.34')

_________________
>>> float(20 / 10)

_________________
>>> float('123')

_________________
>>> bool(1.0)

_________________
>>> bool(0.0)

_________________
>>> str(12 ** 2)

_________________
>>> str(int(12.34))

_________________
>>> int(float('123.34'))

_________________
>>> bool(int(3 / 2))
```

```
_________________
>>> bool(int(2 / 3))

_________________
```

6.3 Η δομή και οι δείκτες της συμβολοσειράς

Οι συμβολοσειρές στην Python διαθέτουν εσωτερική δομή, έτσι κάθε χαρακτήρας τους αντιστοιχεί σε έναν αριθμό (δείκτη) ο οποίος συνάδει με τη θέση του χαρακτήρα εντός της συμβολοσειράς. Η αρίθμηση των δεικτών γίνεται με δύο τρόπους:

- με θετικούς ακεραίους από αριστερά (αρχή), ξεκινώντας από το 0
- με αρνητικούς ακεραίους από δεξιά (τέλος), ξεκινώντας από το -1.

Pos Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
text=	**i**		**l**	**o**	**v**	**e**		**p**	**y**	**t**	**h**	**o**	**n**	**.**
Pos index	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Η προσπέλαση κάποιου χαρακτήρα μπορεί να γίνει άμεσα (όχι βέβαια για να μεταβληθεί στη ίδια συμβολοσειρά) για να εμφανιστεί ή να ελεγχθεί η τιμή του ή για να χρησιμοποιηθεί σε κάποιο υπολογισμό ή για να τοποθετηθεί σε **μία νέα συμβολοσειρά**.

Μήκος της συμβολοσειράς. Το μήκος της συμβολοσειράς μπορεί να υπολογισθεί χρησιμοποιώντας τη συνάρτηση:

- `len(object)`
  - πχ `len('hello')` → `5`

Παράδειγμα 6.10

(*)Συντάξτε στο κέλυφος και εκτελέσετε ώστε να εμφανιστούν τα αποτελέσματα των παρακάτω εντολών που διαχειρίζονται εντολές προσπέλασης χαρακτήρων σε συμβολοσειρές μέσω των δεικτών τους και της συνάρτησης μήκους:

```
>>> text = 'i love python.'
# Έλεγχος εάν ο 14 χαρακτήρας από την αρχή είναι η τελεία
>>> text[13] == '.'
    True
# Εμφάνισε τον 8 χαρακτήρα
# ή το χαρακτήρα στη θέση 7, προσοχή η αρίθμηση αρχίζει από το 0
>>> print(text[7])
    p
# Επιστροφή του τελευταίου χαρακτήρα
>>> text[-1]
'.'
>>> text[len(text) - 1]
'.'
```

6.3.1 Δραστηριότητα, δείκτες συμβολοσειράς

(*)Συντάξτε και συμπληρώστε τα κενά στις παρακάτω εντολές στο κέλυφος ώστε να εμφανιστούν τα αντίστοιχα αποτελέσματα, στη συνέχεια εκτελέστε και ελέγξτε τις απαντήσεις σας:

```
# Συμπληρώστε τα κενά ώστε να σχηματιστεί η λέξη Python
>>> text = 'Although Practicality beats purity'
>>> word = text[_] + text[__] + text[_] + text[_] + text[_] + 'nN'[_]

>>> print(word)
Python
```

6.4 Πράξεις και τελεστές στις συμβολοσειρές

- `+` : Τελεστής συνένωσης(concatenation) συμβολοσειρών, χρησιμοποιείται για να ενώσουμε το περιεχόμενο, δύο ή περισσοτέρων συμβολοσειρών.

```
>>> 'hello' + 'there'
'hellothere'
>>> str(1234) + str(5678) + str(90)
'1234567890'
```

- `*` : Τελεστής επανάληψης συμβολοσειρών, χρησιμοποιείται για να επαναλάβουμε το περιεχόμενο μίας συμβολοσειράς.

```
>>> 'hello' * 2
'hellohello'
>>> '-' * 10
'----------'
```

- `+=` : Τελεστής ο οποίος συνενώνει και δημιουργεί νέο αντικείμενο συμβολοσειράς

```
>>> text = 'Hello'
>>> text += 'World'
>>> print(text)
HelloWorld
```

- `[i]` : Τελεστής θέσης επιστρέφει το χαρακτήρα της i-θέσης.

```
>>> s = 'hello'
>>> s[len(s) - 1]
'o'
```

- `[:]` : Τελεστής διαμέρισης (slicing) συμβολοσειράς, χρησιμοποιείται για να πάρουμε ένα τμήμα της συμβολοσειράς (ο χαρακτήρας τέλους δεν περιλαμβάνεται, δηλαδή το διάστημα αρχή : τέλος είναι ανοικτό από τα δεξιά). Προκαθορισμένο βήμα (όταν δεν τοποθετηθεί κάποιο άλλο) είναι το ένα (1). `<συμβολοσειρά>[αρχή:τέλος:βήμα]`

```
>>> 'hello'[1:3]
'el'
>>> 'hello'[0::2]
'hlo'
```

- `in` : Τελεστής ύπαρξης στοιχείου, χρησιμοποιείται για να ελεγχθεί η ύπαρξη ενός χαρακτήρα ή μίας συμβολοσειράς σε μία άλλη.

```
>>> 'e' in 'hello'
True
>>> '.gr' in 'info@eellak.gr'
True
```

- `not in`: Τελεστής μη ύπαρξης στοιχείου, χρησιμοποιείται για να ελέγξουμε τη ΜΗ ύπαρξη ενός χαρακτήρα ή μίας συμβολοσειράς σε μία άλλη.

```
>>> 'w' not in 'hello'
True
```

- `<, >, !=, ==` : Σχεσιακοί Τελεστές για συγκρίσεις μεταξύ συμβολοσειρών. Η Python για να εκτελέσει τις συγκρίσεις μεταξύ των συμβολοσειρών μετατρέπει κάθε χαρακτήρα της συμβολοσειράς στον αντίστοιχο αριθμητικό κώδικα σύμφωνα με τη λειτουργία της `ord` που περιεγράφηκε προηγουμένως και συγκρίνει έναν προς έναν τους αριθμούς αυτούς, όταν βρεθούν ίδιοι προχωράει στον επόμενο μέχρι να αποτιμηθεί το αποτέλεσμα. Έτσι το:

```
>>> 'A' < 'α'
True
>>> 'Αυγο' < 'Αυριο'
True
>>> 'νόμος' != 'νομός'
True
```

Παράδειγμα 6.11

Εκτελέστε στο κέλυφος τις παρακάτω εντολές διαχείρισης των τελεστών πράξεων των συμβολοσειρών:

```
>>> a = 'i love '
>>> b = 'python'
# Επανάληψη της συμβολοσειράς
>>> c = a * 2
# Συνένωση συμβολοσειρών
>>> d = a + c
>>> print(d)
i love i love python
```

Παράδειγμα 6.12

Εκτελέστε στο κέλυφος τις παρακάτω εντολές διαχείρισης των δεικτών και του τελεστή διαμέρισης των συμβολοσειρών:

```
>>> string = 'Η γλώσσα Python και η δομή της συμβολοσειράς'
# Όταν δεν καθορίζονται όρια στον τελεστή λαμβάνονται από αρχή:τέλος
>>> print (string[:])
Η γλώσσα Python και η δομή της συμβολοσειράς
# Επιστρέφεται τμήμα της συμβολοσειράς από τον 10ο χαρακτήρα έως το τέλος
>>> string[9:]
'Python και η δομή της συμβολοσειράς'
# Επιστρέφεται τμήμα της συμβολοσειράς από την αρχή μέχρι και τον 15ο χαρακτήρα (από τον χαρακτήρα στη θέση 0 .. 14)
>>> string[:15]
'Η γλώσσα Python'
```

```
# Επιστρέφεται ο τελευταίος χαρακτήρας της συμβολοσειράς
>>> string[-1]
'ς'
# Επιστρέφονται 13 χαρακτήρες ξεκινώντας από τον τελευταίο
>>> string[-1:-14:-1]
'ςάριεσολοβμυσ'
# Επιστρέφεται όλη η συμβολοσειρά ανεστραμμένη
>>> string[::-1]
'ςάριεσολοβμυσ ςητ ήμοδ η ιακ nohtyP ασσώλγ Η'
# Η παραπάνω εντολή έχει το ίδιο αποτέλεσμα με την παρακάτω εντολή
>>> string[-1 : -len(string) - 1 : -1]
'ςάριεσολοβμυσ ςητ ήμοδ η ιακ nohtyP ασσώλγ Η'
# Δημιουργία αντιγράφου συμβολοσειράς
>>> text = 'Now is better than never.'
>> new_text = text[:]
>>> new_text
'Now is better than never.'
```

```
>>> string[7] + string[17] + string[-2]
'ααά'
>>> string[16:19] * 3
'καικαικαι'
```

Παράδειγμα 6.13

Εκτελέστε στο κέλυφος τις παρακάτω εντολές διαχείρισης των σχεσιακών και των υπαρξιακών τελεστών των συμβολοσειρών:

```
>>> string = 'Η γλώσσα Python και η δομή της συμβολοσειράς'
>>> ('python' in string.lower()) == ('python' not in string)
True
```

```
>>> n = 'anna'
>>> n == n[::-1]
True
>>> text = 'ΝΙΨΟΝ ΑΝΟΜΗΜΑΤΑ ΜΗ ΜΟΝΑΝ ΟΨΙΝ'
>>> text == n[::-1]
False
>>> text = 'ΝΙΨΟΝΑΝΟΜΗΜΑΤΑΜΗΜΟΝΑΝΟΨΙΝ'
>>> text == n[::-1]
True
>>> 'python' < 'Python'
False
```

6.4.1 Δραστηριότητα, πράξεις και τελεστές στις συμβολοσειρές

(*)Συντάξτε τις εντολές και συμπληρώστε στα κενά τι θα εμφανιστεί.

Στη συνέχεια εκτελέστε τις εντολές στο κέλυφος και ελέγξτε τις απαντήσεις σας:

```
>>> text = 'Sparse is better than dense.'
>>> text [:]

_________________
>>> text [-1:]

_________________
>>> text [-1:-5]

_________________
>>> text [-1:-7:-1]

_________________
>>> text[len(text)]

_________________
>>> text[len(text) - 1]

_________________
>>> text [10:16]

_________________
```

6.4.2 Δραστηριότητα, πράξεις και τελεστές στις συμβολοσειρές

Ανοίξτε το κινητό σας τηλέφωνο και το κέλυφος της Python και απαντήστε στις ερωτήσεις

α. Δημιουργήστε στο κινητό σας τηλέφωνο τις επαφές με επώνυμο
   - Άγγελος και Ααρών ποια τοποθετείται πρώτη;

β. Δημιουργήστε στο κινητό σας τηλέφωνο τις επαφές με επώνυμο ΑΓΓΕΛΟΣ και ΑΑΡΩΝ
   - ποια τοποθετείται πρώτη τώρα;

γ. Μπορείτε να εξηγήσετε με εντολές τις γλώσσας στο διερμηνέα τα αποτελέσματα;

6.5 Διάσχιση συμβολοσειράς

Τα μέλη μίας ακολουθίας (sequence) ή λίστας στοιχείων στην γλώσσα Python όπως συναντήθηκε σε προηγούμενο κεφάλαιο είναι εφικτό προσπελαθούν ένα προς ένα με την εντολή ορισμένων επαναλήψεων, `for`.

Παράδειγμα 6.14

Εκτελέστε στο κέλυφος τις παρακάτω εντολές διάσχισης με την εντολή for διάφορων λιστών:

```
>>> for item in [3, 4, 5, 6, 7]:
        print (item, end = '-')

3-4-5-6-7-
>>> for num in range(1, 10):
        print(num, end = ',')

1,2,3,4,5,6,7,8,9,

>>> for color in ['Red', 'Green', 'Blue']:
    print(color, end = ':')

Red:Green:Blue:
```

Στη γλώσσα python η δομή της συμβολοσειράς αποτελεί και αυτή μία ακολουθία. Συνεπώς είναι εφικτή η διάσχιση των μελών της ακολουθίας αυτής (που είναι οι χαρακτήρες που την αποτελούν) όπως ακριβώς και για τα μέλη της ακολουθίας του τύπου `range` ή μίας λίστας.

Έτσι η διάσχιση μίας συμβολοσειράς μπορεί να γίνει με 2 τρόπους:

Α) Άμεσα χωρίς τη χρήση δεικτών με τη σύνταξη:

- `for <member> in <sequence>:`

  `εντολή <member>`

Παράδειγμα 6.15

Συντάξτε και εκτελέστε στο κέλυφος:

```
>>> text = 'COMPLEX IS BETTER THAN COMPLICATED.'
>>> for char in text:
        print(char, end = '-')

C-O-M-P-L-E-X- -I-S- -B-E-T-T-E-R- -T-H-A-N- -C-O-M-P-L-I-C-A-T-E-D-.-
```

Β) Έμμεσα με τη χρήση δεικτών της δομής με τη σύνταξη:

- `for <index> in range(len(<string>):`

  `εντολή <string>[<index>]`

Παράδειγμα 6.16

Συντάξτε και εκτελέστε στο κέλυφος:

```
>>> text = 'Complex is better than complicated.'
>>> for i in range(len(text)):
    print(text[i], end =',')

C,o,m,p,l,e,x, ,i,s, ,b,e,t,t,e,r, ,t,h,a,n, ,c,o,m,p,l,i,c,a,t,e,d,.,
```

6.5.1 Δραστηριότητα (διάσχιση συμβολοσειρών)

Μία συμβολοσειρά ονομάζεται συμμετρική ή παλίνδρομη εάν διαβάζεται το ίδιο ξεκινώντας από την αρχή είτε το τέλος της πχ ΑΝΝΑ, ΣΑΒΒΑΣ.

(*)Συντάξτε στον συντάκτη ή στο κέλυφος και συμπληρώστε τα κενά στον παρακάτω κώδικα ώστε οι συναρτήσεις να ελέγχουν εάν η συμβολοσειρά που δέχονται είναι παλίνδρομη ή όχι.
Στη συνέχεια καλέστε τις συναρτήσεις με είσοδο τα ονόματα: ΑΝΝΑ, ΣΑΒΒΑΣ, ΆΝΝΑ, ΣΆΒΒΑΣ και ελέγξτε τις απαντήσεις σας.

```
def is_palindrom1(s):
    n = len(_)
    for i in range(n /_ _):
        if s[i] != s[______]:
            return ______
    return ______

def is_palindrom2(s):
    return s == s[__:__:__]
```

6.6 Κωδικοποίηση συμβολοσειρών

Οι υπολογιστές χρησιμοποιούν πίνακες (κώδικες) μετατροπής χαρακτήρων σε αριθμούς προκειμένου να αναπαραστήσουν τα γράμματα, τα σύμβολα, τα αριθμητικά ψηφία κλπ. από τα οποία αποτελείται μία συμβολοσειρά. Οι πιο γνωστοί τέτοιοι κώδικες είναι ο κώδικας ASCII και ο UNICODE.

Ο τελευταίος χρησιμοποιείται στην κωδικοποίηση όλων των συστημάτων γραφής που συναντώνται σήμερα στον πλανήτη και υποστηρίζεται πλήρως από την Python 3.11.

Η Python διαθέτει τις συναρτήσεις `chr()` και `ord()` για το χειρισμό των χαρακτήρων των συμβολοσειρών:

- `chr(<θετικός ακέραιος>)` :Συνάρτηση η οποία επιστρέφει **το χαρακτήρα** που βρίσκεται στη θέση αυτή στον κώδικα Unicode

```
>>> chr(945)
'α'
>>> chr(913)
'A'
>>> chr(8364)
'€'
>>> chr(163)
'£'
```

- `ord(<χαρακτήρας>)` :Συνάρτηση η οποία επιστρέφει **τη θέση** που βρίσκεται ο χαρακτήρας αυτός στον κώδικα Unicode

```
>>> ord('α')
945
>>> ord('A')
913
>>> ord('ά')
940
>>> ord('€')
8364
>>> ord('£')
163
```

- *Παρατηρήστε ότι οι δύο συναρτήσεις `chr` και `ord` εκτελούν την αντίστροφη λειτουργία.*
- *Παρατηρήστε επίσης ότι τονισμένα και άτονα φωνήεντα βρίσκονται σε διαφορετική θέση.*

```
>>> chr(ord('a'))
'a'
```

6.6.1 Δραστηριότητα, συναρτήσεις, ord, chr

(,**)Συντάξτε τις παρακάτω εντολές στο κέλυφος και συμπληρώστε τα κενά ώστε να απαντήστε στα ερωτήματα στη συνέχεια εκτελέστε και ελέγξτε τις απαντήσεις σας:*

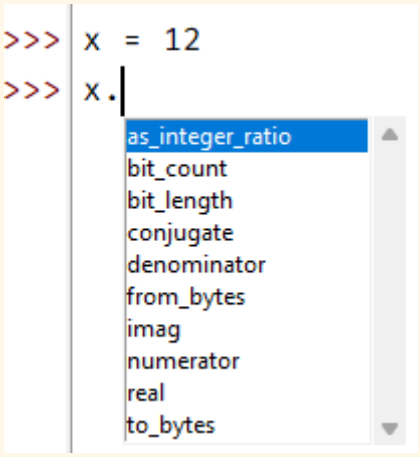
```
# Α. Συμπληρώστε τι θα εμφανίσει ο παρακάτω κώδικας
```

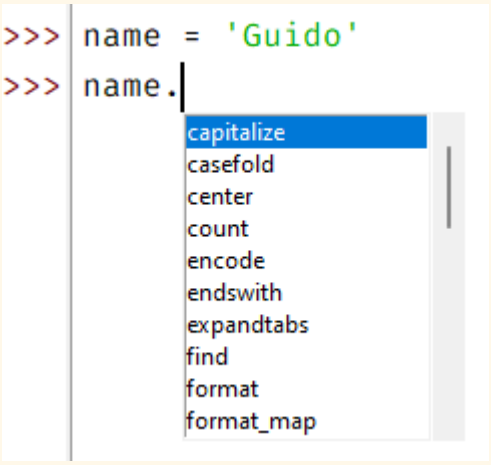
```
>>> for i in range(913, 913 + 25):
        print(chr(i), end=' ')

_________________
# Β. Συμπληρώσετε τον παρακάτω κώδικα ώστε να εμφανιστούν
# όλοι πεζοί χαρακτήρες του ελληνικού αλφαβήτου (α, β, γ, ..., ω)
>>> for i in range(___, ____):
        print(chr(i), end=' ')

# Γ. Χρησιμοποιώντας τη συνάρτηση ord βρείτε τα αριθμητικά
# όρια του λατινικού αλφαβήτου στον κώδικα Unicode και στη συνέχεια


_________________
_________________
# Δ. Εκτελέστε αντίστοιχες εντολές για να τυπώσετε όλους
# τους κεφαλαίους και τους πεζούς χαρακτήρες του λατινικού αλφαβήτου

>>> for ______________________
        ______________________

>>> for ______________________
        ______________________
```

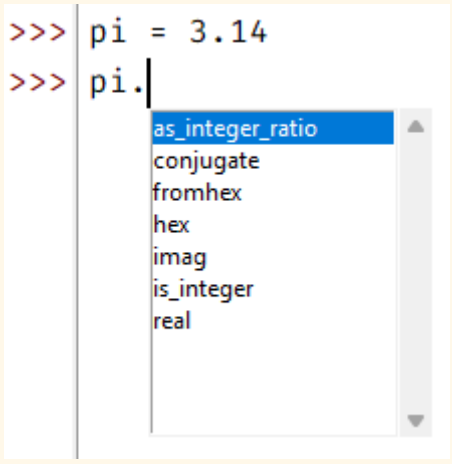
6.7 Αντικείμενα και λειτουργικότητα

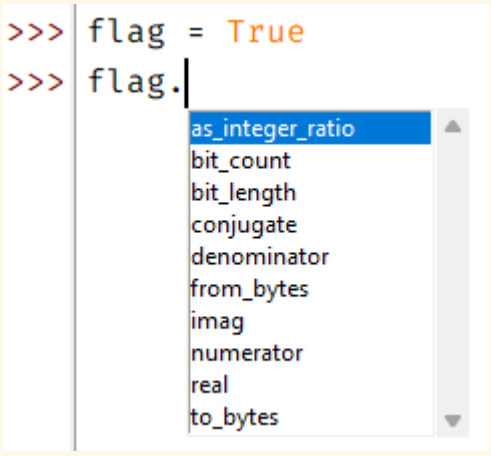
Όπως αναφέρθηκε τα αντικείμενα έχουν δεδομένα/ κατάσταση (για τους απλούς τύπους αντικειμένων `int`, `float`, `str`, `bool` είναι η τιμή τους) καθώς και συμπεριφορά/ λειτουργικότητα η οποία εκφράζεται προγραμματιστικά με συναρτήσεις ειδικού τύπου (καθορίζονται εντός της κλάσης του αντικειμένου και θα μελετηθούν αναλυτικά σε επόμενο κεφάλαιο) που ονομάζονται **μέθοδοι (methods)**. Οι συναρτήσεις αυτές (μέθοδοι) καλούνται με συγκεκριμένο τρόπο που ονομάζεται **σημειογραφία τελείας** (dot notation). Κάθε αντικείμενο ανάλογα την κλάση (τον τύπο του) έχει τη δική του λειτουργικότητα/ συμπεριφορά η οποία εκφράζεται μέσω των δικών του μεθόδων.

Έτσι δημιουργώντας ένα αντικείμενο στο κέλυφος της γλώσσας για να υπάρξει πρόσβαση στις μεθόδους του αντικειμένου προστίθεται μετά από το όνομά του (δηλαδή την αναφορά σε αυτό με τη χρησιμοποιούμενη μεταβλητή) μία τελεία (.). Το περιβάλλον της γλώσσας όπως και των περισσότερων IDEs εμφανίζει αυτόματα τις ενσωματωμένες μεθόδους κάθε αντικειμένου.









Παρατηρήσετε ότι κάθε αντικείμενο διαθέτει διαφορετικές μεθόδους ανάλογα τον τύπο του (δηλαδή την κλάση από την οποία προέρχεται).

6.8 Λειτουργικότητα συμβολοσειρών

Όπως αναφέρθηκε στην αρχή του κεφαλαίου κάθε αντικείμενο εκτός από τα δεδομένα του διαθέτει και μία λειτουργικότητα (συμπεριφορά) η οποία εκφράζεται μέσω κάποιων ειδικών συναρτήσεων που ονομάζονται μέθοδοι. Τα αντικείμενα τύπου συμβολοσειράς διαθέτουν ένα μεγάλο πλήθος έτοιμων μεθόδων οι οποίες μπορούν να χρησιμοποιηθούν για τις ανάγκες του χρήστη. Κάποιες (όχι όλες) θα αναφερθούν στη συνέχεια.

6.8.1 Μέθοδοι συμβολοσειρών upper(), lower(), title()

Οι μέθοδοι που εκφράζουν τη λειτουργικότητα ενός αντικειμένου καλούνται όπως αναφέραμε με τη σημειογραφία τελείας ως εξής:

```
<αντικείμενο>.<μέθοδος(παράμετροι μεθόδου)>
```

Από τις πιο συχνά χρησιμοποιούμενες μεθόδους στις συμβολοσειρές είναι αυτές με τις οποίες «μεταβάλλουμε» το περιεχόμενο μίας συμβολοσειράς **επιστρέφοντας** το αποτέλεσμα σε μία άλλη νέα συμβολοσειρά. Μπορείτε να ανατρέξετε εδώ για περισσότερες πληροφορίες σε όλες τις διαθέσιμες μεθόδους συμβολοσειρών της Python.

Η μετατροπή πεζών γραμμάτων σε κεφαλαία και το αντίστροφο είναι πολύ συχνή αλλά και χρήσιμη λειτουργία στην επεξεργασία κειμένου οι ενσωματωμένες μέθοδοι για τη χρήση είναι:

- `<string>.lower()` :Η μέθοδος όταν κληθεί από μία συμβολοσειρά επιστρέφει τη συμβολοσειρά με τα γράμματα που περιέχει από κεφαλαία σε πεζά
- `<string>.upper()` :Η μέθοδος όταν κληθεί από μία συμβολοσειρά επιστρέφει τη συμβολοσειρά με τα γράμματα που περιέχει από πεζά σε κεφαλαία
- `<string>.title()` :Η μέθοδος όταν κληθεί από μία συμβολοσειρά επιστρέφει τη συμβολοσειρά με το πρώτο γράμμα κάθε λέξης κεφαλαίο.

Παράδειγμα 6.17

Συντάξτε και εκτελέστε για διαχείριση των μεθόδων μετατροπής κεφαλαίων χαρακτήρων ← → πεζούς χαρακτήρες:

```
# Κλήση της μεθόδου lower() για το αντικείμενο 'PYTHON'
>>> 'PYTHON'.lower()
'python'
# Κλήση της μεθόδου lower() για το αντικείμενο με όνομα name
>>> name = 'ANNA'
>>> name.lower()
'anna'
# ΠΡΟΣΟΧΗ οι συμβολοσειρές είναι αμετάβλητα αντικείμενα
# οπότε το αντικείμενο αναφοράς της name δεν μεταβλήθηκε
>>> print(name)
'ANNA'
# Εάν επιθυμούμε να κρατήσουμε το όνομα στα κεφαλαία
# πρέπει να δώσουμε όνομα νέας αναφοράς(μεταβλητής)
```

```
# στην επιστροφή της μεθόδου lower() ως εξής:
>>> name1= name.lower()
>>> print(name1)
'anna'
# Αντίστοιχα λειτουργεί και η μέθοδος upper()
>>> 'anna'.upper()
'ANNA'
# Κλήση της μεθόδου tile() για το αντικείμενο συμβολοσειράς text
>>> text = 'Special cases are not special enough to break the rules.'
>>> text.title()
'Special Cases Are Not Special Enough To Break The Rules.'
```

6.8.2 Δραστηριότητα, μέθοδοι upper(), lower()

(,**)Συντάξτε τις παρακάτω εντολές στο κέλυφος και συμπληρώστε τα κενά που εμφανίζονται τα αποτελέσματα στη συνέχεια εκτελέστε τις εντολές και ελέγξτε τις απαντήσεις σας:*

```
>>> name = 'Guido'
>>> print(name.upper())

______________
>>> print(name.lower())

______________
>>> print(name.lower().upper())

______________
>>> name = name.upper()
>>> print(name)

______________
>>> 'α1β1γ1'.upper()

_______________
# Ε. Μπορείτε χωρίς τη χρήση των μεθόδων  upper(), lower() να εμφανίσετε:
Αα Ββ Γγ Δδ Εε Ζζ Ηη Θθ Ιι Κκ Λλ Μμ Νν Ξξ Οο Ππ Ρρ ς Σσ Ττ Υυ Φφ Χχ Ψψ Ωω
```

6.9 Πρόβλημα. Απόδοση Προσωρινών Διαπιστευτηρίων

Στη συνέχεια θα μελετηθεί ένα γενικό πρόβλημα που αφορά μεταξύ άλλων και το χειρισμό των συμβολοσειρών.

Σε ένα σύστημα ελέγχου πρόσβασης ζητείται να δημιουργηθούν προσωρινά διαπιστευτήρια εισόδου για κάθε ένα χρήστη ως εξής: Θα ζητούνται το όνομά του και το επώνυμό του με λατινικούς χαρακτήρες καθώς και η ημερομηνία γέννησής του με τη μορφή dd/mm/yyyy.
Στη συνέχεια θα εμφανίζονται:

- **Το όνομα πρόσβασης (username) που θα αποτελείται από 6 πεζούς χαρακτήρες: 2 από τους πρώτους χαρακτήρες του ονόματός του και άλλους 4 από τους πρώτους χαρακτήρες του επωνύμου του.**
- **Ο κωδικός πρόσβασης (password) που θα αποτελείται από 8 χαρακτήρες: 2 από τα αρχικά, ονόματος και επωνύμου με κεφαλαίους χαρακτήρες και 6 ψηφία από την ημερομηνία, την ημέρα, το μήνα και τα 2 τελευταία ψηφία από το έτος γέννησής του.**

Σημείωση:
Σε περίπτωση όπου, κάποιος από τους 2 χαρακτήρες του ονόματος, ή κάποιος από τους 4 χαρακτήρες του επωνύμου δεν αντιστοιχεί σε γράμμα του λατινικού αλφαβήτου ή κάποιος από τους χαρακτήρες της ημερομηνίας γέννησης δεν αντιστοιχεί σε ψηφίο θα εμφανίζονται ως username και password οι λέξεις invalid.

Η εισαγωγή των χρηστών θα σταματά όταν ως όνομα δοθεί το 'enter'. Πχ

```
George
Papavasileiou
13/10/1985
Username    : gepapa
Password    : GP131085
```

6.9.1 Επίλυση του προβλήματος, Απόδοση Προσωρινών Διαπιστευτηρίων

Αλγόριθμος της λύσης.

1. Εισαγωγή δεδομένων ονόματος
2. Επανάλαβε τα βήματα 2.1 – 2.8 όσο ο χρήστης δίνει κάποιο όνομα
   2.1. Εισαγωγή επωνύμου, ημερομηνίας γέννησης
   2.2. Εντολές απόσπασης χαρακτήρων από το όνομα και το επώνυμο
   2.3. Εντολές απόσπασης ψηφίων από την ημερομηνία
   2.4. Έλεγχος **για κάθε** χαρακτήρα του username εάν είναι λατινικό γράμμα
   2.5. Έλεγχος **για κάθε** χαρακτήρα της εξαγόμενης ημερομηνίας εάν είναι ψηφίο
   2.6. Εάν τα δεδομένα πέρασαν και τους 2 ελέγχους σύνθεσε το password αλλιώς τα διαπιστευτήρια είναι invalid
   2.7. Εμφάνιση αποτελεσμάτων
   2.8. Εισαγωγή επόμενου ονόματος

Προσπαθήσετε να υλοποιήσετε σε γλώσσα Python τον παραπάνω αλγόριθμο χωρίς να δείτε την ενδεικτική λύση.

```
print("Χρησιμοποιήστε λατινικούς χαρακτήρες")
fname = input("Δώστε το όνομα του χρήστη : ")
while fname != "":
    # Εισαγωγή στοιχείων
    lname = input("Δώστε το επώνυμο του χρήστη : ")
    birth = input("Δώστε την ημερομηνία γέννησης dd/mm/yyyy : ")

    ## Δημιουργία στοιχείων για σύνθεση username/password
    # Οι 2 πρώτοι χαρακτήρες του ονόματος σε πεζούς χαρακτήρες
    part1 = fname[:2].lower()
    # Οι 4 πρώτοι χαρακτήρες του επωνύμου σε πεζούς χαρακτήρες
    part2 = lname[:4].lower()
    # Οι αριθμοί μόνο από την ημερομηνία
    part3 = birth[:2] + birth[3:5] + birth[8:]

    # Έλεγχος εάν κάθε χαρακτήρας του username είναι λατινικό γράμμα
    check1 = True
    username = part1 + part2
    for char in username:
        if char not in "abcdefghijklmnopqrstuvwxyz":
            check1 = False
            break

    # έλεγχος εάν κάθε χαρακτήρας της εξαγόμενης ημερομηνίας είναι ψηφίο
    check2 = True
    for char in part3:
        if char not in "0123456789":
            check2 = False
            break

    # Εάν τα δεδομένα πέρασαν και τους 2 ελέγχους σύνθεσε το password
    # το username είναι ήδη έτοιμο
    if check1 and check2:
        # password σε κεφαλαίους χαρακτήρες
        password = fname[0].upper() + lname[0].upper() + part3
    else:
        username = "invalid"
        password = "invalid"

    # Εμφάνιση αποτελεσμάτων
    print("username\t:", username)
    print("password\t:", password)
    fname = input("Δώστε το όνομα του χρήστη : ")
```

6.10 Αναζήτηση

Η λειτουργία της αναζήτησης (searching) χρησιμοποιείται ώστε να λαμβάνεται η πληροφορία ύπαρξης ή όχι, μίας τιμής (συχνά ονομάζεται κλειδί: key,) σε μία συλλογή δεδομένων. Η πληροφορία αυτή μπορεί να αφορά:

α. στο εάν υπάρχει ή όχι η τιμή αυτή,
β. σε ποια θέση βρίσκεται ή
γ. εάν η τιμή επαναλαμβάνεται πολλές φορές στη συλλογή, σε όλες τις θέσεις που βρίσκεται αυτή.

Η αναζήτηση απαιτείται πολύ συχνά σε προγράμματα που χρησιμοποιούνται στην καθημερινότητα:

- κάθε φορά που πληκτρολογούνται τα στοιχεία μίας επαφής σε μια κινητή συσκευή,
- σε κάθε σύνδεση που ζητούνται διαπιστευτήρια πχ σε μία ιστοσελίδα,
- σε κάθε αναζήτηση σε μία μηχανή αναζήτησης κ.ά..

Οι βασικοί αλγόριθμοι αναζήτησης είναι δύο:

α. Η σειριακή αναζήτηση (sequential search) και
β. Η δυαδική αναζήτηση (binary search).

Όταν η δομή δεδομένων έχει λίγα στοιχεία, ο τρόπος που τοποθετούνται τα στοιχεία στη δομή (πχ με κάποια σειρά ή όχι) όπως και ο αλγόριθμος αναζήτησης ενός στοιχείου σε αυτή, δεν έχουν ιδιαίτερη πρακτική σημασία διότι οι χρόνοι αναζήτησης είναι μικροί.

Όσο όμως αυξάνονται τα στοιχεία της δομής (10^3, 10^6, 10^9, 10^{12} , ..) και όσο πιο συχνές αναζητήσεις εκτελούνται, τόσο πιο σημαντικό ρόλο παίζουν στον χρόνο αναζήτησης α) η διάταξη των στοιχείων της δομής και β) ο αλγόριθμος που θα επιλεχθεί.

Σειριακή Αναζήτηση

Η σειριακή αναζήτηση γίνεται σε μία δομή δεδομένων όταν τα στοιχεία της δομής αυτής δεν είναι τοποθετημένα με κάποια διάταξη ή σειρά (unordered). Έτσι προκειμένου να βρεθεί το επιθυμητό κλειδί η αναζήτηση πρέπει να γίνει από την αρχή μέχρι το τέλος της δομής ή μέχρι να βρεθεί το κλειδί αναζήτησης. Ο μέγιστος αριθμός αναζητήσεων που θα εκτελεστεί είναι ίσος με το πλήθος των στοιχείων της δομής (στην περίπτωση όπου το κλειδί βρίσκεται στην τελευταία θέση: worst case scenario).

Παράδειγμα 6.18

Θα παρουσιαστούν στη συνέχεια διάφορες εκδοχές της σειριακής αναζήτησης με μορφή συναρτήσεων οι οποίες μπορούν να εκτελεστούν σε μία ακολουθιακή δομή δεδομένων όπως είναι αυτή των συμβολοσειρών (οι ίδιες συναρτήσεις μπορούν να χρησιμοποιηθούν και σε άλλες ακολουθιακές δομές της Python όπως πχ στις λίστες).

```
# Σειριακή αναζήτηση στοιχείου σε ακολουθία, 1η θέση εμφάνισης
def serial_search_first(key, sequence):
    for i in range(len(sequence)):
        if key == sequence [i]:
```

```
            return i
    return -1
# Σειριακή αναζήτηση στοιχείου σε ακολουθία, όλες οι θέσεις εμφάνισης
def serial_search_all(key, sequence):
    positions = []
    for i in range(len(sequence)):
        if key == sequence[i]:
            positions += [i]
    return positions
# Αναζήτηση της θέσης μίας συμβολοσειράς σε μία άλλη συμβολοσειρά
def search(sub_string, string):
    n = len(sub_string)
    m = len(string)
    for i in range(m - n + 1):
        if sub_string == string[i : i + n]:
            return i
    return -1

text = 'There should be one and preferably only one obvious way to do it'
# Η πρώτη θέση που βρίσκεται ο χαρακτήρας 'o' στο αντικείμενο text
serial_search_first('o', text)
8

# Σε ποιες θέσεις βρίσκεται ο χαρακτήρας 'o' στο αντικείμενο text
serial_search_all('o', text)
[8, 16, 35, 40, 44, 48, 57, 61]

>>> text = 'There should be one and preferably only one obvious way to do
it'
# Σε ποια θέση υπάρχει η υποσυμβολοσειρά 'one' στη συμβολοσειρά text
>>> search('one', text)
16
# Σε ποια θέση υπάρχει η υποσυμβολοσειρά 'it' στη συμβολοσειρά text
>>> search('it', text)
62

>>> text = 'There should be one and preferably only one obvious way to do
it'
# Υπάρχει ή όχι ο χαρακτήρας 'w' στη συμβολοσειρά text
>>> 'w' in text
True
# Υπάρχει ή όχι ο χαρακτήρας 'z' στη συμβολοσειρά text
>>> 'z' in text
```

```
False
```

Δυαδική Αναζήτηση

Η Δυαδική αναζήτηση είναι ένας εξαιρετικά αποδοτικός αλγόριθμος αναζήτησης αλλά είναι εφικτή μόνο σε ταξινομημένη δομή δεδομένων (σε δομή δηλαδή όπου τα στοιχεία της βρίσκονται σε κάποια διάταξη) και θα μελετηθεί σε επόμενο μάθημα.

6.10.1 Μέθοδοι αναζήτησης της Python για τις συμβολοσειρές

Αφού όπως αναφέρθηκε η αναζήτηση είναι μία πολύ συχνή λειτουργία στις συμβολοσειρές η Python διαθέτει ενσωματωμένες μεθόδους για την εξυπηρέτηση κάποιων βασικών αλγοριθμικών λειτουργιών αναζήτησης.

Αυτές είναι οι μέθοδοι `find()`, `count()` με τις οποίες είναι εφικτή η αναζήτηση μίας συμβολοσειράς μέσα σε μία άλλη συμβολοσειρά.

- `<string>.find(sub_string, start, end)` :Αναζητεί και επιστρέφει την πρώτη θέση της υποσυμβολοσειράς (sub_string), στη συμβολοσειρά (string) ξεκινώντας από τη θέση start (εάν δε δοθεί, λαμβάνεται η αρχή: 0) μέχρι τη θέση end (εάν δε δοθεί, λαμβάνεται το τέλος της συμβολοσειράς). Εάν δεν βρεθεί η υποσυμβολοσειρά επιστρέφεται από τη μέθοδο το -1.

```
>>> s = 'implementation'
>>> s.find('i')
0
>>> s.find('i', 1)
11
>>> s.find('i', 12)
-1
```

- `<string>.count(sub_string, start, end)` :Αναζητεί και επιστρέφει πόσες φορές εμφανίζεται η υποσυμβολοσειρά (sub_string) εντός της συμβολοσειράς (string) ξεκινώντας από τη θέση start (εάν δε δοθεί, λαμβάνεται η αρχή:0) μέχρι τη θέση end (εάν δε δοθεί, λαμβάνεται το τέλος της συμβολοσειράς). Εάν δεν βρεθεί η υποσυμβολοσειρά επιστρέφεται από τη μέθοδο το 0.

```
>>> s = 'hippopotamus'
>>> s.count('i')
1
>>> s.count('po')
2
>>> s.count('w')
0
```

Παράδειγμα 6.19

Παρουσίαση διαφόρων χρήσεων των μεθόδων find και count της Python σε πιο σύνθετες αναζητήσεις:

```
# Η μέθοδος find() παρόλο που βρίσκει ΜΟΝΟ την 1η θέση εμφάνισης
# ενός substring σε ένα string  μπορεί να χρησιμοποιηθεί για να βρίσκει
# όλες τις θέσεις που αυτό εμφανίζεται βλ τη συνάρτηση που ακολουθεί

def find_all(sub_string, string):
    positions = []
    pos = string.find(sub_string)
    while pos != -1:
        positions += [pos]
        pos = string.find(sub_string, pos + 1)
    return positions

>>> text = 'There should be one and preferably only one obvious way to do
it'
>>> find_all('one', text)
[16, 40]
>>> find_all('o', text)
[8, 16, 35, 40, 44, 48, 57, 60]
```

6.10.2 Δραστηριότητες (Αναζήτηση)

(*)Να υλοποιηθεί μία συνάρτηση η οποία θα δέχεται ένα κείμενο και θα επιστρέφει το πλήθος των λέξεων που υπάρχουν σε αυτό. Οι λέξεις στο κείμενο χωρίζονται με τους χαρακτήρες (κενό, τελεία, κόμμα, θαυμαστικό, ελληνικό ή αγγλικό ερωτηματικό) και η πρόταση τελειώνει πάντα με ένα από τα παραπάνω σημεία στίξης.

(**)Η λύση της παραπάνω δραστηριότητας μπορεί να επιτευχθεί και χρησιμοποιώντας τη μέθοδο count συμπληρώνοντας τα κενά στον παρακάτω κώδικα.

```
def count_words(text):
    words = _
    symbols = ' .,!;?'

    for sep in _______:
        ______ += text.count(___)
    return _____
```

6.11 Πρόβλημα Α. Στοίχιση πρότασης

(***) Μία από τις πιο συχνές λειτουργίες ενός επεξεργαστή κειμένου πχ Libre Office είναι αυτή της πλήρους στοίχισης μίας γραμμής ή μίας παραγράφου. Αυτό επιτυγχάνεται αυξάνοντας ομοιόμορφα τα κενά μεταξύ των λέξεων. Εδώ ζητείται να υλοποιήσετε μία συνάρτηση με όνομα `align` η οποία θα δέχεται ένα κείμενο μίας γραμμής που περιέχει λέξεις χωρισμένες με ένα κενό, καθώς και έναν θετικό ακέραιο με τιμή μεγαλύτερη από το μήκος του κειμένου. Η συνάρτηση αυξάνοντας ομοιόμορφα το πλήθος των κενών χαρακτήρων μεταξύ των λέξεων του αρχικού κειμένου θα επιστρέφει νέο κείμενο μίας

γραμμής με μήκος ίσο με το θετικό ακέραιο. Εάν δεν μπορεί να γίνει απόλυτα ομοιόμορφη κατανομή των κενών αυτά που υπολείπονται τοποθετούνται ανάμεσα από τις λέξεις αρχίζοντας από τα αριστερά προς τα δεξιά.

```
>>> align('Δύσκολη φαίνεται αυτή η δραστηριότητα', 60)
------------------------------------------------------------

Δύσκολη       φαίνεται       αυτή       η      δραστηριότητα
```

← 7 → ← 7 → ← 7 → ← 6 →

Αλγόριθμος της λύσης.

1. Μετράμε το τρέχον μήκος του κειμένου
2. Εάν το νέο πλάτος είναι μεγαλύτερο από το μήκος του κειμένου
   2.1. Μετράμε πόσα κενά υπάρχουν στο κείμενο
   2.2. Υπολογίζουμε την επέκταση του κειμένου
   2.3. Υπολογίζουμε το πλήθος των κενών που πρέπει να προστεθούν στα ήδη υπάρχοντα
   2.4. Επειδή το πλήθος αυτό πρέπει να είναι ακέραιος υπολογίζεται το υπόλοιπο
   2.5. Για κάθε χαρακτήρα του παλαιού κειμένου
      2.5.1. Τοποθετείται στο νέο κείμενο
      2.5.2. Εάν ο χαρακτήρας είναι το κενό
         2.5.2.1. τοποθετούνται και όσα επιπλέον κενά υπολογίστηκαν
         2.5.2.2. και ένα ακόμα κενό από τα υπόλοιπα (που περισσεύουν)
         2.5.2.3. μειώνονται τα κενά που υπολείπονται
   2.6. Τυπώνεται μία συμβολοσειρά ελέγχου
   2.7. Τυπώνεται το νέο κείμενο
   2.8. Επιστρέφεται το νέο κείμενο
3. Αλλιώς επιστρέφεται το κείμενο ως έχει

Προσπαθήστε να υλοποιήσετε τον αλγόριθμο χωρίς να δείτε την ενδεικτική λύση

```
def align(text, width):
    n = len(text)
    if width > n:
        blanks = text.count(" ")
        new_text = ""
        extension = width - n
        extra = extension // blanks
        remainder = extension % blanks
        for char in text:
            new_text += char
            if char == " ":
                new_text += " " * extra
                if remainder > 0:
                    new_text += " "
                    remainder -= 1
```

```
        print("-" * width)
        print(new_text)
        return new_text
    else:
        return text
```

6.12 Καθαρισμός, συνένωση, διαχωρισμός, αντικατάσταση σε συμβολοσειρές

6.12.1 Καθαρισμός συμβολοσειρών μέθοδοι lstrip(), rstrip(), strip()

Πολύ συχνά οι συμβολοσειρές ανάλογα από πού προέρχονται τα δεδομένα που περιέχουν (εισαγωγή από το χρήστη, από αρχεία κειμένου, από λειτουργία αντιγραφής επικόλλησης, από αποκόμματα ιστού (web scrapings), από δεδομένα αισθητήρων κ.ά.) υπάρχει περίπτωση να έχουν στην αρχή ή στο τέλος τους περιττούς κενούς χαρακτήρες ή χαρακτήρες διαφυγής (escape characters/ white spaces) οι οποίοι δεν εμφανίζονται στην οθόνη του υπολογιστή.

Αυτό μπορεί να προκαλέσει περίεργα αποτελέσματα εμφάνισης, λογικά λάθη ή λάθη χρόνου εκτέλεσης.

Η Python διαθέτει 3 μεθόδους καθαρισμού τέτοιων χαρακτήρων από τις συμβολοσειρές: από αριστερά, από δεξιά, από αριστερά και από δεξιά: `lstrip()`, `rstrip()`, `strip()`.

- `<string>.lstrip()` :Επιστρέφει συμβολοσειρά αφού έχει αφαιρέσει όλους τους λευκούς χαρακτήρες από αριστερά από τα αριστερά της συμβολοσειράς.
- `<string>.rstrip()` :Επιστρέφει συμβολοσειρά αφού έχει αφαιρέσει όλους τους λευκούς χαρακτήρες από τα δεξιά της συμβολοσειράς
- `<string>.strip()` :Επιστρέφει συμβολοσειρά αφού έχει αφαιρέσει όλους τους λευκούς χαρακτήρες από τα αριστερά και από τα δεξιά της συμβολοσειράς

Παράδειγμα 6.20

Συντάξτε και εκτελέστε τις εντολές διαχείρισης καθαρισμού συμβολοσειρών από λευκούς χαρακτήρες:

```
>>> text = '\n\n      \t Καλημέρα Ελλάδα \n\n\t'
>>> print(text)


            Καλημέρα Ελλάδα



# Καθαρισμός λευκών χαρακτήρων από αριστερά
>>> text.lstrip()
'Καλημέρα Ελλάδα \n\n\t'
>>> print(text)
# Καθαρισμός λευκών χαρακτήρων από δεξιά
>>> text.rstrip()
'\n\n      \t Καλημέρα Ελλάδα'
>>> print(text)
# Καθαρισμός λευκών χαρακτήρων από αριστερά και δεξιά
>>> text.strip()
'Καλημέρα Ελλάδα'
print(text)
```

6.12.2 Συνένωση, διαχωρισμός συμβολοσειρών μέθοδοι join(), split()

Παρουσιάστηκε σε προηγούμενη παράγραφο η λειτουργία του τελεστή συνένωσης συμβολοσειρών (+). Εκτός από τον τελεστή αυτό υπάρχει η ενσωματωμένη μέθοδος `join()` η οποία μπορεί να χρησιμοποιηθεί για να συνενώσει τα στοιχεία χαρακτήρων μίας ακολουθίας σε μία ενιαία συμβολοσειρά χρησιμοποιώντας κάποιο χαρακτήρα ή κάποια συμβολοσειρά διαχωρισμού.

- `<string>.join(<διασχίσιμη δομή>)` :Συνενώνει όλα τα στοιχεία της δομής (πχ λίστας, πλειάδας, λεξικού) επιστρέφοντας μία ενιαία συμβολοσειρά. Τα στοιχεία της ακολουθίας πρέπει να είναι τύπου string, ως διαχωριστικό μεταξύ των στοιχείων χρησιμοποιείται το περιεχόμενο της string.

Παράδειγμα 6.21

Συντάξτε και εκτελέστε τις εντολές για τη λειτουργία της μεθόδου συνένωσης συμβολοσειρών join:

```
# Συνένωσε τα στοιχεία της λίστας χωρίς διαχωριστικό χαρακτήρα
>>> ''.join(['Red', 'Green', 'Blue'])
'RedGreenBlue'
# Συνένωσε τα στοιχεία της λίστας με διαχωριστικό το χαρακτήρα -
>>> '-'.join(['Red', 'Green', 'Blue'])
'Red-Green-Blue'
# Συνένωσε τα στοιχεία της λίστας με διαχωριστικό το string (Color)
>>> '(Color)'.join(['Red', 'Green', 'Blue'])
'Red(Color)Green(Color)Blue'
# Προσοχή όταν η λίστα έχει μόνο ένα στοιχείο
# δε χρησιμοποιείται το διαχωριστικό
>>> 'Καλημέρα'.join(['Ελλάδα'])
'Ελλάδα'
>>> 'Καλημέρα'.join(['Ελλάδα','.'])
'ΕλλάδαΚαλημέρα.'
```

Η αντίστροφη λειτουργία από τη συνένωση συμβολοσειρών είναι ο διαχωρισμός τους. Αυτός στη γλώσσα Python επιτυγχάνεται με την ενσωματωμένη μέθοδο `split()`. Η μέθοδος αυτή λαμβάνει μια συμβολοσειρά και ένα διαχωριστικό χαρακτήρα ή συμβολοσειρά και επιστρέφει μια λίστα με τα στοιχεία της συμβολοσειράς τα οποία χωρίζονται από το διαχωριστικό χαρακτήρα που χρησιμοποιήθηκε.

- `<string>.split(<διαχωριστικός χαρακτήρας>, maxsplit=<πλήθος διαχωρισμών>)` :Διαχωρίζει τα στοιχεία της συμβολοσειράς string επιστρέφοντας μια λίστα των στοιχείων αυτών. Ο διαχωρισμός των στοιχείων επιτυγχάνεται με το διαχωριστικό χαρακτήρα. Όταν αυτός δε δίνεται λαμβάνεται το κενό. Η παράμετρος `maxsplit` είναι προαιρετική για να περιορίσουμε το πλήθος των διαχωρισμών που θα εκτελεστούν.

Παράδειγμα 6.22

Συντάξτε και εκτελέστε τις εντολές για τη λειτουργία της μεθόδου διαχωρισμού συμβολοσειρών split:

```
# Διαχωρισμός με τη χρήση του διαχωριστικού χαρακτήρα ,
>>> 'Καλημέρα,Ελλάδα'.split(',')
['Καλημέρα', 'Ελλάδα']
# Διαχωρισμός με τη χρήση του διαχωριστικού χαρακτήρα @
>>> 'info@ellak.gr'.split('@')
['info', 'ellak.gr']
# Διαχωρισμός όταν  οι λέξεις διαχωρίζονται με ΕΝΑ ΜΟΝΟ κενό
>>> text = 'Errors should never pass silently.'
>>> text.split()
['Errors', 'should', 'never', 'pass', 'silently.']
# Διαχωρισμός όταν οι λέξεις διαχωρίζονται με ΠΑΡΑΠΑΝΩ από ΕΝΑ κενά
>>> text = 'Errors      should     never    pass    silently.'
>>> text.split()
['Errors', 'should', 'never', 'pass', 'silently.']
# Διαχωρισμός όταν  οι λέξεις διαχωρίζονται με ΠΑΡΑΠΑΝΩ από ΕΝΑ κενά
# ΠΡΟΣΟΧΗ ΤΙ ΘΑ ΣΥΜΒΕΙ ΕΑΝ καθορίσουμε διαχωριστικό χαρακτήρα
>>> text = 'Errors      should     never    pass    silently.'
>>> text.split(' ')
['Errors', '', '', '', '', '', 'should', '', '', '', '', 'never', '', '',
'', 'pass', '', '', '', 'silently.']
# Καθορισμός πλήθους διαχωρισμών
text = 'Errors should never pass silently.'
text.split(maxsplit=2)
['Errors', 'should', 'never pass silently.']
```

6.13 Ιεραρχική Σχεδίαση και τμηματικός προγραμματισμός

Η ιεραρχική σχεδίαση επίλυσης ενός σύνθετου προβλήματος περιλαμβάνει τη διαδικασία διάσπασής του σε μικρότερα (και άρα απλούστερα) προβλήματα τα οποία είναι πιο εύκολο να επιλυθούν και η λύση των οποίων επιλύει και το αρχικό πρόβλημα. Η τεχνική αυτή σχεδίασης ονομάζεται και από επάνω προς τα κάτω (top down design). Υλοποιείται με τον τμηματικό προγραμματισμό όπου επιλύεται κάθε υποπρόβλημα με ένα υποπρόγραμμα (στην Python με συνάρτηση). Η σύνθεση των επιμέρους υποπρογραμμάτων οδηγεί και στη λύση του αρχικού προβλήματος.

Ο τμηματικός προγραμματισμός μειώνει τα λάθη, επιτρέπει την ευκολότερη παρακολούθηση, κατανόηση και συντήρηση του προγράμματος από τρίτους.

Κάποια ερωτήματα που προκύπτουν στην ιεραρχική σχεδίαση είναι:

α. Σε πόσα υποπροβλήματα θα πρέπει διασπαστεί το αρχικό πρόβλημα;
β. Εάν το αρχικό πρόβλημα μπορεί να διασπαστεί πχ σε 3 υποπροβλήματα και τα 2 από αυτά μπορούν να διασπαστούν σε άλλα 3 το καθένα κοκ θα συνεχίσουμε τη διάσπαση μέχρι να μην μπορεί να γίνει άλλη;
γ. Ποιο κριτήριο θα ληφθεί υπόψη για να σταματήσουν οι διασπάσεις; πχ το πλήθος των γραμμών κώδικα που απαιτεί κάποιο υποπρόβλημα για να λυθεί αποτελεί κριτήριο για να σταματήσουμε τη διάσπαση;

Οι απαντήσεις σε όλα τα παραπάνω ερωτήματα δυστυχώς είναι ενδεικτικές και όχι ποσοτικά συγκεκριμένες. Κάποια κριτήρια για τη διάσπαση που μπορούν να ληφθούν υπόψη είναι:

- λειτουργικού τύπου, δηλαδή είναι σκόπιμο κάποιο υποπρόγραμμα να εκτελεί 1, 2 συγκεκριμένες ενδεχομένως και συναφείς λειτουργίες,
- ο διαχωρισμός έτσι ώστε να ευνοείται η επαναχρησιμοποίηση του κώδικά των υποπρογραμμάτων και σε άλλα προγράμματα,
- η αυτονομία που πρέπει να έχουν μεταξύ τους δηλαδή να μην είναι εξαρτώμενο το ένα από το άλλο
- η εμπειρία του προγραμματιστή

Ένα τέτοιο παράδειγμα ακολουθεί ώστε να δούμε ορισμένες από αυτές τις έννοιες στην πράξη.

6.13.1 Πρόβλημα τηλεφωνικών αριθμών

Κάθε φορά που σε μία ιστοσελίδα ενός ηλεκτρονικού καταστήματος ζητούνται τα στοιχεία επικοινωνίας του χρήστη συνήθως ανάμεσα σε αυτά είναι και το τηλέφωνο επικοινωνίας, κινητό ή σταθερό. Λάθος καταχώρηση ενός στοιχείου επικοινωνίας καθιστά προβληματική τη διεπαφή καταστήματος – πελάτη.

Να γίνει πρόγραμμα το οποίο θα δέχεται μία συμβολοσειρά και θα επιστέφει True/False ανάλογα με το εάν η συμβολοσειρά αυτή αποτελεί αποδεκτό τηλεφωνικό αριθμό της Ελλάδος ή όχι.

Κανόνες Τηλεφωνικών αριθμών της Ελλάδος

Το μήκος των αριθμών του σχεδίου αριθμοδότησης τηλεφωνικών αριθμών ορίζεται ως ακολούθως:

a) Το string πρέπει να αποτελείται μόνο από ψηφία δηλαδή οι αριθμοί δίνονται με τη μορφή 2106465781 και όχι 210-6465781 ή 210 6465781.
b) 10 ψηφία για τους αριθμούς που ξεκινούν με τα ψηφία 2, 6, 8, 9 με εξαίρεση εκείνους της σειράς 807 το μήκος των οποίων είναι 7 ψηφία,
c) 3, 4, 5 ή 6 ψηφία για εκείνους που ξεκινούν με το ψηφίο 1 και
d) 5 ψηφία για εκείνους που ξεκινούν με τα ψηφία 54

Ανάλυση της λύσης.

Από τον τρόπο της εκφώνησης το πρόβλημα έχει ήδη μερικώς διασπαστεί σε μικρότερα προβλήματα. Θα παρουσιαστούν δύο προσεγγίσεις διάσπασης του αρχικού προβλήματος σε ακόμη πιο απλά επί μέρους προβλήματα και για κάθε ένα από αυτά θα υλοποιηθεί για την επίλυσή του μία συνάρτηση. Στο τέλος σε μία συνάρτηση main θα κληθούν όλες τις συναρτήσεις που υλοποιήθηκαν. Για τα παρακάτω ερωτήματα να υλοποιηθεί από μία συνάρτηση η οποία:

1) Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν όλοι της οι χαρακτήρες είναι αριθμητικά ψηφία ή αλλιώς False.
2) Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν αποτελείται από 10 χαρακτήρες και το 1ο ψηφίο είναι 2, 6, 8, 9 ή αλλιώς False.
3) Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν αποτελείται από 7 χαρακτήρες και τα 3 πρώτα ψηφία είναι 807 ή αλλιώς False.
4) Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν αποτελείται από 3, 4, 5, 6 χαρακτήρες και το 1ο ψηφίο είναι 1 ή αλλιώς False.
5) Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν αποτελείται από 5 χαρακτήρες και οι δύο πρώτοι είναι 54 ή αλλιώς False.

Παράδειγμα 6.23

Η λύση που ακολουθείται στο παράδειγμα έχει αλγοριθμική προσέγγιση, πιο περιορισμένη χρήση των διαθέσιμων συναρτήσεων και των διαθέσιμων τελεστών των συμβολοσειρών. Για την επίλυση αυτή υλοποιούνται 5 + 1 συναρτήσεις.

Συντάξτε και εκτελέστε το παρακάτω πρόγραμμα:

```
def sub1(s):
    """
    Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν
    όλοι της οι χαρακτήρες είναι αριθμητικά ψηφία ή αλλιώς False"""
    for i in range(len(s)):
        if s[i] not in "0123456789":
            return False
    return True


def sub2(s):
    """
    Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν
```

```
    αποτελείται από 10 χαρακτήρες και το 1ο ψηφίο είναι 2, 6, 8, 9
    ή αλλιώς False """
    if len(s) == 10:
        if s[0] == '2' or s[0] == '6' or s[0] == '8' or s[0] == '9':
            return True
        else:
            return False
    else:
        return False


def sub3(s):
    """
    Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True
    εάν αποτελείται από 7 χαρακτήρες και τα 3 πρώτα ψηφία είναι 807
    ή αλλιώς False"""
    if len(s) == 7 and s[0:3] == "807":
        return True
    else:
        return False


def sub4(s):
    """
    Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True
    εάν αποτελείται από 3, 4, 5, 6
    χαρακτήρες και το 1ο ψηφίο είναι 1 ή αλλιώς False"""
    n = len(s)
    if (n == 3 or n == 4 or n == 5 or n == 6) and s[0] == "1":
        return True
    else:
        return False


def sub5(s):
    """
    Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν
    Αποτελείται από 5 χαρακτήρες και οι δύο πρώτοι
    είναι 54 ή αλλιώς False"""
    if len(s) == 5 and s[0:2] == "54":
        return True
    else:
        return False
```

```
def main(s):
    if (sub1(s) == True) and (
        sub2(s) == True or sub3(s) == True or sub4(s) == True or
        sub5(s) == True
    ):
        return True
    else:
        return False
```

Παράδειγμα 6.24

Στη 2η λύση που ακολουθεί, χρησιμοποιείται λιγότερο αλγοριθμική προσέγγιση και περισσότερο η χρήση του ισχυρού συντακτικού της γλώσσας Python και των διαθέσιμων τελεστών και των συναρτήσεων των συμβολοσειρών που διαθέτει η γλώσσα. Έτσι προκύπτει διαφορετική διάσπαση και κατανομή λειτουργιών στις συναρτήσεις και διαφορετική τεχνική χωρίς τη χρήση δομών επιλογής οπότε δεν προκύπτουν πολύπλοκες αλγοριθμικές δομές. Η ενδεικτική λύση και διάσπαση των λειτουργιών χρησιμοποιεί 2 + 1 συναρτήσεις χωρίς δομές επιλογής.

Συντάξτε και εκτελέστε το παρακάτω πρόγραμμα:

```
def sub_a(s):
    """
    Θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν
    όλοι της οι χαρακτήρες είναι αριθμητικά ψηφία ή αλλιώς False"""
    for i in range(len(s)):
        if s[i] not in '0123456789':
            return False
    return True


def sub_b_c_d(s):
    """
    Θα δέχεται ένα string και θα επιστρέφει True εάν
    b) έχει  10 ψηφία για τους αριθμούς που ξεκινούν με τα ψηφία 2, 6,
    8, 9 με εξαίρεση εκείνους της σειράς 807 το μήκος των οποίων είναι
    7 ψηφία ή
    c) 3, 4, 5 ή 6 ψηφία για εκείνους που ξεκινούν με το ψηφίο 1 ή
    d) 5 ψηφία για εκείνους που ξεκινούν με τα ψηφία 54
    """
    b = (s[0] in '2689' and len(s) == 10) or \
        (len(s) == 7 and s[0:3] == '807')
    c = s[0] == '1' and (str(len(s)) in '3456')
```

```
    d = s[0:2] == '54' and len(s) == 5
    return (b or c or d)

def main(s):
    return sub_a(s) and sub_b_c_d(s)
```

6.13.2 Δραστηριότητα

Η δεύτερη (2) ενδεικτική λύση υπερτερεί σε κομψότητα σε σχέση με την 1η, αλλά περιέχει ένα δύσκολο λογικό λάθος (bug). Δηλαδή σε σπάνιες περιπτώσεις και μόνο για ορισμένες τιμές συμβολοσειρών που δεν αποτελούν τηλεφωνικούς αριθμούς θα επιστρέψει True.

- μπορείτε να βρείτε ποιες είναι οι τιμές αυτές και
- πως μπορεί να διορθωθεί το λάθος αυτό;

6.14 Πρόβλημα αντικατάστασης κειμένου

Πολύ συχνή εργασία σε επίπεδο εφαρμογών γραφείου είναι η αντικατάσταση μίας υποσυμβολοσειράς εντός ενός κειμένου με μία άλλη. Για παράδειγμα θα μπορούσε να είναι ένα πρότυπο εγγράφου κειμένου στο οποίο ζητείται αλλαγή στο τμήμα των υπογραφών ώστε το όνομα του παλαιού Διευθυντή και να αντικατασταθεί με αυτό του νέου ή στο τμήμα της επικοινωνίας ένα τηλέφωνο με ένα άλλο κλπ.

Αρχικά φαίνεται απλό διότι όλοι οι συντάκτες και οι επεξεργαστές κειμένου διαθέτουν αυτή τη λειτουργία γνωστή ως (Εύρεση και Αντικατάσταση: Find and Replace). Έτσι οι ενέργειες που έχει να κάνει κάποιος υπάλληλος για να εκτελέσει αυτή τη λειτουργία σε ένα και μόνο έγγραφο είναι:

- Να μετακινηθεί στον αντίστοιχο φάκελο στο σκληρό δίσκο
- Να ανοίξει το έγγραφο
- Να καλέσει τη λειτουργία εύρεσης αντικατάστασης
- Να συμπληρώσει το παλαιό όνομα για αναζήτηση
- Να συμπληρώσει το νέο όνομα για αντικατάσταση
- Να εκτελέσει τη λειτουργία
- Να αποθηκεύσει το νέο κείμενο
- Να κλείσει το αρχείο

Οι ενέργειες αυτές δεν θα απαιτούσαν στην καθημερινότητα πάνω από 1 λεπτό από έναν υπάλληλο για ένα έγγραφο. Φανταστείτε τώρα ο φάκελος του δίσκου να περιέχει 30 τέτοια πρότυπα εγγράφων και ο δίσκος να περιέχει και άλλους 30 φακέλους από 30 περίπου έγγραφα ο καθένας και οι παραπάνω λειτουργίες να πρέπει να εκτελεστούν σε όλα τα αρχεία.

Ο χρόνος που θα απαιτούνταν για ένα υπάλληλο θα ήταν 30 x 30 = 900 αρχεία x 1 λεπτό = 900 λεπτά / 60 = 15 ώρες. Θα χρειαζόταν λοιπόν χωρίς διαλλείματα καθυστερήσεις ή άλλες ασχολίες 1 ½ ημέρα μονότονης εργασίας. Αυτό χωρίς να ληφθεί υπόψη η πιθανότητα ο υπάλληλος να κάνει κάποιο λάθος ή να παραλείψει κάποια αρχεία.

Τέτοιες λειτουργίες είναι σκόπιμο να ανατίθενται σε κάποιο λογισμικό το οποίο θα τις επιτελέσει πιο γρήγορα και χωρίς λάθη.

Παράδειγμα 6.25

(***) Ολόκληρο το σενάριο που αναφέρθηκε θα επιλυθεί σταδιακά και σε επόμενα κεφάλαια μπορείτε να το βελτιώσετε μαθαίνοντας νέες τεχνικές και εργαλεία ώστε να γίνονται οι εργασίες των 15 ωρών σε μερικά μόνο δευτερόλεπτα.

Προς το παρόν εδώ θα αντιμετωπιστεί ΜΟΝΟ ένα τμήμα του προβλήματος, αυτό της εύρεσης και αντικατάστασης σε μία συμβολοσειρά ενός τμήματός της με ένα άλλο.

Ανάλυση της λύσης

Ας υποθέσουμε ότι στην παρακάτω συμβολοσειρά:

```
string = 'aaaabbbcccdddeeeeebbbbbc'
```

ζητείται η αντικατάσταση του τμήματος bb με το xxx. Παρατηρείται ότι αυτό το τμήμα βρίσκεται σε τρία (3) σημεία στις παρακάτω θέσεις:

```
string = 'aaaabbbcccdddeeeeebbbbbc'
```

Έτσι **η νέα συμβολοσειρά** μετά την αντικατάσταση (θυμίζουμε τα αντικείμενα συμβολοσειράς δεν μπορούν να τροποποιηθούν) θα πρέπει να γίνει:

```
new_string = 'aaaaxxxbcccdddeeeeexxxxxxbc'
```

1.1. Ξεκινώντας από την αρχή της συμβολοσειράς δημιουργούνται ανά ένα χαρακτήρα τμήματα συμβολοσειρών ίσα με το μήκος του τμήματος που ζητείται να αλλάξουμε. Αυτό επαναλαμβάνεται όχι μέχρι να τελειώσουν οι χαρακτήρες της συμβολοσειράς αλλά μέχρι μην περισσεύουν αρκετοί χαρακτήρες για να δημιουργηθεί το προς αναζήτηση τμήμα (bb). Έτσι θα δημιουργηθούν τα παρακάτω τμήματα:

```
aa,aa,aa,ab,bb,bc,cc,cc,cd,dd,dd,de,ee,ee,ee,ee,eb,bb,bb,bc,
```

1.2. Κάθε φορά που δημιουργείται ένα τέτοιο τμήμα το συγκρίνεται με αυτό που αναζητείται (bb)

Οι επιτυχείς αναζητήσεις θα συμβούν εδώ:

```
aa,aa,aa,ab,bb,bc,cc,cc,cd,dd,dd,de,ee,ee,ee,ee,eb,bb,bb,bc,
```

1.3. Οπότε στην περίπτωση επιτυχούς αναζήτησης, στη νέα συμβολοσειρά τοποθετείται το νέο τμήμα (δηλαδή το xxx στο παράδειγμά μας).

1.4. Αλλιώς τοποθετείται στη νέα συμβολοσειρά ο πρώτος χαρακτήρας του τμήματος που δημιουργήθηκε.

2. Τελειώνοντας οι χαρακτήρες που έχουν περισσέψει από την παλαιά συμβολοσειρά και δεν είναι αρκετοί για να δημιουργηθεί τμήμα τοποθετούνται στη νέα.

```
def replace(string, old, new):
    # n το μέγεθος του string που ψάχνω
    n = len(old)
    # new_string το νεο string μετά την αντικατάσταση
    new_string = ""
    i = 0
    # Επανάληψη από την αρχή μέχρι n χαρακτήρες πριν το τέλος
    while i <= len(string) - n:
        # Φτιάξε τμήματα n - χαρακτήρων string[i:i+n]
        # και σύγκρινέ τα με αυτό που ψάχνεις
        if string[i:i+n] == old:
            # σε περίπτωση επιτυχούς αναζήτησης
            # βάλε το νέο τμήμα
            new_string += new
            i += n
        else:
```

```
            # αλλιώς βάλε τον 1ο χαρακτήρα του τμήματος
            # και προχώρα στον επόμενο
            new_string += string[i]
            i += 1
    # επέστρεψε το new_string και ότι περισσεύει
    return new_string + string[i:]
```

Χρήση της συνάρτηση που υλοποιήθηκε σεε μία συμβολοσειρά πολλαπλών γραμμών (multiline string) που περιέχει το zen της Python.

```
zen = """
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!
"""
>>> print (replace(zen, 'is', 'are'))
```

Το αποτέλεσμα της εκτέλεσης είναι:

```
Beautiful are better than ugly.
Explicit are better than implicit.
Simple are better than complex.
Complex are better than complicated.
Flat are better than nested.
Sparse are 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 are better than never.
Although never are often better than *right* now.
If the implementation are hard to explain, it's a bad idea.
If the implementation are easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

6.14.1 Εύρεση και αντικατάσταση, μέθοδος replace()

Η λειτουργία της εύρεσης και αντικατάστασης είναι πολύ συχνή όταν πραγματοποιούνται χειρισμοί κειμένου μέσα από κώδικα γι' αυτό η Python διαθέτει την ενσωματωμένη μέθοδο `replace()` που επιτελεί τις παραπάνω λειτουργίες.

- `<string>.replace(old, new, count)` :Επιστρέφει νέα συμβολοσειρά όπου έχουν αντικατασταθεί `count` (εάν δεν βάλουμε τιμή γίνονται όλες οι αντικαταστάσεις) τμήματα `old` με το `new`

```
>>> '11234536613'.replace('1','xx',2)
'xxxx234536613'
>>> '11234536613'.replace('1','xx')
'xxxx2345366xx3'
```

6.14.2 Δραστηριότητα

(**)Στον ΣΥΝΔΕΣΜΟ θα βρείτε σε μορφή κειμένου utf ένα πολύ γνωστό μυθιστόρημα, επιλέξτε το 1ο κεφάλαιο ως τμήμα κειμένου (πολλών γραμμών) και αντιγράψτε το. Στη συνέχεια δημιουργήστε μία συμβολοσειρά πολλαπλών γραμμών στην Python (Multiline String).

Στη συνέχεια με χρήση της γλώσσας Python να δημιουργήσετε ένα νέο κείμενο στο οποίο:

α. Να αντικαταστήσετε όλες τις λέξεις rabbit είτε με πεζά είτε με κεφαλαία σε goat.
β. Το όνομα της ηρωίδας Alice σε Hannah
γ. Να αφαιρέσετε όλα τα σύμβολα εκτός από το κενό.
δ. Όλα τα σημεία στίξης να γίνουν τελείες.
ε. Η αλλαγή γραμμής να διατηρηθεί ως έχει.

6.15 Να θυμάμαι

Δομή Δεδομένων

Δομή δεδομένων (Data Structure) στην πληροφορική είναι ένας τρόπος αποθήκευσης και οργάνωσης δεδομένων με τέτοιο τρόπο ώστε να εκτελούνται αποδοτικά διάφορες λειτουργίες σε αυτά. Το ποια δομή θα επιλέξει ο προγραμματιστής να χρησιμοποιήσει για τα δεδομένα του θα επηρεάσει και τις επιτρεπόμενες λειτουργίες, τα εργαλεία και τεχνικές που θα μπορεί να χρησιμοποιήσει και τελικά και το πρόγραμμα επίλυσης του προβλήματός του.

Χαρακτηριστικά Συμβολοσειρών

Οι συμβολοσειρές στην Python αποτελούν και αυτές μία δομή δεδομένων. Διαθέτουν εσωτερική **ακολουθιακή δομή**(sequence), δηλαδή μπορούμε να χρησιμοποιήσουμε δείκτες για να αναφερθούμε στα στοιχεία τους. Είναι **διασχίσιμες** ακολουθίες (iterable sequences) χαρακτήρων κειμένου. Έτσι μπορούμε πχ με τη δομή επανάληψης να προσπελάσουμε όλα τα στοιχεία τους ένα - ένα. Είναι **αμετάβλητες** δομές, (μη τροποποιήσιμες, μη μεταβαλλόμενες: immutable). Δηλαδή δεν μπορούμε να τροποποιήσουμε τους χαρακτήρες τους, όπως και να προσθέσουμε ή να αφαιρέσουμε χαρακτήρες σε αυτές.

Αντικείμενα

Ένα αντικείμενο σε μία γλώσσα προγραμματισμού είναι μία οντότητα η οποία εμπεριέχει **δεδομένα** και **λειτουργικότητα.** Στη γλώσσα Python συγκεκριμένα, όλα τα δεδομένα που συναντήσαμε έως τώρα είτε αφορούσαν σε ακεραίους αριθμούς, είτε σε αριθμούς κινητής υποδιαστολής, είτε σε λογικά δεδομένα, είτε σε συμβολοσειρές αποτελούν αντικείμενα (objects). Κάθε αντικείμενο έχει μία **ταυτότητα** (id), ένα **τύπο** (type/ κλάση/ τάξη), και κάποια **τιμή** και αποδίδουμε σε αυτό κάποιο όνομα για να μπορούμε να το χειριστούμε μέσα από το πρόγραμμά μας.

Δομή και δείκτες συμβολοσειράς

Η συμβολοσειρά διαθέτει εσωτερική δομή, έτσι κάθε χαρακτήρας της αντιστοιχεί σε έναν αριθμό (δείκτη) ο οποίος συνάδει με τη θέση του χαρακτήρα εντός της συμβολοσειράς. Η αρίθμηση των δεικτών γίνεται με δύο τρόπους:

- με θετικούς ακεραίους από αριστερά (αρχή), ξεκινώντας από το 0
- με αρνητικούς ακεραίους από δεξιά (τέλος), ξεκινώντας από το -1.

Pos Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
text=	**i**		**l**	**o**	**v**	**e**		**p**	**y**	**t**	**h**	**o**	**n**	**.**
Pos index	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Τελεστές Συμβολοσειράς

- `+` : Τελεστής συνένωσης(concatenation) συμβολοσειρών, χρησιμοποιείται για να ενώσουμε το περιεχόμενο, δύο ή περισσοτέρων συμβολοσειρών.
- `*` : Τελεστής επανάληψης συμβολοσειρών, χρησιμοποιείται για να επαναλάβουμε το περιεχόμενο μίας συμβολοσειράς.
- `+=` : Τελεστής ο οποίος συνενώνει και δημιουργεί νέο αντικείμενο συμβολοσειράς
- `[ i ]`: Τελεστής θέσης επιστρέφει το χαρακτήρα της i-θέσης.
- `[ : ]`: Τελεστής διαμέρισης (slicing) συμβολοσειράς, χρησιμοποιείται για να πάρουμε ένα τμήμα της συμβολοσειράς (ο χαρακτήρας τέλους δεν περιλαμβάνεται, δηλαδή το διάστημα αρχή : τέλος είναι ανοικτό από τα αριστερά). Προκαθορισμένο βήμα (όταν δεν βάλουμε κάποιο άλλο) είναι το ένα (1).
- `in` : Τελεστής ύπαρξης στοιχείου, χρησιμοποιείται για να ελέγξουμε την ύπαρξη ενός χαρακτήρα ή μίας συμβολοσειράς σε μία άλλη.
- `not in` : Τελεστής μη ύπαρξης στοιχείου, χρησιμοποιείται για να ελέγξουμε τη ΜΗ ύπαρξη ενός χαρακτήρα ή μίας συμβολοσειράς σε μία άλλη.
- `<, >, !=, ==` : Σχεσιακοί Τελεστές για συγκρίσεις μεταξύ συμβολοσειρών. Η Python για να εκτελέσει τις συγκρίσεις μεταξύ των συμβολοσειρών μετατρέπει κάθε χαρακτήρα της συμβολοσειράς στον αντίστοιχο αριθμητικό κώδικα σύμφωνα με τη λειτουργία της `ord` που περιγράψαμε προηγουμένως και συγκρίνει έναν προς έναν τους αριθμούς αυτούς, όταν βρει ίδιους προχωράει στον επόμενο μέχρι να μπορέσει να αποτιμηθεί το αποτέλεσμα.

Κώδικες

Οι υπολογιστές χρησιμοποιούν πίνακες (κώδικες) μετατροπής χαρακτήρων σε αριθμούς προκειμένου να αναπαραστήσουν τα γράμματα, τα σύμβολα, τα αριθμητικά ψηφία κλπ. από τα οποία αποτελείται μία συμβολοσειρά. Οι πιο γνωστοί τέτοιοι κώδικες είναι ο κώδικας ASCII και ο UNICODE.

Αναζήτηση

Η λειτουργία της αναζήτησης (searching) χρησιμοποιείται ώστε να λαμβάνεται η πληροφορία ύπαρξης ή όχι, μίας τιμής (συχνά ονομάζεται κλειδί: key,) σε μία συλλογή δεδομένων. Η πληροφορία αυτή μπορεί να αφορά: α) στο εάν υπάρχει ή όχι η τιμή αυτή, β) σε ποια θέση βρίσκεται ή γ) εάν η τιμή επαναλαμβάνεται πολλές φορές στη συλλογή, σε όλες τις θέσεις που βρίσκεται αυτή. Οι βασικοί αλγόριθμοι αναζήτησης είναι η σειριακή (sequential search) αναζήτηση και η δυαδική αναζήτηση (binary search) η οποία μπορεί να υλοποιηθεί μόνο σε ταξινομημένη δομή.

Ιεραρχική σχεδίαση και τμηματικός προγραμματισμός

Η ιεραρχική σχεδίαση επίλυσης ενός σύνθετου συνήθως προβλήματος περιλαμβάνει τη διαδικασίας διάσπασής του σε μικρότερα προβλήματα (και άρα απλούστερα) τα οποία είναι πιο εύκολο να επιλυθούν η λύση των οποίων επιλύει και το αρχικό πρόβλημα. Η τεχνική αυτή σχεδίασης ονομάζεται και από επάνω προς τα κάτω (top down design). Υλοποιείται με τον τμηματικό προγραμματισμό όπου επιλύεται κάθε υποπρόβλημα με ένα υποπρόγραμμα (στην Python με συνάρτηση). Η σύνθεση των επιμέρους υποπρογραμμάτων οδηγεί και στη λύση του αρχικού προβλήματος. Ο τμηματικός προγραμματισμός μειώνει τα λάθη, επιτρέπει την ευκολότερη παρακολούθηση, κατανόηση και συντήρηση του προγράμματος από τρίτους.

Συναρτήσεις Συμβολοσειράς

`len(string)` : Επιστρέφει το μήκος της συμβολοσειράς

`ord(char)` : Επιστρέφει τη θέση του char στον πίνακα Unicode

`chr(pos)` : Επιστρέφει το χαρακτήρα Unicode που βρίσκεται στη θέση pos

`str(value)` : Μετατρέπει την τιμή value σε αντικείμενο τύπου string

`max(string)` : Επιστρέφει το χαρακτήρα με την μεγαλύτερη ord τιμή της συμβολοσειράς string

`min(string)` : Επιστρέφει το χαρακτήρα με την μικρότερη ord τιμή της συμβολοσειράς string

Μέθοδοι Συμβολοσειράς

`.upper()` : Επιστρέφει συμβολοσειρά με τα γράμματα του αντικειμένου σε κεφαλαίους

`.lower()` : Επιστρέφει συμβολοσειρά με τα γράμματα του αντικειμένου σε πεζούς

`.title()` : Επιστρέφει συμβολοσειρά με το πρώτο γράμμα κάθε λέξης του αντικειμένου σε κεφαλαίο και όλους τους υπόλοιπους σε πεζούς

`.find(sub, start, end)` : Επιστρέφει την πρώτη θέση της sub, στο αντικείμενο ξεκινώντας από τη θέση start μέχρι τη θέση end

`.count(sub, start, end)` : Επιστρέφει πόσες φορές εμφανίζεται η sub εντός της συμβολοσειράς (string) ξεκινώντας από τη θέση start μέχρι τη θέση end

`.replace()` : Επιστρέφει νέα συμβολοσειρά όπου έχουν αντικατασταθεί count αντικαταστάσεις του τμήματος old με το new

`.join(δομή)` : Συνενώνει όλα τα στοιχεία της δομής (πχ λίστας, πλειάδας, λεξικού) επιστρέφοντας μία ενιαία συμβολοσειρά . Τα στοιχεία της ακολουθίας πρέπει να είναι τύπου string, ως διαχωριστικό μεταξύ των στοιχείων χρησιμοποιείται το αντικείμενο.

`.split(διαχωριστικό)` : Διαχωρίζει τα στοιχεία του αντικειμένου επιστρέφοντας μια λίστα των στοιχείων αυτών. Ο διαχωρισμός των στοιχείων επιτυγχάνεται με το διαχωριστικό χαρακτήρα.

`.strip()` : Επιστρέφει συμβολοσειρά αφού έχει αφαιρέσει όλους τους λευκούς χαρακτήρες από αριστερά και δεξιά του αντικειμένου

`.lstrip()` : Επιστρέφει συμβολοσειρά αφού έχει αφαιρέσει όλους τους λευκούς χαρακτήρες από αριστερά του αντικειμένου

`.rstrip()` : Επιστρέφει συμβολοσειρά αφού έχει αφαιρέσει όλους τους λευκούς χαρακτήρες από δεξιά του αντικειμένου

`.capitalize()` : Επιστρέφει το αντικείμενο με το πρώτο γράμμα σε κεφαλαίο και όλα τα υπόλοιπα σε πεζά.

`.isalpha()` : Επιστρέφει True εάν όλοι οι χαρακτήρες του αντικειμένου είναι γράμματα αλφαβήτου

`.isalnum()` : Επιστρέφει True εάν όλοι οι χαρακτήρες του αντικειμένου είναι αλφαριθμητικοί

`.center(width)` : Επιστρέφει τους χαρακτήρες του αντικειμένου στοιχισμένους στο κέντρο τους πλάτους width

`.ljust(width)` : Επιστρέφει τους χαρακτήρες του αντικειμένου στοιχισμένους αριστερά στο πλάτος width

`.rjust(width)` : Επιστρέφει τους χαρακτήρες του αντικειμένου στοιχισμένους δεξιά στο πλάτος width

6.16 Ερωτήσεις Κατανόησης

1. Οι συμβολοσειρές στη γλώσσα Python είναι διασχίσιμες όπως οι λίστες ΝΑΙ ΟΧΙ
2. Οι συμβολοσειρές στην Python είναι τροποποιήσιμες όπως οι λίστες ΝΑΙ ΟΧΙ
3. Τα αντικείμενα είναι οντότητες με δεδομένα και λειτουργικότητα ΝΑΙ ΟΧΙ
4. Τα αντικείμενα έχουν ταυτότητα, ΑΜΚΑ και ΑΦΜ ΝΑΙ ΟΧΙ
5. Ο κώδικας ASCII περιέχει όλα τα γράμματα όλων των γλωσσών ΝΑΙ ΟΧΙ
6. Οι συμβολοσειρές στην Python ακολουθούν την κωδικοποίηση UNICODE ΝΑΙ ΟΧΙ
7. Οι αλγόριθμοι αναζήτησης είναι η δυαδική και η τριαδική ΝΑΙ ΟΧΙ
8. Η ιεραρχική σχεδίαση πρέπει να αποφεύγεται γιατί περιπλέκει τη διαδικασία επίλυσης ενός προβλήματος ΝΑΙ ΟΧΙ
9. Για να δημιουργήσω ένα αντικείμενο τύπου string με αναφορά σε αυτό `name` και περιεχόμενο Greece μπορώ να κάνω τα παρακάτω;

   α. `name ← 'Greece'` ΝΑΙ ΟΧΙ

   β. `name → 'Greece'` ΝΑΙ ΟΧΙ

   γ. `name = 'Greece'` ΝΑΙ ΟΧΙ

   δ. `name = "Greece"` ΝΑΙ ΟΧΙ

   ε. `name = str("Greece")` ΝΑΙ ΟΧΙ

   στ. `name = "  Greece"` ΝΑΙ ΟΧΙ

   ζ. `name = "Greece   ".rstrip()` ΝΑΙ ΟΧΙ

   η. `name =  'G' + '\nr'.strip() + 2 * 'e' + 'ce'` ΝΑΙ ΟΧΙ

   θ. `name = 'g'.upper() + ' r'.lstrip() +  'EE'.lower() + 'ec'[::-1]` ΝΑΙ-ΟΧΙ

   ι. `name = 'g'.upper() + ' r'.lstrip() +  chr(101) + chr(ord('e')) + 'process'[3:5]` ΝΑΙ-ΟΧΙ

10. Το παρακάτω αντικείμενο συμβολοσειράς `text` έχει μήκος `len(text)`:

    α. `text = '012345'` `len(text) = ___`

    β. `text = '012345'[-1]` `len(text) = ___`

    γ. `text = '012345'[-2]` `len(text) = ___`

    δ. `text = '012345'[:2]` `len(text) = ___`

    ε. `text = '012345'[1:3]` `len(text) = ___`

    στ. `text = '012345'[:]` `len(text) = ___`

ζ. `text = '012345'[::2]` `len(text) = ___`

η. `text = '012345'[:]` `len(text) = ___`

θ. `text = '012345'[-1:-3:-1]` `len(text) = ___`

ι. `text += text` `len(text) = ___`

11. Το αποτέλεσμα των παρακάτω εντολών είναι `True` ή `False`;

α. `'0' < '1'` `True` `False`

β. `'α' >= 'αβ'` `True` `False`

γ. `'α' < 'αα'` `True` `False`

δ. `len(str(123)) == len(3 * '3')` `True` `False`

ε. `'a'.upper() == 'A'.lower()` `True` `False`

στ. `'ABC'[-3] == 'a'.upper()[0]` `True` `False`

ζ. `'a'[-1] + 'a '[1].strip() == 'AAaa'[-2:]` `True` `False`

η. `str('a') + str('b') < 'AB'` `True` `False`

θ. `len('\n') + len('1') == 2` `True` `False`

ι. `int(max('123')) == 10 // 3` `True` `False`

12. Το αποτέλεσμα των παρακάτω εντολών είναι `True` ή `False`;

α. `'αβγδεζηθικ'.find('ε') == 5` `True` `False`

β. `'123123'.find('1') == 1` `True` `False`

γ. `'123123'.find('1') in [0, 1]` `True` `False`

δ. `'123123'.find('1', 1) == 3` `True` `False`

ε. `'123123'.find('4') == 0` `True` `False`

στ. `'abcdabcdabcd'.count('a') == 3` `True` `False`

ζ. `'abcdabcdabcd'.count('e') == 0` `True` `False`

η. `'axax'.replace('a', 'x') == 'xx'` `True` `False`

θ. `'aaaxx'.replace('a', 'xx') == 2 ** 3 * 'x'` `True` `False`

ι. `''.join(['1', '2', '3']) == '123'` `True` `False`

ια. `'-'.join(['1', '2', '3']) == '1-2-3-'` `True` `False`

ιβ. `'a b c'.split() == ['a', 'b', 'c']` `True` `False`

ιγ. `'1 2 3'.split() == [1, 2, 3]` True False

6.17 Ερωτήσεις Ανάπτυξης

1. Τι ονομάζουμε Δομή Δεδομένων σε μία γλώσσα προγραμματισμού;
2. Να αναφέρεται και να εξηγήσετε 3 χαρακτηριστικά των συμβολοσειρών στη γλώσσα Python.
3. Τι είναι τα αντικείμενα (objects) σε μία γλώσσα προγραμματισμού;
4. Να αναφέρετε τα χαρακτηριστικά των αντικειμένων;
5. Εξηγήστε πως λειτουργεί η εσωτερική δομή και οι δείκτες στα αντικείμενα των συμβολοσειρών στη γλώσσα Python;
6. Να αναφέρετε τους τελεστές των συμβολοσειρών στη γλώσσα Python.
7. Ποια η λειτουργία του τελεστή των συμβολοσειρών `+` στη γλώσσα Python;
8. Ποια η λειτουργία του τελεστή των συμβολοσειρών `*` στη γλώσσα Python;
9. Ποια η λειτουργία του τελεστή των συμβολοσειρών `:` στη γλώσσα Python;
10. Έχουν ισχύ οι σχεσιακοί τελεστές στις συμβολοσειρές; Εάν ναι, εξηγήστε τον τρόπο λειτουργίας και σύγκρισης 2 συμβολοσειρών μεταξύ τους.
11. Ποιους αλγόριθμους αναζήτησης γνωρίζετε; Ποιος από αυτούς λειτουργεί σε ταξινομημένη δομή δεδομένων;
12. Τι σημαίνει ο όρος dot notation (σημειογραφία τελείας); Δώστε ένα παράδειγμα.

6.18 Ασκήσεις

Να λύσετε τις δραστηριότητες:

6.2.2

6.2.4

6.3.1

6.4.1

6.4.2

6.5.1

6.6.1

6.8.2

6.10.2

6.13.2

6.14.2

1. Να επιλύσετε το πρόβλημα της παραγράφου 6.9.1 των προσωρινών διαπιστευτηρίων χρησιμοποιώντας την τεχνική της ιεραρχικής σχεδίασης
2. Να υλοποιήσετε τις δικές σας συναρτήσεις οι οποίες εκτελούν τις ίδιες λειτουργίες με τις παρακάτω μεθόδους των συμβολοσειρών της Python. Για τις μεθόδους που δεν έχουν αναφερθεί στο κεφάλαιο αυτό αναζητήστε βοήθεια με τη συνάρτηση help της python, πχ `help(str.ljust)`
    α. capitalize
    β. title
    γ. upper, lower
    δ. isupper, islower
    ε. isnumeric
    στ. isalpha
    ζ. ljust, rjust
    η. center
    θ. count
3. Να υλοποιήσετε τη συνάρτηση **is_date** η οποία θα δέχεται μία συμβολοσειρά και θα επιστρέφει True εάν αυτή αποτελεί έγκυρη ημερομηνία ή False σε διαφορετική περίπτωση. Θεωρήστε ότι αποδεκτοί τύποι ημερομηνίας είναι οι παρακάτω: dd/mm/yyy, dd/mm/yyyy, ddmmyy.
4. Χωρίς τη χρήση της συνάρτησης len να υλοποιήσετε τη δική σας συνάρτηση **my_len** η οποία θα δέχεται μία συμβολοσειρά και θα επιστρέφει το μήκος της.
5. Να υλοποιήσετε μία συνάρτηση **count_vow_cons** η οποία θα δέχεται μία συμβολοσειρά και θα επιστρέφει το πλήθος των φωνηέντων και των συμφώνων (επιλέξτε εσείς σε ποια γλώσσα ελλ/λατιν επιθυμείτε).

6. Να υλοποιήσετε τη συνάρτηση **extract** η οποία θα δέχεται μία συμβολοσειρά και ένα χαρακτήρα και θα επιστρέφει μία νέα συμβολοσειρά η οποία ΔΕΝ θα περιέχει το χαρακτήρα αυτό. Με ποια ενσωματωμένη μέθοδο και πως θα μπορούσε να υλοποιηθεί η παραπάνω λειτουργία;
7. Να υλοποιήσετε τη συνάρτηση **unique** η οποία θα δέχεται ένα κείμενο και θα επιστρέφει όλους τους διαφορετικούς χαρακτήρες που χρησιμοποιούνται σε αυτό.
8. Να υλοποιήσετε μία συνάρτηση **letter_freq** η οποία θα δέχεται μία συμβολοσειρά (επιλέξτε εσείς σε ποια γλώσσα ελλ/λατιν επιθυμείτε, πεζά ή κεφαλαία ή και τα δύο) και θα επιστρέφει ένα πίνακα συχνοτήτων εμφάνισης των γραμμάτων του αλφαβήτου στη συμβολοσειρά αυτή.
9. Να υλοποιήσετε μία συνάρτηση με όνομα **is_amka** η οποία θα δέχεται ένα string και θα επιστρέφει True εάν είναι αποδεκτός αριθμός μητρώου κοινωνικής ασφάλισης και False εάν δεν είναι. Θεωρήστε ότι για να είναι αποδεκτή μία συμβολοσειρά ως ΑΜΚΑ πρέπει:
    α. Να έχει 11 χαρακτήρες
    β. Να είναι όλοι οι χαρακτήρες της ψηφία
    γ. Οι πρώτοι 6 χαρακτήρες να είναι έγκυρη ημερομηνία της μορφής ddmmyy
10. Να υλοποιήσετε μία συνάρτηση **max_word** η οποία θα δέχεται ένα κείμενο με μορφή συμβολοσειράς όπου οι λέξεις θα είναι διαχωρισμένες με ένα κενό χαρακτήρα και δεν θα υπάρχει άλλο σημείο στίξης και θα επιστρέφει τη λέξη με το μεγαλύτερο μήκος.
11. Να επεκτείνετε την προσέγγιση της λύσης στο πρόβλημα της στοίχισης μίας πρότασης στο παράδειγμα 6.11 ώστε η συνάρτηση **align** να μπορεί να δέχεται και να στοιχίζει κεντρικά μία ολόκληρη παράγραφο (multiline string)
12. Να υλοποιήσετε συνάρτηση με όνομα **extract_email** η οποία θα δέχεται ένα κείμενο το οποίο θα περιέχει ανάμεσα στα άλλα και κάποια δνση ηλ. ταχυδρομείου της μορφής: `<κενό>aaaa..@bbb...b.xx<κενό>` και να διαχωρίζει την δνση αυτή από το υπόλοιπο κείμενο επιστρέφοντάς την ως συμβολοσειρά.
13. Πολύ συχνά όταν γράφουμε κείμενο που περιέχει ελληνικούς χαρακτήρες (ειδικά κεφαλαίους) εάν δεν γυρίσει το πληκτρολόγιο στην ελληνική γλώσσα μπορεί να πληκτρολογηθούν και λατινικοί χαρακτήρες μέσα σε αυτό. Όταν πρόκειται για χαρακτήρες που είναι στο ίδιο πλήκτρο του πληκτρολογίου και διαφέρουν οπτικά πχ (U-Θ) (R-P) μπορεί να γίνει εύκολα διόρθωση. Όταν όμως πρόκειται για χαρακτήρες που είναι όμοιοι (E, T, Y, I, O, A, H, K, Z, X, B, N, M) η οπτική διόρθωση είναι ανέφικτη. Να υλοποιήσετε τη συνάρτηση **purify** η οποία θα δέχεται μία συμβολοσειρά ελληνικών κεφαλαίων γραμμάτων και εάν βρεθεί κάποιος λατινικός χαρακτήρας σε αυτή θα τον αντικαθιστά με τον αντίστοιχο ελληνικό που βρίσκεται στο ίδιο πλήκτρο και θα επιστρέφει τη νέα συμβολοσειρά.
14. Ο Κώδικας του Καίσαρα είναι μία από τις απλούστερες και πιο γνωστές τεχνικές κρυπτογράφησης. Είναι ένας κώδικας αντικατάστασης στον οποίο κάθε γράμμα του αρχικού κειμένου αντικαθίσταται από κάποιο άλλο γράμμα με σταθερή απόσταση κάθε φορά στο αλφάβητο (βλ εδώ). Να δημιουργήσετε τις συναρτήσεις:
    α. Συνάρτηση **encode** η οποία θα δέχεται ένα κείμενο και τη μετατόπιση (offset) και θα επιστρέφει την κωδικοποιημένη συμβολοσειρά
    β. Συνάρτηση **decode** η οποία θα δέχεται το κωδικοποιημένο κείμενο και τη μετατόπιση και θα επιστρέφει την αρχική συμβολοσειρά.

Σημείωση: Το κείμενο προς κωδικοποίηση θεωρήστε ότι περιέχει μόνο κεφαλαία λατινικά γράμματα

Κεφάλαιο 7

7. Ενσωματωμένες σύνθετες δομές δεδομένων

Τελειώνοντας αυτό το μάθημα θα έχεις μάθει

- Τί είναι μια σύνθετη δομή δεδομένων στην Python και ποιες είναι οι ενσωματωμένες σύνθετες δομές σε αυτή
- Τί είναι συλλογή και τί ακολουθία
- Τί είναι οι πλειάδες και πώς τις χρησιμοποιούμε
- Τί είναι οι λίστες και γιατί είναι από τις πιο σημαντικές δομές στην Python
- Τι είναι λεξικό και πώς το χρησιμοποιούμε
- Τί είναι σύνολο και πώς το χρησιμοποιούμε
- Τους τύπους συνόλων στην Python και το λόγο ύπαρξης τους
- Την αλγοριθμική δομή της εξαίρεσης και τη χρήση της
- Τις συναρτήσεις που εφαρμόζονται στις συλλογές και τις ακολουθίες
- Τις μεθόδους κάθε ενσωματωμένης δομής δεδομένων

7.1 Σύνθετες δομές δεδομένων

Στην προηγούμενη ενότητα είδαμε μία σύνθετη δομή δεδομένων τη συμβολοσειρά. Στην Python έχουμε αρκετές ακόμα ενσωματωμένες (δηλαδή ορισμένες από την ίδια τη γλώσσα) σύνθετες δομές δεδομένων. Στη συμβολοσειρά τα στοιχεία που τη συνθέτουν είναι άλλες συμβολοσειρές που στην τελική ανάλυση, στις αριθμημένες «θέσεις», έχουν μέγεθος ένα – είναι με άλλα λόγια χαρακτήρες. Επιπλέον η συμβολοσειρά είναι μη μετατρέψιμη (immutable) στην Python, με άλλα λόγια δεν είναι φτιαγμένη για να τροποποιείται, ενώ η θέση μέσα στη συμβολοσειρά έχει σημασία, με άλλα λόγια η δομή είναι αυτό που ονομάζουμε ακολουθία. Αλλάζοντας οποιαδήποτε από αυτές τις ιδιότητες έχουμε μια διαφορετική, ιδιαίτερα χρήσιμη δομή. Παίρνουμε με αυτό τον τρόπο την πλειάδα, τη λίστα και το σύνολο.

Μια ακόμα ενδιαφέρουσα δομή που όμως αντιστοιχίζει ζευγάρια συμβόλων ή κλειδιών και τιμών είναι το λεξικό. Δε χρειάζεται κανείς να επιχειρηματολογήσει πόσο διαδεδομένα και χρήσιμα είναι τα λεξικά πέρα από τις εφαρμογές. Εδώ θα δούμε πώς αξιοποιούνται και εντός του πλαισίου μιας εφαρμογής.

Στη γλώσσα Python, όλα είναι αντικείμενα. Αυτό περιλαμβάνει και τις συμβολοσειρές. Περιλαμβάνει ακόμα τις πλειάδες, τις λίστες, τα λεξικά και τα σύνολα που θα δούμε σε αυτό το κεφάλαιο. Για αυτό και όλα τους έχουν βασικές λειτουργίες ή αλλιώς μεθόδους με τις οποίες, για παράδειγμα, τα δημιουργούμε, τα προσπελαύνουμε και, εφόσον το επιτρέπουν, τα τροποποιούμε. Για κάθε δομή θα δούμε σύντομα και τις αντίστοιχες μεθόδους.

7.2 Η δομή της πλειάδας

Μια πλειάδα (tuple) διαφέρει από μια συμβολοσειρά μόνο στο ότι τα στοιχεία της δεν είναι αποκλειστικά χαρακτήρες. Μια ακολουθία στοιχείων, είτε αυτά είναι αριθμοί, είτε συμβολοσειρές, είτε και οποιουδήποτε άλλου τύπου, μπορούν να ομαδοποιηθούν σε μία δομή αυτήν της πλειάδας.

Αν θέλουμε να είμαστε πιο τυπικοί η πλειάδα είναι μια δομή δεδομένων με τα εξής χαρακτηριστικά:

- ✓ είναι σύνθετη, δηλαδή χρησιμοποιείται για την αποθήκευση πολλαπλών δεδομένων ταυτόχρονα
- ✓ είναι διατεταγμένη, δηλαδή η σειρά των μελών είναι σημαντική (ακολουθία)
- ✓ είναι μη μετατρέψιμη, δηλαδή τα στοιχεία της δεν μπορούν να αλλάξουν μόλις εκχωρηθούν σε αυτά τιμές (δεδομένα)
- ✓ τα δεδομένα της μπορούν να είναι ετερογενή, δηλαδή πολλών διαφορετικών τύπων χωρίς περιορισμό

Αυτό το τελευταίο χαρακτηριστικό είναι αυτό που καθιστά την πλειάδα ένα πολύ σημαντικό εργαλείο για την Python.

Λόγω της φύσης της η πλειάδα χρησιμοποιείται όταν οι τιμές παραμένουν σταθερές και δεν μπορούν να αλλάξουν, για την αναπαράσταση σταθερών συνόλων τιμών ή για την ομαδοποίηση στοιχείων που δεν πρέπει να αλλάξουν. Είναι απλούστερη ως προς τη χρήση και πιο αποτελεσματική ως προς τις επιδόσεις της μνήμης από τη λίστα. Προτιμάται ιδιαίτερα όταν θέλουμε να εισάγουμε σταθερές ή μεταβλητές μιας χρήσης.

Ας δούμε μερικά παραδείγματα διαφορετικών τρόπων δημιουργίας πλειάδων (tuple):

```
# Τρόποι δημιουργίας πλειάδων (tuples)
tuple1 = (1, 2, 3, 4, 5, 6)    # Κλασική δημιουργία πλειάδας (tuple)
# που χρησιμοποιεί παρενθέσεις για τη δημιουργία
# της.
tuple2 = tuple([10, 20, 30])    # Εναλλακτικός τρόπος δημιουργίας
# πλειάδας χρησιμοποιώντας τη συνάρτηση tuple().
# Σημειώστε τις αγκύλες μέσα στην παρένθεση, θα τις εξηγήσουμε
# σε επόμενη ενότητα.
tuple3 = 1, 2, 'τρία'    # Όταν η Python βλέπει να ανατίθεται
# μια ακολουθία σε μια μεταβλητή χωρίς παρενθέσεις
# αγκύλες ή άγκιστρα, υποθέτει ότι πρόκειται για
# πλειάδα. Για αυτό και μπορούμε και με αυτό τον τρόπο
# να ορίσουμε μια πλειάδα. Έτσι αν ρωτήσουμε για την
# tuple3, γράφοντας:
 print(type(tuple3)
# θα μας επιστρέψει:
<class 'tuple'>
```

Σημείωση: Όταν δημιουργούμε μια μοναδιαία πλειάδα δηλ. μια πλειάδα με ένα μόνο μέλος, ή αλλιώς στοιχείο, ιδιαίτερα όταν πρόκειται για αριθμό ή μεταβλητή, πρέπει μετά το στοιχείο να τοποθετήσουμε και κόμμα. Αυτό γιατί αλλιώς θα εκληφθεί ως αριθμητική παράσταση.

```
  x = 3
  tuple4 = (3)    # Μια απλή μεταβλητή, όχι tuple. Πράγματι αν
# ζητήσουμε τον τύπο της:
  print(type(tuple4))
<class 'integer'>    # Θα είναι ακέραιος ! Με τιμή 3.
  tuple4 = 3,    # Προσέξτε το κόμμα, το tuple4 είναι πλέον tuple
  print(type(tuple4))
<class 'tuple'>
  tuple1[3] = 14    # Προσπαθούμε να αλλάξουμε την τιμή 4 που
# υπάρχει στη θέση 3 (4η) της λίστας tuple1 σε 14. Δεν
# επιτρέπεται. Θα προκληθεί σφάλμα TypeError !
```

Είδαμε μόλις ότι δεν μπορούμε να αλλάξουμε μια πλειάδα. Το ίδιο ισχύει και για τα στοιχεία της. Έτσι, αν προσπαθούσαμε να αλλάξουμε ένα από τα στοιχεία της tuple1 στο πρώτο παράδειγμα θα ήταν και πάλι ανέφικτο.

Πράγματι η εντολή που ακολουθεί και που προσπαθεί να αλλάξει το 4ο στοιχείο της πλειάδας tuple1, η οποία θυμίζουμε είναι η (1, 2, 3, 4, 5, 6) από 4 σε 5, θα δώσει πάλι σφάλμα TypeError!

```
  tuple1[3] = 5    # Θα προκληθεί και πάλι σφάλμα TypeError!
```

Προσοχή! Έχουμε πει ότι μέλος/ στοιχείο μιας πλειάδας μπορεί να είναι ο,τιδήποτε: ένας ακέραιος, ένα string αλλά ακόμα και μια μεταβλητή ή ένα αντικείμενο. Ας πάρουμε την περίπτωση της μεταβλητής. Η μεταβλητή που αποτελεί το 3ο μέλος της πλειάδας μας δεν μπορεί να αλλάξει. Η τιμή της όμως; Ας δούμε:

```
  x = 3
  y = 5
  tuple5 = (x, y)
  print(x + ' και ' + y)
3 και 5
  x = 7    # Τροποποιούμε την τιμή της μιας μεταβλητής
  print(x + ' και ' + y)
7 και 5
```

Επομένως, εφόσον ως στοιχεία μιας πλειάδας έχουν οριστεί μεταβλητές, οι μεταβλητές αυτές δεν μπορούν να αλλαχθούν, καθώς η πλειάδα είναι μη μετατρέψιμη (immutable). Οι τιμές όμως των μεταβλητών αυτών μπορούν φυσικά να αλλάζουν αφού είναι μεταβλητές (mutable).

7.2.1 Στοιχεία μιας πλειάδας (πρόσβαση, διάσχιση)

Η πρόσβαση στα στοιχεία μιας πλειάδας γίνεται με τον ίδιο ακριβώς τρόπο που γίνεται η πρόσβαση και στα στοιχεία μιας συμβολοσειράς. Η αρίθμηση των θέσεων των στοιχείων ξεκινά από 0 και τελειώνει σε (μέγεθος -1). Λέμε ότι ο δείκτης, ή το ευρετήριο, της πλειάδας ξεκινάει από 0 και φτάνει μέχρι μέγεθος -1. Παρότι τη λίστα την ορίζουμε με παρενθέσεις, η πρόσβαση στα στοιχεία της μέσω δεικτοδότησης, ή αλλιώς ευρετηρίου, γίνεται με αγκύλες. Μερικά παραδείγματα:

```
new_tuple = (1, 2, 3, 4, 5, 6)
print(new_tuple[1])    # Εκτύπωση του 2ου στοιχείου της πλειάδας
2
print(new_tuple[5])    # Εκτύπωση του 6ου στοιχείου της πλειάδας
6
```

Η πρόσβαση μπορεί να γίνει τόσο με θετική όσο και με αρνητική δεικτοδότηση. Η θέση -1 είναι η τελευταία θέση της πλειάδας, η -2 η προτελευταία και ούτω καθ΄ εξής. Μερικά ακόμα παραδείγματα:

```
print(new_tuple[-2])    # Εκτύπωση του προτελευταίου
# στοιχείου της πλειάδας
5
print(new_tuple[-1])    # Εκτύπωση του τελευταίου στοιχείου
# της πλειάδας
6    # Προσέξτε ότι δίνει το ίδιο αποτέλεσμα με την print(new_tuple[5])
```

Μπορούμε επίσης να προσπελάσουμε ένα υποσύνολο μιας πλειάδας αν υποδείξουμε ένα διάστημα για τον δείκτη. Το κομμάτι (slice) που θα προσπελάσουμε ορίζεται από δύο ορίσματα, την αρχική και την τελική θέση του στην πλειάδα. Όταν εκτελείται αυτή η διεργασία, την οποία ονομάζουμε κατάτμηση (στα αγγλικά slicing), επιστρέφει μια νέα πλειάδα που περιλαμβάνει τα επιλεγμένα στοιχεία.

```
tuple_slice = new_tuple[0:3]'
print(tuple_slice)
(1, 2, 3, 4)
```

Για να επισκεφτούμε τα στοιχεία μιας πλειάδας μπορούμε να χρησιμοποιήσουμε ένα βρόγχο σταθερού μεγέθους (δηλαδή ένα βρόγχο for). Κάθε στοιχείο της πλειάδας θα ανατεθεί διαδοχικά στη μεταβλητή που ορίστηκε για αυτό το σκοπό, επιτρέποντας τη χρησιμοποίησή του μέσα στο βρόγχο:

```
for x in new_tuple:
      print(x)
```

7.2.2 Βασικές λειτουργίες πλειάδας, πράξεις και συναρτήσεις που εφαρμόζονται σε αυτή

Οι ακολουθίες στην Python περιλαμβάνουν τις συμβολοσειρές, τις πλειάδες, τις λίστες, και το προϊόν της εφαρμογής της range σε ένα εύρος τιμών. Υπάρχουν μια σειρά από πράξεις και συναρτήσεις που εφαρμόζονται σε όλες τις ακολουθίες (iterables) και επομένως και στις πλειάδες. Επίσης υπάρχουν ενσωματωμένες συναρτήσεις που μετατρέπουν μία ακολουθία σε άλλη διαφορετικού τύπου. Είδαμε ήδη τη δεικτοδότηση και την κατάτμηση που ανήκουν σε αυτή την κατηγορία. Ας δούμε εδώ αναλυτικά μερικές ακόμα δυνατότητες.

Πράξεις για ακολουθίες

Δύο πράξεις εφαρμόζονται σε όλες τις ακολουθίες η συνένωση (concatenation) και ο πολλαπλασιασμός. Η πρώτη εφαρμόζεται ανάμεσα σε δύο ομοειδείς ακολουθίες τις οποίες και συνενώνει σε μία νέα. Η δεύτερη εφαρμόζεται ανάμεσα σε ένα ακέραιο και μία ακολουθία και ουσιαστικά επαναλαμβάνει σε μία νέα ακολουθία, πάντα ίδιου τύπου, τα στοιχεία της ακολουθίας τόσες φορές όσες υποδεικνύει ο ακέραιος. Στις δύο αυτές πράξεις αντιστοιχούν οι τελεστές + και * αντίστοιχα. Τέλος μπορούμε να έχουμε μια ακολουθία από ακολουθίες. Ας δούμε μερικά παραδείγματα:

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
print(tuple1 + tuple2)    # Εκτύπωση της συνένωσης των δύο πλειάδων
(1, 2, 3, 4, 5, 6)
print(2 * tuple1)    # Εκτύπωση του πολλαπλασιασμού της
# πλειάδας επί δύο (θα έδινε το ίδιο αποτέλεσμα και το
# tuple1 * 2)
(1, 2, 3, 1, 2, 3)
tuple3 = (tuple1, tuple2)
print(tuple3)
((1, 2, 3), (4, 5, 6))    # Εκτύπωση μιας πλειάδας πλειάδων
```

Φυσικά μπορεί κάποια στοιχεία να είναι πλειάδες και κάποια άλλα όχι. Αν αναρωτιέστε που μπορεί να είναι χρήσιμο αυτό, ας δούμε άλλο ένα παράδειγμα. Ένα σημείο στο επίπεδο ορίζεται από τις δύο συντεταγμένες του στον άξονα x και τον άξονα y. Μία απόσταση ανάμεσα σε ένα σημείο και μια γραμμή ορίζεται από την απόσταση του σημείου από το πλησιέστερο σημείο της γραμμής. Αν η γραμμή είναι κύκλος και το σημείο το κέντρο του, η απόσταση αυτή είναι ίδια με όποιο σημείο του κύκλου και αν μετρήσουμε και είναι η ακτίνα του κύκλου. Ας ορίσουμε τώρα το κέντρο, την ακτίνα και τον ίδιο τον κύκλο:

```
center = (10, 20)    # Η πλειάδα που ορίζει το κέντρο του
# κύκλου
r = 5
circle = (center, r)    # Η πλειάδα που ορίζει τον κύκλο
print(circle)
```

```
((10, 20), 5 )    # Η πλειάδα που ορίζει τον κύκλο αναλυτικά
```

Έλεγχος συμπερίληψης

Μπορούμε επίσης να ελέγξουμε αν ένα στοιχείο ανήκει σε μία πλειάδα ή και το αντίθετο. Αυτό γίνεται με την εφαρμογή των τελεστών `in` και `not in` για έλεγχο συμπερίληψης ή μη αντίστοιχα. Και οι δύο τελεστές εφαρμόζονται ανάμεσα σε ένα πιθανό στοιχείο και μία πλειάδα, δημιουργώντας μία παράσταση που μπορεί να είναι είτε αληθής είτε ψευδής. Όπως όλες οι παραστάσεις μπορεί να χρησιμοποιηθεί ως συνθήκη, για παράδειγμα σε μία εντολή επιλογής ή επανάληψης. Θυμηθείτε το έχουμε ήδη χρησιμοποιήσει στην προηγούμενη παράγραφο σε ένα `for`. Ας δούμε άλλο ένα παράδειγμα:

```
for x in tuple1
     print(x)  # Θα τυπώσει διαδοχικά τους αριθμούς 1,2 και 3
```

Συγκρίσεις

Και οι συγκριτικοί τελεστές μπορούν να εφαρμοστούν στις ακολουθίες. Αρκεί οι δύο ακολουθίες να είναι συγκρίσιμες. Εφόσον αυτό συμβαίνει, συγκρίνονται τα στοιχεία στην ίδια θέση (ξεκινώντας από τη θέση μηδέν) μέχρι να βρεθεί διαφορετικό και τότε η ακολουθία που έχει το στοιχείο με τη μεγαλύτερη τιμή θα αντιμετωπιστεί ως μεγαλύτερη. Περισσότερα για αυτό θα πούμε σε άλλο σημείο.

Συναρτήσεις

Υπάρχουν όπως έχουμε ήδη αναφέρει μια σειρά από συναρτήσεις που εφαρμόζονται σε όλες τις ακολουθίες, άρα και στις πλειάδες. Αυτές περιλαμβάνουν την εύρεση του μήκους της ακολουθίας δηλαδή τη συνάρτηση `len( )` και, εφόσον οι τιμές είναι συγκρίσιμες, το μέγιστο και ελάχιστο στοιχείο με τις συναρτήσεις `max( )`και `min( )` αντίστοιχα. Ακόμα η, επίσης ενσωματωμένη, συνάρτηση `sorted( )` επιστρέφει μια ταξινομημένη λίστα με τα ίδια στοιχεία με την πλειάδα μας. Ας δούμε κι εδώ κάποια παραδείγματα:

```
tuple1 = (2, 1, 3, 5)
print(len(tuple1 ))    # Θα τυπώσει το μέγεθος της πλειάδας
4
print(max(tuple1 ))    # Θα τυπώσει τη μεγαλύτερη τιμή

5
print(min(tuple1 ))    # Θα τυπώσει τη μικρότερη τιμή

1
print(sorted(tuple1 ))    # Θα τυπώσει μια λίστα με τα ίδια
# στοιχεία ταξινομημένα.
[1, 2, 3, 5]
```

Μέθοδοι

Όλες οι ακολουθίες, όπως όλοι οι τύποι δεδομένων στην Python είναι αντικείμενα. Επομένως έχουν μεθόδους. Τις λιγότερες μεθόδους από οποιονδήποτε τύπο ακολουθίες τις έχει η πλειάδα και μάλιστα είναι κοινές με όλους τους άλλους τύπους. Για να δούμε τις μεθόδους ενός αντικειμένου χρησιμοποιούμε την ενσωματωμένη συνάρτηση `dir( )`. Η συνάρτηση επιστρέφει μια λίστα με τα ονόματα των μεθόδων του αντικειμένου μας. Αν την εφαρμόσουμε σε μία πλειάδα θα μας επιστρέψει μόλις δύο μεθόδους τις `count( )` και `index( )`. Και οι δύο μέθοδοι εφαρμόζονται σε ένα πιθανό στοιχείο της πλειάδας και επιστρέφουν η μεν πρώτη τον αριθμό των εμφανίσεών του στην πλειάδα, η δε δεύτερη τη θέση της πρώτης εμφάνισης αποδιδόμενη με τον αντίστοιχο δείκτη.

```
tuple1 = (1, 2, 3, 3)
print(dir(tuple1 ))    # Θα τυπώσει τις μεθόδους της πλειάδας
['count', 'index']
print(tuple1.count(3))    # Θα τυπώσει τη μεγαλύτερη τιμή
2
print(tuple1.index(3))    # Θα τυπώσει τη θέση πρώτης εμφάνισης

2
```

Άλλες χρήσεις των πλειάδων

Οι πλειάδες χρησιμοποιούνται για πολλαπλή εκχώρηση τιμών – με ή χωρίς τις παρενθέσεις. Φυσικά, το πλήθος των μεταβλητών και τιμών στις δύο πλευρές της εκχώρησης πρέπει να είναι το ίδιο:

```
x, y, z = (1, 2, 3)    # Θα ήταν το ίδιο με παρενθέσεις μπροστά # ή χωρίς
πίσω.
print(x)    # Θα τυπώσει την τιμή του x
1
print(y)    # Θα τυπώσει την τιμή του y
2
print(z)    # Θα τυπώσει την τιμή του z

3
```

Χρησιμοποιούνται ακόμα για να επιστρέψει μια συνάρτηση πολλαπλές τιμές χρησιμοποιώντας τη `return`. Δε χρειάζεται καμία τροποποίηση, επιστρέφει απλά μια πλειάδα.

```
def limits(a)
      x = min(a)
      y = max(a)
      return x, y
tuple1 = (1, 2, 3, 10, 7)
print(limits(tuple1))    # Θα τυπώσει τις min και max τιμές της πλειάδας
(1, 10)
```

Χρησιμοποιούνται επίσης στην ανταλλαγή τιμών, ένα ιδιαίτερο χαρακτηριστικό της Python.

```
X = 'Νίκος'
Y = 'Μαρία'
Y, X = X, Y    # Εκχώρηση τιμών μέσω ανταλλαγής
print(X)    # Θα τυπώσει την τιμή του X
'Μαρία'    # Αντίστοιχα η τιμή του Y θα είναι 'Νίκος'
```

Οι τρεις αυτές χρήσεις δεν είναι πολύ διαφορετικές όπως μπορεί κάποιος να δεί από την ακόλουθη υλοποίηση.

```
def swap(x,y):
      return y, x     # Συνάρτηση ανταλλαγής

x = 1
y = 2
x,y = swap(x,y)    # Θα ανταλλάξει τις δύο τιμές
print(x, ' και ', y)    # Θα τυπώσει τις νέες τιμές
 2 και 1
```

Μετατροπή ακολουθίας σε διαφορετικού τύπου.

Κάθε είδους ακολουθία στην Python μπορεί να μετατραπεί εύκολα σε μια άλλου τύπου απλά χρησιμοποιώντας μια συνάρτηση με το όνομα του τύπου. Αυτό που παίρνουμε είναι μια νέα ακολουθία του τύπου που προσδιορίζει η συνάρτηση που χρησιμοποιήσαμε με στοιχεία τα στοιχεία της ακολουθίας που χρησιμοποιήσαμε σαν βάση.

```
list1 = [1, 2, 4]    # Ορίζει μια λίστα με τρία στοιχεία
tuple1 = tuple(list1)    # Δημιουργεί μια πλειάδα από τη λίστα
print(tuple1)    # Θα τυπώσει την πλειάδα
(1, 2, 4)
```

Τέλος οι πλειάδες χρησιμοποιούνται και στα λεξικά αλλά αυτό θα το δούμε στην αντίστοιχη ενότητα.

7.2.3 Να θυμάμαι

Πλειάδες

Η πλειάδα είναι μια δομή δεδομένων με τα εξής χαρακτηριστικά:

- ✓ είναι σύνθετη, δηλαδή χρησιμοποιείται για την αποθήκευση πολλαπλών δεδομένων ταυτόχρονα
- ✓ είναι διατεταγμένη, δηλαδή η σειρά των μελών είναι σημαντική (ακολουθία)
- ✓ είναι μη μετατρέψιμη (immutable), δηλαδή τα στοιχεία της δεν μπορούν να αλλάξουν μόλις εκχωρηθούν σε αυτά τιμές (δεδομένα)
- ✓ τα δεδομένα της μπορούν να είναι ετερογενή, δηλαδή πολλών διαφορετικών τύπων χωρίς περιορισμό

Πράξεις για ακολουθίες

Δύο πράξεις εφαρμόζονται σε όλες τις ακολουθίες η συνένωση (concatenation) και ο πολλαπλασιασμός:

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
print(tuple1 + tuple2)
(1, 2, 3, 4, 5, 6)
print(2 * tuple1)
(1, 2, 3, 1, 2, 3)
```

Έλεγχος συμπερίληψης

Μπορούμε επίσης να ελέγξουμε αν ένα στοιχείο ανήκει σε μία πλειάδα ή και το αντίθετο. Αυτό γίνεται με την εφαρμογή των τελεστών `in` και `not in` για έλεγχο συμπερίληψης ή μη.

Συγκρίσεις

Και οι συγκριτικοί τελεστές (`==`, `!=`, `>=`, `>`, `<=`, `<`)μπορούν να εφαρμοστούν στις πλειάδες όπως σε όλες τις ακολουθίες. Αρκεί οι δύο πλειάδες να είναι συγκρίσιμες.

Συναρτήσεις

Μια σειρά από συναρτήσεις εφαρμόζονται σε όλες τις ακολουθίες, άρα και στις πλειάδες. Αυτές περιλαμβάνουν την εύρεση του μήκους της ακολουθίας με τη συνάρτηση `len( )` και, εφόσον οι τιμές είναι συγκρίσιμες, το μέγιστο και ελάχιστο με τις `max( )`και `min( )` αντίστοιχα.

Μέθοδοι

Μία πλειάδα, ως δομή, έχει δύο μεθόδους, τις `count( )` και `index( )`. Και οι δύο μέθοδοι εφαρμόζονται σε ένα πιθανό στοιχείο της πλειάδας και επιστρέφουν η μεν πρώτη τον αριθμό των εμφανίσεών του στην πλειάδα, η δε δεύτερη τη θέση της πρώτης εμφάνισης αποδιδόμενη με τον αντίστοιχο δείκτη.

7.2.4 Ερωτήσεις Κατανόησης

1. Η πλειάδα στην Python είναι μια απλή δομή δεδομένων.

   ΝΑΙ ΟΧΙ

2. Η πλειάδα στην Python έχει μόνο δύο μεθόδους, τις `count( )` και `index( ).`

   ΝΑΙ ΟΧΙ

3. Η πλειάδα χρησιμοποιείται κυρίως για την αποθήκευση μεταβλητών τιμών.

   ΝΑΙ ΟΧΙ

4. Σε πλειάδες αποθηκεύουμε ομοειδή δεδομένα.

   ΝΑΙ ΟΧΙ

5. Η γλώσσα Python διαθέτει συναρτήσεις που εφαρμόζονται στις ακολουθίες, όπως οι `len( )`, `max( )`και `min( )`

   ΝΑΙ ΟΧΙ

6. Οι συγκρίσεις δεν εφαρμόζονται στις πλειάδες.

   ΝΑΙ ΟΧΙ

7.2.5 Ερωτήσεις Ανάπτυξης

1. Ποια είναι τα χαρακτηριστικά της πλειάδας στην Python;
2. Ποιες ιδιαίτερες λειτουργίες της Python στηρίζονται στις πλειάδες;
3. Περιγράψτε τον έλεγχο συμπερίληψης στην Python.

7.2.6 Ασκήσεις

7.3 Η δομή της λίστας

Μια λίστα (list) διαφέρει από μια πλειάδα μόνο στο ότι είναι μετατρέψιμη (mutable), δηλαδή τα στοιχεία της μπορούν να αλλάξουν όσες φορές θέλουμε. Όταν αλλάζει κάποιο από τα στοιχεία της η λίστα παραμένει η ίδια. Σχετικά με μία συμβολοσειρά μια λίστα έχει δύο διαφορές, το ότι είναι μετατρέψιμη και το ότι τα στοιχεία της δεν είναι αποκλειστικά χαρακτήρες. Μια ακολουθία στοιχείων, είτε αυτά είναι αριθμοί, είτε συμβολοσειρές, είτε και οποιουδήποτε άλλου τύπου, μπορούν να ομαδοποιηθούν στη δομή της λίστας και να τροποποιηθούν χωρίς περιορισμούς μετά.

Αν θέλουμε να είμαστε πιο τυπικοί η λίστα είναι μια δομή δεδομένων με τα εξής χαρακτηριστικά:

- ✓ είναι σύνθετη, δηλαδή χρησιμοποιείται για την αποθήκευση πολλαπλών δεδομένων ταυτόχρονα
- ✓ είναι διατεταγμένη, δηλαδή η σειρά των μελών είναι σημαντική (ακολουθία)
- ✓ είναι μετατρέψιμη, δηλαδή τα στοιχεία της μπορούν να αλλάξουν και αφού εκχωρηθούν σε αυτά τιμές (δεδομένα)
- ✓ τα δεδομένα της μπορούν να είναι ετερογενή, δηλαδή πολλών διαφορετικών τύπων χωρίς περιορισμό

Η μεταβλητότητα είναι ένα από τα χαρακτηριστικά που καθιστούν την λίστα το πλέον, ίσως, δυναμικό εργαλείο για την Python.

Λόγω της φύσης της η λίστα χρησιμοποιείται για να ομαδοποιήσει μια ακολουθία μεταβλητών που μπορούν να αλλάξουν ανά πάσα στιγμή. Κατ΄ αυτόν τον τρόπο μπορεί να χρησιμοποιηθεί για την αποτύπωση και ομαδοποίηση των τιμών κάποιων μεταβλητών του περιβάλλοντος και άλλων ασταθών τιμών. Έχει λιγότερους περιορισμούς από την πλειάδα, πολλές περισσότερες μεθόδους αλλά και μεγαλύτερες απαιτήσεις διαχείρισης μνήμης. Προτιμάται ιδιαίτερα όταν θέλουμε να εισάγουμε αρχικές τιμές που θα τροποποιηθούν αργότερα.

Ας δούμε τους διαφορετικούς τρόπους δημιουργίας λιστών:

```
# Τρόποι δημιουργίας λιστών (lists)
list1 = [1, 2, 3, 4, 5, 6]    # Κλασική δημιουργία λίστας (list)
# που χρησιμοποιεί αγκύλες για τη δημιουργία της.
list2 = list((10, 20, 30))    # Εναλλακτικός τρόπος δημιουργίας
# λίστας χρησιμοποιώντας τη συνάρτηση list() σε κάποια
# ακολουθία.
list3 = [ ]    # Δημιουργία κενής λίστας. Μπορεί να γεμίσει
# αργότερα.
# Προσέξτε τις αγκύλες παντού εκτός από τη συνάρτηση
# μετατροπής. Δεν μπορούμε να τις παραλείψουμε γιατί τότε η Python θα
# θεωρήσει ότι πρόκειται για πλειάδα.

# Γράφοντας για τη list2:
 print(type(list2))
```

```
# θα μας επιστρέψει:
<class 'list'>
```

```
 list1[3] = 14    # Αλλάζουμε την τιμή 4 που είναι στη θέση 3
# (4η) της λίστας list1 σε 14. Και αυτό δεν προκαλεί κανένα
# σφάλμα
```

Είδαμε μόλις ότι μπορούμε να αλλάξουμε μια τιμή στη λίστα. Το ίδιο ισχύει και για ολόκληρη τη λίστα φυσικά.

Έχουμε πεί ότι μέλος/ στοιχείο μιας λίστας, μπορεί να είναι ο,τιδήποτε: ένας ακέραιος, ένα string αλλά ακόμα και μια μεταβλητή ή ένα αντικείμενο. Στην περίπτωση της μεταβλητής, μπορούμε όχι μόνο να αλλάξουμε την τιμή της μεταβλητής όπως σε μία πλειάδα αλλά και την ίδια τη μεταβλητή. Ας δούμε:

```
  x = 3
  y = 5
  list4 = [x, y]
  print(x + ' και ' + y)
3 και 5
  x = 7    # Τροποποιούμε την τιμή της μιας μεταβλητής
  print(x + ' και ' + y)
7 και 5
  z = 9
  list4 = (y, z)    # Τροποποιούμε την λίστα
  print(list4)
[5, 9]    # και όλα καλά.
```

Εφόσον και η λίστα και τα στοιχεία της μπορούν να τροποποιηθούν, αν ως στοιχεία της έχουν οριστεί μεταβλητές, τόσο οι μεταβλητές αυτές όσο και οι τιμές τους, μπορούν να αλλαχθούν, καθώς η λίστα είναι μετατρέψιμη (mutable).

7.3.1 Στοιχεία μιας λίστας (πρόσβαση, διάσχιση)

Η πρόσβαση στα στοιχεία μιας λίστας γίνεται με τον ίδιο τρόπο που γίνεται η πρόσβαση και στα στοιχεία μιας συμβολοσειράς ή μιας πλειάδας. Η αρίθμηση των θέσεων των στοιχείων ξεκινά από 0 και τελειώνει σε μέγεθος -1. Η λίστα, την οποία εξ' αρχής ορίζουμε με αγκύλες, τις διατηρεί και στη δεικτοδότηση. Μερικά παραδείγματα:

```
new_list = [1, 2, 3, 4, 5, 6]
print(new_tuple[1])    # Εκτύπωση του 2ου στοιχείου της λίστας
2
print(new_tuple[5])    # Εκτύπωση του 6ου στοιχείου της λίστας
6
```

Η πρόσβαση μπορεί να γίνει τόσο με θετική όσο και με αρνητική δεικτοδότηση. Η θέση -1 είναι η τελευταία θέση της λίστας, η -2 η προτελευταία και ούτω καθ΄ εξής. Μερικά ακόμα παραδείγματα:

```
print(new_list[-2])    # Εκτύπωση του προτελευταίου
# στοιχείου της πλειάδας
5
print(new_list [-1])    # Εκτύπωση του τελευταίου στοιχείου
# της πλειάδας
6    # Προσέξτε ότι δίνει το ίδιο αποτέλεσμα με την
# εντολή print(new_tuple[5])
```

Μπορούμε επίσης να προσπελάσουμε ένα υποσύνολο μιας λίστας, όπως ακριβώς και με την πλειάδα, αν υποδείξουμε ένα διάστημα για τον δείκτη. Το κομμάτι (slice) που θα προσπελάσουμε ορίζεται από δύο ορίσματα, την αρχική και την τελική θέση του στην πλειάδα. Όταν εκτελείται αυτή η διεργασία (slicing) σε μια λίστα, επιστρέφει μια νέα λίστα που περιλαμβάνει τα επιλεγμένα στοιχεία.

```
list_slice = new_list[0:2]'
print(list_slice)
(1, 2, 3)
```

Όπως ακριβώς στην περίπτωση της πλειάδας, για να επισκεφτούμε τα στοιχεία μιας λίστας μπορούμε να χρησιμοποιήσουμε ένα βρόγχο for. Κάθε στοιχείο της λίστας θα ανατεθεί διαδοχικά στη μεταβλητή που ορίστηκε για αυτό το σκοπό, επιτρέποντας τη χρησιμοποίησή του μέσα στο βρόγχο:

```
for x in new_list :
     print(x)  # Θα τυπώσει διαδοχικά τους αριθμούς 1,2 και 3
```

7.3.2 Βασικές λειτουργίες λίστας, πράξεις και συναρτήσεις που εφαρμόζονται σε αυτή

Όλες οι πράξεις, συναρτήσεις και τελεστές που εφαρμόζονται στις ακολουθίες εφαρμόζονται φυσικά και στις λίστες. Ας δούμε πάλι μερικές δυνατότητες.

Πράξεις για ακολουθίες

Έχουμε ήδη αναφέρει ότι δύο πράξεις εφαρμόζονται σε όλες τις ακολουθίες, άρα και στις λίστες: η συνένωση (concatenation) και ο πολλαπλασιασμός. Χρησιμοποιούνται για αυτές οι τελεστές `+` και `*` αντίστοιχα. Τέλος μπορούμε να έχουμε μια λίστα από λίστες. Ας δούμε πάλι μερικά παραδείγματα:

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
print(list1 + list2)    # Εκτύπωση της συνένωσης των δύο λιστών
[1, 2, 3, 4, 5, 6]
print(2 * list1)    # Εκτύπωση του πολλαπλασιασμού της
# λίστας επί δύο (θα έδινε το ίδιο αποτέλεσμα και το
# list1 * 2)
[1, 2, 3, 1, 2, 3]
list3 = (list1, list2)
print(list3)
[[1, 2, 3], [4, 5, 6]]    # Εκτύπωση μιας λίστας λιστών
```

Έλεγχος συμπερίληψης

Μπορούμε επίσης, όπως και στην πλειάδα, να ελέγξουμε αν ένα στοιχείο ανήκει σε μία λίστα ή και το αντίθετο. Αυτό γίνεται με την εφαρμογή των τελεστών `in` και `not in` για έλεγχο συμπερίληψης ή μη αντίστοιχα. Και οι δύο τελεστές εφαρμόζονται ανάμεσα σε ένα πιθανό στοιχείο και μία πλειάδα, δημιουργώντας μία παράσταση που μπορεί να είναι είτε αληθής είτε ψευδής. Όπως όλες οι παραστάσεις μπορεί να χρησιμοποιηθεί ως συνθήκη, για παράδειγμα σε μία εντολή επιλογής ή επανάληψης. Θυμηθείτε το έχουμε ήδη χρησιμοποιήσει τόσο στην προηγούμενη παράγραφο όσο και στις πλειάδες. Ας δούμε άλλο ένα παράδειγμα:

```
newlist = [x for x in list2 if x <= 5]    # Διατρέχουμε τη λίστα με
# έλεγχο συμπερίληψης και περιλαμβάνουμε στη νέα τα
# στοιχεία που ικανοποιούν την επιπλέον συνθήκη.
print(newlist)    # Εκτύπωση της νέας λίστας που προέκυψε από
# εφαρμογή συνθήκης στα στοιχεία της παλαιής λίστας
[4, 5]
```

Συγκρίσεις

Και στις λίστες μπορούν να εφαρμοστούν οι συγκριτικοί τελεστές. Εφόσον οι δύο λίστες είναι συγκρίσιμες, συγκρίνονται τα στοιχεία στην ίδια θέση – ξεκινώντας φυσικά από τη θέση μηδέν – μέχρι να βρεθεί διαφορετικό και τότε η ακολουθία που έχει το στοιχείο με τη μεγαλύτερη τιμή θα αντιμετωπιστεί ως μεγαλύτερη. Τα επόμενα στοιχεία, ο αριθμός τους ή οι τιμές τους δεν παίζουν ρόλο. Περίπου έτσι κατατάσσονται αλφαβητικά οι εγγραφές με βάση το επώνυμο (που είναι μία συμβολοσειρά) σε ένα τηλεφωνικό κατάλογο (που συνήθως είναι ένα λεξικό μπορεί όμως να είναι και μια λίστα αντικειμένων). Περισσότερα για αυτό αργότερα.

Συναρτήσεις

Υπάρχουν όπως έχουμε ήδη αναφέρει μια σειρά από συναρτήσεις που εφαρμόζονται και στις λίστες. Αυτές περιλαμβάνουν την εύρεση του μήκους της λίστας δηλαδή τη συνάρτηση `len( )` και, εφόσον οι τιμές είναι συγκρίσιμες μεταξύ τους, το μέγιστο και ελάχιστο στοιχείο με τις συναρτήσεις `max( )`και `min( )` αντίστοιχα. Ακόμα η, επίσης ενσωματωμένη, συνάρτηση `sorted( )` επιστρέφει μια ταξινομημένη λίστα με τα ίδια στοιχεία. Ας δούμε πώς λειτουργούν:

```
list1 = [0, 2, 1, 3, 4]
print(len(list1))    # Θα τυπώσει το μέγεθος της λίστας
5
print(max(list1))    # Θα τυπώσει τη μεγαλύτερη τιμή

4
print(min(list1))    # Θα τυπώσει τη μικρότερη τιμή

0
print(sorted(list1))    # Θα τυπώσει τη λίστα μας αλλά

# ταξινομημένη με αύξουσα σειρά. Προσοχή δεν

# αλλάζει τη λίστα απλά επιστρέφει μια άλλη,

# ταξινομημένη εκδοχή της

[0, 1, 2, 3, 4]
```

Μέθοδοι

Όλες οι ακολουθίες στην Python, άρα και οι λίστες, είναι αντικείμενα και έχουν μεθόδους. Τις λιγότερες μεθόδους από οποιονδήποτε τύπο ακολουθίας τις έχει όπως είδαμε η πλειάδα. Αντίθετα η λίστα έχει μια ποικιλία μεθόδων. Αν εφαρμόσουμε την ενσωματωμένη συνάρτηση `dir( )` σε μια λίστα, η συνάρτηση επιστρέφει μια μακροσκελή λίστα από μεθόδους. Σε αυτή περιλαμβάνονται πολλές με διπλή κάτω παύλα που είναι ιδιωτικές και δεν θα μας απασχολήσουν εδώ. Ακόμα κι έτσι μένουν έντεκα (11) δημόσιες μέθοδοι έναντι μόλις δύο (2) της πλειάδας. Αυτό είναι ένα ακόμα χαρακτηριστικό που καθιστά τη λίστα μια από τις πιο δυναμικές δομές δεδομένων στην Python. Ας δούμε ποιες είναι αυτές οι μέθοδοι και τι κάνουν:

`.append( )` Δέχεται ως όρισμα ένα στοιχείο που θέλουμε να προσθέσουμε στη λίστα και το προσθέτει στο τέλος της, αυξάνοντας έτσι το συνολικό μήκος της κατά 1.

`.clear()` Χωρίς όρισμα, διαγράφει όλα τα στοιχεία από τη λίστα.

`.copy()` Χωρίς όρισμα, επιστρέφει ένα αντίγραφο της λίστας (με αντιγραφή by value).

`.count()` Δέχεται ως όρισμα μία τιμή και επιστρέφει τον αριθμό των στοιχείων της λίστας με αυτή την τιμή.

`.extend()` Δέχεται ως όρισμα μία λίστα, ή οποιαδήποτε άλλη ακολουθία, και προσθέτει τα στοιχεία της στο τέλος της υπάρχουσας λίστας.

`.index()` Δέχεται ως όρισμα μία τιμή και επιστρέφει τη θέση στη λίστα που αυτή εμφανίζεται για πρώτη φορά.

`.insert( )` Δέχεται ως ορίσματα μία θέση στη λίστα και μια τιμή και εισάγει στη συγκεκριμένη θέση ένα στοιχείο με αυτή την τιμή. Το στοιχείο που τυχόν υπήρχε στη συγκεκριμένη θέση τοποθετείται στην επόμενη και όλα τα επόμενα σε μία επόμενη από την τρέχουσα θέση.

`.pop( )` Δέχεται ως όρισμα μία θέση στη λίστα και απομακρύνει από τη λίστα το στοιχείο που βρίσκεται στη συγκεκριμένη θέση. Οι θέσεις των επόμενων στοιχείων στη λίστα τροποποιούνται κατάλληλα ώστε στη νέα λίστα να υπάρχει συνέχεια στην αρίθμηση των θέσεων (δεικτοδότηση). Χωρίς όρισμα απομακρύνει το τελευταίο στοιχείο στη λίστα.

`.remove()` Χωρίς όρισμα, διαγράφει το πρώτο στοιχείο της λίστας και τροποποιεί κατάλληλα τη δεικτοδότηση. Μπορεί να δεχτεί ως όρισμα τιμή, την οποία θα αναζητήσει στη λίστα και θα διαγράψει το στοιχείο στο οποίο θα τη συναντήσει για πρώτη φορά.

`.reverse()` Χωρίς όρισμα, αντιστρέφει τη σειρά των στοιχείων της λίστας. Επειδή η λίστα είναι μετατρέψιμη, το αποτέλεσμα της ενέργειας είναι μόνιμο.

`.sort()` Χωρίς όρισμα, ταξινομεί τα στοιχεία της λίστας. Αντίθετα από τη συνάρτηση sorted( ), η μέθοδος .sort( ) έχει μόνιμο αποτέλεσμα στη λίστα, η οποία είναι πλέον ταξινομημένη με αύξουσα σειρά. Μπορούμε επίσης να δώσουμε ως όρισμα reverse = True και τότε θα πάρουμε τη λίστα με φθίνουσα ταξινόμηση.

Ας δούμε μερικά παραδείγματα:

```
list = [2,3,5]    # Ορίζουμε μια λίστα
list.append(6)
print(list)
[2, 3, 5, 6]
list.clear()
print(list)
 []
a = [3,4,5]
list.extend(a)
print(list)
[3, 4, 5]
list.pop()
```

```
a = list.copy()
a.insert(7,2)
a.pop(1)
print(a)
[3, 2]
list.insert(1,9)
list.reverse()
print(list)
[4, 9, 3]
list.sort(reverse = True)
[9, 4, 3]
```

Ταξινόμηση λίστας

Εφόσον η λίστα είναι μια μετατρέψιμη δομή η ταξινόμησή της είναι εφικτή, όπως είδαμε χρησιμοποιώντας τη μέθοδο .sort( ). Παρά το γεγονός ότι υπάρχει αυτή η μέθοδος μπορεί σε κάποιες περιπτώσεις να θελήσουμε να χρησιμοποιήσουμε μια συγκεκριμένη μέθοδο ταξινόμησης αλλά και εναλλακτικούς αλγόριθμους για την κάθε μία, φτιάχνοντας αντίστοιχες συναρτήσεις. Μια ενδιαφέρουσα τέτοια περίπτωση είναι η ταξινόμηση εισαγωγής.

Η συνάρτηση insertionSort που ακολουθεί δέχεται ως είσοδο μία αταξινόμητη λίστα ulist. Αρχικά υπολογίζει το μήκος της λίστας (length). Εφόσον η λίστα έχει περισσότερα από ένα στοιχεία, η συνάρτηση διασχίζει τον πίνακα ξεκινώντας από το δεύτερο στοιχείο. Λαμβάνει το τρέχον στοιχείο (current) και το συγκρίνει με τα στοιχεία στο ταξινομημένο τμήμα του πίνακα που προηγείται. Εάν το τρέχον είναι μικρότερο από ένα στοιχείο στο ταξινομημένο τμήμα, η συνάρτηση τοποθετεί το τρέχον πριν από το στοιχείο αυτό. Αυτή η διαδικασία συνεχίζεται μέχρι να βρεθεί η σωστή θέση για το τρέχον στοιχείο και να εισαχθεί σε αυτή.

```
def insertionSort(ulist):
    length = len(ulist)    # Βάλε το μήκος της λίστας σε μια μεταβλητή
    if length <= 1:
        return     # Στις τετριμμένες περιπτώσεις (0 ή 1 στοιχεία) δε
        # χρειάζεται ταξινόμηση

        for i in range(1, length):    # Διέσχισε τη λίστα ξεκινώντας από
            # το 2ο στοιχείο
        current = ulist[i]    # Αποθήκευσε το τρέχον προς ταξινόμηση
        # στοιχείο στη μεταβλητή current
        j = i-1
        while j >= 0 and current < ulist[j]:    # Όσο συναντάς
        # μεγαλύτερα στοιχεία πήγαινέ τα μια θέση δεξιά
            ulist[j+1] = ulist[j]    # Μεταφορά στοιχείων δεξιά
            j -= 1
```

```
            ulist[j+1] = current  # Βάλε το τρέχον στη σωστή θέση
```

Στην πραγματικότητα, μια που δουλεύουμε με λίστες, δε χρειάζεται να πάμε τα στοιχεία μία θέση δεξιά. Αρκεί να βάλουμε το τρέχον (current) στη σωστή θέση με την insert, που θα πάει όλα τα επόμενα στοιχεία μια θέση δεξιά, και να το διαγράψουμε από την αρχική του θέση με τη pop. Το for μας γίνεται τώρα:

```
for i in range(1, length):  # Διέσχισε τη λίστα ξεκινώντας από
     # το 2ο στοιχείο
current = ulist[i]  # Αποθήκευσε το τρέχον προς ταξινόμηση
# στοιχείο στη μεταβλητή current
j = i-1
while j >= 0 and current < ulist[j]:  # Όσο συναντάς
# μεγαλύτερα στοιχεία πηγαινέ τα μια θέση δεξιά
j -= 1 # Μετακίνηση του δείκτη στη επόμενη θέση
ulist.insert(j+1, current) # Βάλε το τρέχον στη θέση του
ulist.pop(i+1) # Διέγραψε το από την παλιά. Βάζουμε +1
# γιατί προστέθηκε στοιχείο μπροστά
```

Αναζήτηση στοιχείου σε ταξινομημένη λίστα

Και για την αναζήτηση είδαμε ότι υπάρχει η μέθοδος .index( ) που εντοπίζει τουλάχιστον την πρώτη εμφάνιση ενός στοιχείου στη λίστα. Μας επιστρέφει μάλιστα τη θέση του. Όμως υπάρχουν αρκετοί τρόποι αναζήτησης, διαφορετικών δυνατοτήτων και κυρίως ταχυτήτων. Μπορούμε να χρησιμοποιήσουμε εκτός από την .index( ), μία επανάληψη for ή ακόμα μια συνάρτηση. Πιθανόν να θέλουμε να χρησιμοποιήσουμε μια συγκεκριμένη μέθοδο αναζήτησης. Ας υποθέσουμε ότι μας ενδιαφέρει η δυαδική αναζήτηση:

```
def binary_search(list,item):
      start = 0
      end = len(list) - 1

      while start <= end :
            mid = (start + end) // 2
            current = list[mid]

            if current == item :
                  return mid
            elif current > item :
                  end = mid – 1
            else :
                  start = mid + 1
      return -1
```

```
list = [7, 8, 9, 10, 11 ]
answer = binary_search(list, 10)
if answer == -1:
      print('Item not found')
else :
      print('Item found at position', answer)
Item found at position 3
```

Μετατροπή ακολουθίας σε διαφορετικού τύπου

Όπως είδαμε στην προηγούμενη ενότητα οι ακολουθίες μπορούν να μετατραπούν σε άλλες διαφορετικού τύπου. Στην περίπτωση της προηγούμενης ενότητας μετατρέψαμε μια λίστα σε πλειάδα. Το ίδιο εύκολο είναι και το αντίστροφο:

```
tuple1 = (1, 2, 4)     # Ορίζει μια πλειάδα με τρία στοιχεία
list1= list(tuple1)     # Δημιουργεί μια λίστα από την πλειάδα
print(list1)     # Θα τυπώσει την λίστα
[1, 2, 4]
```

Αυτή η μετατροπή είναι πιο συνηθισμένη καθώς οι λίστες επιτρέπουν περισσότερους χειρισμούς των στοιχείων τους.

7.3.3 Να θυμάμαι

Λίστες

Η λίστα είναι μια δομή δεδομένων με τα εξής χαρακτηριστικά:

- ✓ είναι σύνθετη, δηλαδή χρησιμοποιείται για την αποθήκευση πολλαπλών δεδομένων ταυτόχρονα
- ✓ είναι διατεταγμένη, δηλαδή η σειρά των μελών είναι σημαντική (ακολουθία)
- ✓ είναι μετατρέψιμη (mutable), δηλαδή τα στοιχεία της μπορούν να αλλάξουν και αφού εκχωρηθούν σε αυτά τιμές
- ✓ τα δεδομένα της μπορούν να είναι ετερογενή, δηλαδή πολλών διαφορετικών τύπων χωρίς περιορισμό

Πράξεις, έλεγχος συμπερίληψης και συγκρίσεις

Όπως σε κάθε ακολουθία έτσι και στις λίστες εφαρμόζονται η συνένωση (concatenation, με τελεστή +) και ο πολλαπλασιασμός (με θετικό ακέραιο, τελεστής *). Εφαρμόζονται επίσης οι τελεστές in και not in για τον έλεγχο συμπερίληψης ή μη.

Τέλος, και με την προϋπόθεση ότι οι λίστες είναι συγκρίσιμες εφαρμόζονται και οι συγκριτικοί τελεστές όπως της ισότητας (==), διαφορετικότητας (!=), μεγαλύτερου ίσου (>=), γνήσια μεγαλύτερου (>),, μικρότερου ίσου (<=), γνήσια μικρότερου (<) κλπ. Εφόσον οι δύο λίστες είναι συγκρίσιμες, συγκρίνονται τα στοιχεία στην ίδια θέση. Αν βρεθεί διαφορετικό στοιχείο και τότε η ακολουθία που έχει το στοιχείο με τη μεγαλύτερη τιμή θα αντιμετωπιστεί ως μεγαλύτερη. Αν εξαντληθούν τα στοιχεία της μίας τότε η άλλη θεωρείται μεγαλύτερη.

```
list1 = [1, 2, 3, 4]
list2 = [1, 2, 3]
print(list1 < list2)
```

```
False
```

```
print(list1 != list2)
```

```
True
```

```
print(list1 > list2)
```

```
True
```

Μέθοδοι λίστας

Η λίστα έχει 11 δημόσιες μεθόδους, τις `.clear()` Χωρίς όρισμα, διαγράφει όλα τα στοιχεία από τη λίστα.

`.copy()` Χωρίς όρισμα, επιστρέφει ένα αντίγραφο της λίστας (με αντιγραφή by value).

`.count()` Δέχεται ως όρισμα μία τιμή και επιστρέφει τον αριθμό των στοιχείων της λίστας με αυτή την τιμή.

`.extend()` Δέχεται ως όρισμα μία λίστα, ή οποιαδήποτε άλλη ακολουθία, και προσθέτει τα στοιχεία της στο τέλος της υπάρχουσας λίστας.

`.index()` Δέχεται ως όρισμα μία τιμή και επιστρέφει τη θέση στη λίστα που αυτή εμφανίζεται για πρώτη φορά.

`.insert()` Δέχεται ως ορίσματα μία θέση στη λίστα και μια τιμή και εισάγει στη συγκεκριμένη θέση ένα στοιχείο με αυτή την τιμή. Το στοιχείο που τυχόν υπήρχε στη συγκεκριμένη θέση τοποθετείται στην επόμενη και όλα τα επόμενα σε μία επόμενη από την τρέχουσα θέση.

`.pop()` Δέχεται ως όρισμα μία θέση στη λίστα και απομακρύνει από τη λίστα το στοιχείο που βρίσκεται στη συγκεκριμένη θέση. Οι θέσεις των επόμενων στοιχείων στη λίστα τροποποιούνται κατάλληλα ώστε στη νέα λίστα να υπάρχει συνέχεια στην αρίθμηση των θέσεων (δεικτοδότηση). Χωρίς όρισμα απομακρύνει το τελευταίο στοιχείο στη λίστα.

`.remove()` Χωρίς όρισμα, διαγράφει το πρώτο στοιχείο της λίστας και τροποποιεί κατάλληλα τη δεικτοδότηση. Μπορεί να δεχτεί ως όρισμα τιμή, την οποία θα αναζητήσει στη λίστα και θα διαγράψει το στοιχείο στο οποίο θα τη συναντήσει για πρώτη φορά.

`.reverse()` Χωρίς όρισμα, αντιστρέφει τη σειρά των στοιχείων της λίστας. Επειδή η λίστα είναι μετατρέψιμη, το αποτέλεσμα της ενέργειας είναι μόνιμο.

`.sort()` Χωρίς όρισμα, ταξινομεί τα στοιχεία της λίστας. Αντίθετα από τη συνάρτηση sorted( ), η μέθοδος .sort( ) έχει μόνιμο αποτέλεσμα στη λίστα, η οποία είναι πλέον ταξινομημένη με αύξουσα σειρά. Μπορούμε επίσης να δώσουμε ως όρισμα reverse = True και τότε θα πάρουμε τη λίστα με φθίνουσα ταξινόμηση.

Ταξινόμηση και αναζήτηση

Εκτός από την συνάρτηση `sorted( )` και τη μέθοδο `.sort()` για την ταξινόμηση αλλά και τον τελεστή `in` και τις μεθόδους `.index( )` και `.count( )` για την αναζήτηση, υπάρχουν αρκετοί αλγόριθμοι ταξινόμησης και ιδιαίτερα για ταξινομημένες λίστες αρκετοί αλγόριθμοι αναζήτησης. Η ταξινόμηση με εισαγωγή (**insertion sort**) και η δυαδική αναζήτηση (**binary search**) αντίστοιχα είναι δύο τέτοια παραδείγματα.

7.3.4 Ερωτήσεις Κατανόησης

1. Η λίστα στην Python έχει μη τροποποιήσιμα δεδομένα.

ΝΑΙ ΟΧΙ

2. Στις λίστες δεν αποθηκεύουμε υποχρεωτικά ομοειδή δεδομένα.

ΝΑΙ ΟΧΙ

3. Η λίστα στην Python έχει μόνο δύο (2) μεθόδους.

ΝΑΙ ΟΧΙ

4. Η λίστα στην Python έχει δεδομένο μέγεθος (πλήθος στοιχείων) από τη στιγμή της δημιουργίας της.

ΝΑΙ ΟΧΙ

5. Ανάμεσα σε λίστες μπορούμε να εφαρμόσουμε συγκρίσεις;

ΝΑΙ ΟΧΙ

7.3.5 Ερωτήσεις Ανάπτυξης

1. Ποια είναι τα χαρακτηριστικά της λίστας στην Python;

2. Περιγράψτε τη συνένωση (concatenation) δύο λιστών στην Python.

3. Είναι, κατά την άποψή σας, η λίστα κατάλληλη δομή για εφαρμογές Internet of Things; Γιατί;

4. Περιγράψτε την απομάκρυνση ενός στοιχείου από μια λίστα.

5. Περιγράψτε τρεις (3) τρόπους για την αναζήτηση ενός στοιχείου σε μια λίστα.

6. Περιγράψτε την ταξινόμηση με εισαγωγή (insertion sort) χωρίς να χρησιμοποιήσετε κώδικα.

7. Περιγράψτε την δυαδική αναζήτηση (binary search) χωρίς να χρησιμοποιήσετε κώδικα.

7.3.6 Ασκήσεις

1. Η ταξινόμηση φυσαλίδας (bubble sort) ταξινομεί μια λίστα συγκρίνοντας επανειλημμένα διπλανά στοιχεία και ανταλλάσσοντάς τα αν η σειρά είναι λάθος. Ο αλγόριθμος διασχίζει τη λίστα πολλές φορές, κάθε φορά καταλήγοντας να μεταφέρει το εκάστοτε μεγαλύτερο αταξινόμητο στοιχείο στη σωστή του θέση προς το τέλος της λίστας (για αύξουσα ταξινόμηση). Αν έχουμε τον ακόλουθο κώδικα:

```
def bubbleSort(list):
    n = len(list)
    swapped = False
    for i in range(n-1):
        for j in range(0, n-i-1):
            if list[j] > list[j + 1]:
                swapped = True
                list[j], list[j + 1] = list[j + 1], list[j]
```

α. Σχολιάστε τι κάνει κάθε γραμμή κώδικα.

β. Εξηγείστε γιατί ο κώδικας χάνει χρόνο δηλ. δεν είναι βελτιστοποιημένος

γ. Αν ξέρουμε ότι ο κώδικας για την βελτιστοποίηση είναι ο ακόλουθος, εντοπίστε που πρέπει να μπεί (θέση και εσοχή):

```
if not swapped:
    return
```

2. Τρέξτε τον παραπάνω κώδικα για τη λίστα [60, 40, 30 20, 10, 50, 70, 80] με και χωρίς τη βελτιστοποίηση και μετρείστε πόσες φορές επαναλήφθηκε το κάθε for στην κάθε περίπτωση.

3. Μπορείτε να σκεφτείτε ένα τρόπο ταξινόμησης της παραπάνω λίστας που θα αξιοποιεί τη συνάρτηση max( ) και τη μέθοδο .index( ) για να ταξινομήσει τη λίστα κατά αύξουσα σειρά;

7.4 Η δομή του λεξικού

Ένα λεξικό (dictionary) διαφέρει από μια λίστα στο ότι δεν έχει ακέραια δεικτοδότηση. Μάλιστα, όχι μόνο οι δείκτες των στοιχείων του δεν είναι ακέραιοι αλλά επίσης δεν έχουν καν αρίθμηση. Αντίθετα έχουν ονόματα που λέγονται κλειδιά. Τα κλειδιά αυτά μπορούν να ανήκουν σε οποιοδήποτε μη τροποποιήσιμο τύπο.

Στην πράξη ένα λεξικό χρησιμοποιεί μια γενίκευση της έννοιας του δείκτη για να προσδιορίζει τα στοιχεία του. Εφόσον όμως δεν υπάρχει μια προϋπάρχουσα αριθμητική διάταξη των δεικτών (αυτή των ακέραιων αριθμών) όπως στις άλλες σύνθετες δομές που εξετάσαμε η σύνδεση μεταξύ του κλειδιού και της τιμής που προσδιορίζει πρέπει να γίνει από το ίδιο το λεξικό. Ο τρόπος που έχει επιλεγεί στην Python είναι να θεωρείται στοιχείο ενός λεξικού το ζεύγος ενός κλειδιού και μιας τιμής. Έτσι η διάταξη του στοιχείων ενός λεξικού είναι απλά αυτή της εισαγωγής των ζευγαριών σε αυτό. Η διάταξη μάλιστα ισχύει από την Python 3.7 και μετά καθώς σε προηγούμενες εκδόσεις τα λεξικά δεν ήταν καν διατεταγμένα.

Αν θέλουμε να είμαστε πιο τυπικοί το λεξικό είναι μια δομή δεδομένων με τα εξής χαρακτηριστικά:

- ✓ είναι σύνθετη, δηλαδή χρησιμοποιείται για την αποθήκευση πολλαπλών δεδομένων ταυτόχρονα
- ✓ είναι διατεταγμένη, δηλαδή η σειρά των μελών είναι συγκεκριμένη και είναι αυτή της εισαγωγής τους (ισχύει από την 3.7 και μετά)
- ✓ είναι μετατρέψιμη, δηλαδή τα στοιχεία του λεξικού μπορούν να προστεθούν, να αφαιρεθούν και να αλλάξουν και μετά την πρώτη εισαγωγή
- ✓ τα στοιχεία/ μέλη της αποτελούνται από ζεύγη κλειδιών και τιμών (δεδομένων). Τα δεδομένα μπορούν να είναι ετερογενή, δηλαδή πολλών διαφορετικών τύπων χωρίς περιορισμό, ενώ
- ✓ για τα κλειδιά υπάρχουν δύο περιορισμοί: να είναι μη τροποποιήσιμου τύπου και, κυρίως, να μην επαναλαμβάνονται τιμές κλειδιού.

Ας δούμε μερικά παραδείγματα δημιουργίας λεξικών :

```
# Τρόποι δημιουργίας λεξικών (dictionaries)
dict1 ={}    # Δημιουργία κενού λεξικού για να γεμίσουμε#
dict1['John'] = 'janitor'   # Προσθέτουμε ζευγάρια κλειδιού- τιμής
dict1['Mary'] = 'accountant'
dict1['George'] = 'guard'
dict1['Michael'] = 'programmer'
dict1['Jane'] = 'programmer'    # Ενώ το κλειδί δεν μπορεί να
# εμφανίζεται δύο φορές, η τιμή φυσικά μπορεί.
print(dict1)
{'John': 'janitor', 'Mary': 'accountant', 'George': 'guard', 'Michael': 'programmer',
'Jane': 'programmer'}
dict2 = { 'one' : 'uno', 'two' : 'due', 'three' : 'tre'}    # Απευθείας
# ορισμός με τα ζεύγη κλειδιών-τιμών. Προσέξτε ότι
```

```
# αντί για ίσον ( = ) χρησιμοποιούμε άνω-κάτω τελεία ( : )
dict3 = dict(one = 'unos', two = 'dos', three = 'tres')     # Ορισμός μέσω
# της συνάρτησης dict(). Προσέξτε ότι εδώ χρησιμοποιούμε
# και πάλι το ίσον ( = ) και τα κλειδιά δεν μπαίνουν σε
# εισαγωγικά.

print(type(dict1))    # Αν αναζητήσουμε τον τύπο της λίστας θα
# πάρουμε σε όλες τις περιπτώσεις:
<class 'dict'>
```

Επίσης μπορούμε να αξιοποιήσουμε την συνάρτηση `dict( )` εάν έχουμε είτε μια ακολουθία από πλειάδες των δύο είτε δύο ακολουθίες με ίδιο μέγεθος (σε συνδυασμό με τη `zip( )` σε αυτή την περίπτωση) για να δημιουργήσουμε ένα λεξικό.

```
list1 = [('one', 1),('two', 2), ('three', 3)]
dict3 = dict(list1)
print(dict3)
{'one': 1, 'two': 2, 'three': 3}
dict4 = dict(zip(['a', 'b', 'c'],[1,2,3]))
print(dict4)
{'a': 1, 'b': 2, 'c': 3}
```

7.4.1 Στοιχεία ενός λεξικού (πρόσβαση, διάσχιση)

Στην περίπτωση των λεξικών η πρόσβαση στα στοιχεία τους μπορεί να είναι τριών ειδών: πρόσβαση στα κλειδιά (keys), στις τιμές (values), ή συνολικά στα στοιχεία (items).

Η πρόσβαση στις τιμές είναι η ευκολότερη αφού – όπως όλες οι ακολουθίες – το λεξικό έχει κατασκευαστεί έχοντας υπόψη την πρόσβαση στις τιμές που αποθηκεύει. Αξιοποιεί μάλιστα για το σκοπό αυτό τα κλειδιά όπως οι προηγούμενες ακολουθίες που εξετάσαμε αξιοποιούν τους δείκτες θέσης. Μερικά παραδείγματα (υποθέτοντας το dict1 της προηγούμενης παραγράφου):

```
print( dict1[“John”])     # Εκτύπωση τιμής του κλειδιού “John”
janitor
```

Υπάρχει και η μέθοδος get( ) του λεξικού με την οποία επίσης δίνεται πρόσβαση στις τιμές, με παρόμοιο τρόπο:

```
print( dict1.get(“Michael”))
programmer
```

Για πρόσβαση γενικά στα κλειδιά ή στις τιμές το λεξικό μας προσφέρει δύο ακόμα μεθόδους τις .keys( ) και .values( ). Η πρώτη μας επιστρέφει όλα τα κλειδιά και η δεύτερη όλες τις τιμές του λεξικού μας. Επιστρέφουν μια συλλογή κλειδιών με όνομα dict_keys και dict_values αντίστοιχα. Ατυχώς αυτά έχουν οριστεί ως αντικείμενα με το αντίστοιχο όνομα και δεν είναι ορισμένα με τρόπο ώστε να διασχίζονται με βάση τη θέση Φυσικά οι τιμές δεν θα μπορούσαν εύκολα να οργανωθούν έτσι εφόσον επιτρέπουν την επανάληψη και οι ακολουθίες στην Python όχι . Ας τα δούμε όμως λίγο:

```
print( dict1.keys( ))     # Ζητάμε τα κλειδιά της λίστας μας
dict_keys(['John', 'Mary', 'George', 'Michael', 'Jane'])
print( dict1.values( ))     )))     # Ζητάμε τις τιμές της λίστας μας
dict_values(['janitor', 'accountant', 'guard', 'programmer', 'programmer'])
print( type(dict1.keys( )))     # Ζητάμε τον τύπο του dict1.keys
<class 'dict_keys'>
```

Για να επισκεφτούμε με τη σειρά (διάσχιση) τα στοιχεία ενός λεξικού μπορούμε να χρησιμοποιήσουμε ένα βρόγχο σταθερού μεγέθους (δηλαδή ένα βρόγχο for). Η διάσχιση μπορεί να γίνει ως προς τα κλειδιά, ως προς τις τιμές ή ως προς τα στοιχεία.

Αν δοκιμάσουμε πρώτα με το κλειδί, κάθε κλειδί στο λεξικό μας θα ανατεθεί διαδοχικά στη μεταβλητή που ορίστηκε για αυτό το σκοπό, επιτρέποντας τη χρησιμοποίησή του μέσα στο βρόγχο:

```
for x in dict1.keys( ):     # Το x διασχίζει τα κλειδιά του
# λεξικού μας
      print(x)
      print(dict1[x])     # Για να βρούμε την τιμή πρέπει να πάμε
# μέσω του κλειδιού
John janitor Mary accountant George guard Michael programmer Jane programmer
for x in dict1:     # Το x και πάλι διασχίζει τα κλειδιά του
# λεξικού μας
```

```
      print(x)
      print(dict1[x])
# Με το ίδιο ακριβώς αποτέλεσμα:
John janitor Mary accountant George guard Michael programmer Jane programmer
```

Με έμμεσο τρόπο – από το κλειδί – μας δόθηκε πρόσβαση και στην αντίστοιχη τιμή. Ας δούμε όμως τι γίνεται αν διασχίσουμε τις τιμές:

```
for y in dict1.values( ):    # Το y διασχίζει τις τιμές του λεξικού μας
      print(x)
janitor accountant guard programmer programmer
```

Δεν υπάρχει όμως τώρα άμεσος τρόπος να εντοπίσουμε πιο είναι το αντίστοιχο κλειδί, οπότε δεν μπορούμε άμεσα να το επεξεργαστούμε.

Υπάρχουν όμως τρόποι να διασχίσουμε τα στοιχεία (items) του λεξικού μας. Τον απλούστερο μας τον προσφέρει μία ακόμα μέθοδος του αντικειμένου λεξικό η `.items( )`.

```
for x,y in dict1.items( ):    # Το x και πάλι διασχίζει τα κλειδιά
# ενώ το y διασχίζει τις τιμές του λεξικού μας
      print(x ,y)
John janitor Mary accountant George guard Michael programmer Jane programmer
# Αν όμως δε χρειαζόμαστε να επεξεργαστούμε
# ξεχωριστά κλειδί και τιμή, υπάρχει απλούστερη λύση:
for x in dict1.items( ):    # Το x αυτή τη φορά διασχίζει τα
# στοιχεία του λεξικού μας
      print(x )
('John', 'janitor') ('Mary', 'accountant') ('George', 'guard') ('Michael', 'programmer')
('Jane', 'programmer')
```

7.4.2 Βασικές λειτουργίες λεξικού, πράξεις και συναρτήσεις που εφαρμόζονται σε αυτό

Τα λεξικά παρά το γεγονός ότι είναι πλέον διατεταγμένα, δεν αντιμετωπίζονται στην Python ως ακολουθίες. Ως εκ τούτου οι πράξεις που εφαρμόζονται σε όλες τις ακολουθίες (iterables) δεν εφαρμόζονται σε αυτά. Επομένως δεν υπάρχει συνένωση (concatenation) ούτε και πολλαπλασιασμός λεξικών. Επίσης δεν υπάρχει δεικτοδότηση (πέραν των κλειδιών) και κατάτμηση. Αυτό δεν σημαίνει ότι δεν προσφέρονται άλλες δυνατότητες. Ας δούμε μερικές:

Έλεγχος συμπερίληψης

Μπορούμε να ελέγξουμε τρία πράγματα: αν ένα στοιχείο ανήκει σε ένα λεξικό, αν ένα κλειδί περιλαμβάνεται στα κλειδιά του λεξικού και αν μια τιμή εμφανίζεται στις τιμές του λεξικού. Μπορούμε επίσης να ελέγξουμε και το αντίθετο. Αυτό γίνεται με την εφαρμογή των τελεστών `in` και `not in` για έλεγχο συμπερίληψης ή μη αντίστοιχα. Τον πρώτο τελεστή τον έχουμε ήδη χρησιμοποιήσει στη διάσχιση του λεξικού. Αντίστοιχα λειτουργεί και ο not in.

```
print("Simon" not in dict1)
# Χρησιμοποιήσαμε εδώ τη «σύντομη» διατύπωση για τα κλειδιά
```

Συγκρίσεις

Δυο λεξικά μπορούν να συγκριθούν ως προς την ισότητα (τελεστής ==) και βρίσκονται ίσα (True) τότε και μόνο τότε αν έχουν ακριβώς τα ίδια ζεύγη κλειδιών-τιμών (περιέργως ασχέτως σειράς). Σε κάθε άλλη περίπτωση κρίνονται άνισα (False). Επίσης, κάθε άλλη σύγκριση, ανάλογα με την έκδοση Python που χρησιμοποιείται θα δώσει False ή Error.

```
dict1 = {'one':'uno', 'two':'due', 'three':'tre'}
dict2 = {'three':'tre', 'one':'uno', 'two':'due'}
print(dict1 == dict2)
dict2['three'] = 'tres'
print(dict1 == dict2)
```

Συναρτήσεις

Μια σειρά από συναρτήσεις εφαρμόζονται στα λεξικά. Αυτές περιλαμβάνουν την εύρεση του μήκους του λεξικού δηλαδή τη συνάρτηση `len( )` και, το μέγιστο και ελάχιστο με τις συναρτήσεις `max( )`και `min( )` αντίστοιχα. Οι τελευταίες επιστρέφουν όμως το μέγιστο και ελάχιστο **κλειδί**. Ακόμα η, επίσης ενσωματωμένη, συνάρτηση `sorted( )` επιστρέφει μια ταξινομημένη λίστα με τα κλειδιά του λεξικού μας αν δεν επιλέξουμε κλειδιά, τιμές ή στοιχεία. Αν επιλέξουμε `.keys( )` ή `.items( )` η διάταξη είναι με βάση τις τιμές των κλειδιών. Αν επιλέξουμε `.values( )` παίρνουμε διατεταγμένες τις τιμές. Ας δούμε κι εδώ κάποια παραδείγματα θεωρώντας ήδη ορισμένα τα λεξικά της προηγούμενης ενότητας: sorted, dict, set

```
dict1 = {'one':'uno', 'two':'due', 'three':'tre'}
dict2 = {'three':'tre', 'one':'uno', 'two':'due'}
print(dict1.keys())
```

```
dict_keys(['one', 'two', 'three'])
print(max(dict1))
two     # επειδή τα συγκρίνει αλφαβητικά και όχι ως αριθμούς, οπότε 'w' > 'h'
print(sorted(dict2.values()))
['due', 'tres', 'uno']
print(sorted(dict2.items()))
[('one', 'uno'), ('three', 'tres'), ('two', 'due')]
```

Τέλος μπορούμε να χρησιμοποιήσουμε τη συνάρτηση set( ) στην περίπτωση που θέλουμε, ας πούμε, να δούμε τις τιμές του λεξικού μας χωρίς επαναλήψεις.

```
dict2['two'] = 'tres'
set1 = set(dict2.values())
print( set1 )
{'uno', 'tres'}
# Το ένα 'tres'παραλείπεται από το σύνολο
```

Μέθοδοι

Και εδώ για να δούμε τις μεθόδους ενός αντικειμένου μπορούμε να χρησιμοποιήσουμε την ενσωματωμένη συνάρτηση dir( ). Μας ενδιαφέρουν οι φανερές χωρίς διπλή κάτω παύλα. Έχουμε ήδη συναντήσει τις .keys( ), .values( ), .items( ) και .get( ). Για λόγους πληρότητας θα τις αναφέρουμε όλες:

.copy( ) Χωρίς όρισμα, επιστρέφει ένα αντίγραφο της λίστας (με αντιγραφή by value).

.clear( ) Απομακρύνει όλα τα στοιχεία (items) από το λεξικό

.get( ) Με όρισμα ένα κλειδί. Επιστρέφει την τιμή που αντιστοιχεί στο κλειδί που δίνεται ως όρισμα αλλιώς επιστρέφει KeyError!. Μπορεί όμως να πάρει και δεύτερο όρισμα (υποχρεωτικά τη λέξη default ή την έκφραση default = τιμή) – με αυτόν τον τρόπο μπορούμε να δώσουμε μια τιμή για την περίπτωση που το κλειδί δεν υπάρχει στο λεξικό. Αν δεν έχει δοθεί τιμή, επιστρέφει none. Με αυτόν τον τρόπο δεν επιστρέφει ποτέ KeyError!

.items( ) Με δύο παραμέτρους, ένα κλειδί και μια τιμή. Μετατρέπει τα ζευγάρια κλειδιού-τιμής σε πλειάδες των δύο και επιστρέφει μια συλλογή. Η συλλογή μπορεί να χρησιμοποιηθεί σε επαναλήψεις (iterable).

.keys( ) Χωρίς παραμέτρους. Επιστρέφει μια διατρέξιμη συλλογή των κλειδιών του λεξικού μας

.values( ) Χωρίς παραμέτρους. Επιστρέφει μια διατρέξιμη συλλογή των τιμών του λεξικού μας

.pop( ) Με μία παράμετρο. Απομακρύνει από το λεξικό το ζευγάρι κλειδιού-τιμής του οποίου το κλειδί έχει δοθεί ως όρισμα

.popitem( ) Απομακρύνει από το λεξικό το ζευγάρι κλειδιού-τιμής που προστέθηκε τελευταίο

.set_default( ) Με δύο παραμέτρους, ένα κλειδί και μια τιμή. Επιστρέφει την τιμή που αντιστοιχεί στο κλειδί όρισμα. Αν το κλειδί δεν υπάρχει στο λεξικό τότε προσθέτει το κλειδί (πρώτο όρισμα) με τιμή αυτήν που δόθηκε (δεύτερο όρισμα).

`.update( )` Με μία παράμετρο, ένα λεξικό, συνήθως με ένα κλειδί και μια τιμή μόνο. Προσθέτει το ζεύγος κλειδιού και τιμής που λαμβάνει ως ορίσματα στο τέλος του λεξικού. Στην περίπτωση των πολλαπλών ζευγών λειτουργεί σαν την συνένωση (concatenation) ακολουθιών: προσθέτει το λεξικό που παίρνει ως όρισμα μετά από το λεξικό που καλεί τη συνάρτηση.

```
worker = {‘name’:( ‘John’, ‘Daglas’), ‘age’: 43, ‘salary’: 30000 }
print(worker.get(‘name’))
('John', 'Daglas')   # Επιστρέφει την πλειάδα που δώσαμε
ως τιμή στο ‘name’
worker.update({ ‘office’: ‘central’})    # Προσέξτε τις εσωτερικές
# αγκύλες
print(worker)    # Θα τυπώσει το νέο λεξικό
{'name': ('John', 'Daglas'), 'age': 43, 'salary': 30000, 'office': 'central'}
worker.popitem( )
print(worker)    # Θα τυπώσει το νέο λεξικό
{'name': ('John', 'Daglas'), 'age': 43, 'salary': 30000}
worker.set_default(‘office’: ‘central’)    # Προσέξτε ότι δε
# χρειάζεται εσωτερικές αγκύλες (αφού δέχεται μόνο ένα
# ζεύγος)
print(worker)    # Θα τυπώσει το νέο λεξικό
{'name': ('John', 'Daglas'), 'age': 43, 'salary': 30000, 'office': 'central'}
worker.clear( )
{}
```

7.4.3 Να θυμάμαι

Λεξικά

Το λεξικό είναι μια δομή δεδομένων με τα εξής χαρακτηριστικά:

- ✓ είναι σύνθετη, δηλαδή χρησιμοποιείται για την αποθήκευση πολλαπλών δεδομένων ταυτόχρονα
- ✓ είναι διατεταγμένη, δηλαδή η σειρά των μελών είναι συγκεκριμένη και είναι αυτή της εισαγωγής τους (ισχύει από την 3.7 και μετά)
- ✓ είναι μετατρέψιμη, δηλαδή τα στοιχεία του λεξικού μπορούν να προστεθούν, να αφαιρεθούν και να αλλάξουν και μετά την πρώτη εισαγωγή
- ✓ τα στοιχεία/ μέλη της αποτελούνται από ζεύγη κλειδιών και τιμών (δεδομένων). Τα δεδομένα μπορούν να είναι ετερογενή, δηλαδή πολλών διαφορετικών τύπων χωρίς περιορισμό, ενώ
- ✓ για τα κλειδιά υπάρχουν δύο περιορισμοί: να είναι μη τροποποιήσιμου τύπου και, κυρίως, να μην επαναλαμβάνονται τιμές κλειδιού.

Δημιουργία λεξικού

Ανήγοντας ένα κενό λεξικό και ενημερώνοντας με τιμές για συγκεκριμένα κλειδιά:

```
dict1 ={}     # Δημιουργία κενού λεξικού για να γεμίσουμε#
dict1['John'] = 'janitor'    # Προσθέτουμε ζευγάρια κλειδιού- τιμής
dict1['Mary'] = 'accountant'
dict1['George'] = 'guard'
```

Με παράθεση ζευγών κλειδιού-τιμής:

```
dict2 = { 'one' : 'uno', 'two' : 'due', 'three' : 'tre'}
```

Χρησιμοποιώντας τη συνάρτηση `dict( )`

```
dict3 = dict(one = 'unos', two = 'dos', three = 'tres')
```

Συνδιάζοντας την συνάρτηση `dict( )`με την συνάρτηση `zip( )` και εφαρμόζοντας τον συνδυασμό σε δύο ισομεγέθεις ακολουθίες οποιουδήποτε τύπου.

```
dict4 = dict(zip(['a', 'b', 'c'],[1,2,3]))
```

Οι τρόποι δεν εξαντλούνται εδώ.

Διάσχιση ενός λεξικού

Τα λεξικά μας προσφέρουν τρεις μεθόδους, τις `.keys( )`,`.values( )` και `.items( )`, για τη διάσχισή τους. Οι `.keys( )` και `.items( )` διασχίζουν μέσω των κλειδιών και προσφέρουν πρόσβαση και στις τιμές, οι πρώτη έμμεσα και η δεύτερη άμεσα. Η `.values( )` μας επιτρέπει τη διάσχιση των τιμών, χωρίς όμως ταυτόχρονα να προσφέρει εύκολη πρόσβαση στα κλειδιά.

Και στις τρεις περιπτώσεις αξιοποιείται ο τελεστής `in` μέσα σε ένα βρόγχο σταθερών επαναλήψεων (`for`). Προφανώς είναι εφικτός έλεγχος συμπερίληψης.

Συγκρίσεις

Η μοναδική σύγκριση που έχει νόημα μεταξύ λεξικών είναι ως προς την ισότητα (`==`). Δύο λεξικά βρίσκονται ίσα (True) τότε και μόνο τότε αν έχουν ακριβώς τα ίδια ζεύγη κλειδιών-τιμών (ασχέτως σειράς)

Συναρτήσεις

Μια σειρά από ενσωματωμένες συναρτήσεις εφαρμόζονται σε όλες τις ακολουθίες, άρα και στα λεξικά. Αυτές περιλαμβάνουν την εύρεση του μήκους της ακολουθίας δηλαδή τη συνάρτηση `len( )` και, εφόσον οι τιμές είναι συγκρίσιμες, το μέγιστο και ελάχιστο στοιχείο με τις `max( )` και `min( )` αντίστοιχα. Εφαρμόζεται και η συνάρτηση `dir( )` που επιστρέφει όλες τις μεθόδους της δομής δεδομένων ως αντικείμενο. Επίσης εφαρμόζεται η συνάρτηση `sorted( )` που επιστρέφει μια ταξινομημένη λίστα με τα κλειδιά του λεξικού μας αν δεν επιλέξουμε κλειδιά, τιμές ή στοιχεία. Αν επιλέξουμε `.keys( )` ή `.items( )` η διάταξη είναι αυτή των κλειδιών. Αν επιλέξουμε `.values( )` παίρνουμε διατεταγμένες τις τιμές.

Μέθοδοι

Ένα λεξικό ως δομή , έχει δέκα (10) δημόσιες μεθόδους τις εξής:

`.copy( )` Χωρίς όρισμα, επιστρέφει ένα αντίγραφο της λίστας (με αντιγραφή by value).

`.clear( )` Απομακρύνει όλα τα στοιχεία (items) από το λεξικό

`.get( )` Επιστρέφει την τιμή που αντιστοιχεί στο κλειδί- όρισμα. Αν χρησιμοποιήσουμε `default` τότε δεν επιστρέφει KeyError! Επιστρέφει την τιμή που δώσαμε ή none.

`.items( )` Με δύο παραμέτρους, ένα κλειδί και μια τιμή. Μετατρέπει τα ζευγάρια κλειδιού-τιμής σε πλειάδες των δύο και επιστρέφει μια συλλογή που μπορεί να χρησιμοποιηθεί σε επαναλήψεις (iterable)

`.keys( )` Χωρίς παραμέτρους. Επιστρέφει μια διατρέξιμη συλλογή των κλειδιών του λεξικού.

`.values( )` Χωρίς παραμέτρους. Επιστρέφει μια διατρέξιμη συλλογή των τιμών του λεξικού.

`.pop( )` Με μία παράμετρο. Απομακρύνει από το λεξικό το ζευγάρι κλειδιού-τιμής του οποίου το κλειδί έχει δοθεί ως όρισμα

`.popitem( )` Απομακρύνει από το λεξικό το ζευγάρι κλειδιού-τιμής που προστέθηκε τελευταίο

`.set_default( )` Με δύο παραμέτρους, ένα κλειδί και μια τιμή. Επιστρέφει την τιμή που αντιστοιχεί στο κλειδί όρισμα. Αν το κλειδί δεν υπάρχει στο λεξικό τότε προσθέτει το κλειδί (πρώτο όρισμα) με τιμή αυτήν που δόθηκε (δεύτερο όρισμα).

`.update( )` Με μία παράμετρο, ένα λεξικό, συνήθως με ένα κλειδί και μια τιμή μόνο. Προσθέτει το ζεύγος κλειδιού και τιμής που λαμβάνει ως όρισμα στο τέλος του λεξικού. Στην περίπτωση των πολλαπλών ζευγών λειτουργεί σαν την συνένωση (concatenation) ακολουθιών: προσθέτει το λεξικό που παίρνει ως όρισμα μετά από το λεξικό που καλεί τη συνάρτηση..

7.4.4 Ερωτήσεις Κατανόησης

1. Οι τιμές του λεξικού δεν αλλάζουν μετά την πρώτη εισαγωγή.

ΝΑΙ ΟΧΙ

2 Το λεξικό έχει δέκα (1) δημόσιες μεθόδους.

ΝΑΙ ΟΧΙ

4. Τα λεξικά ορίζονται μόνο με ζευγάρια κλειδιών – τιμών (παράθεση στοιχείων).

ΝΑΙ ΟΧΙ

5. Το λεξικό διασχίζεται με τα κλειδιά του μόνο.

ΝΑΙ ΟΧΙ

6. Η σύγκριση λεξικών έχει νόημα μόνο ως προς την ισότητα.

ΝΑΙ ΟΧΙ

7.4.5 Ερωτήσεις Ανάπτυξης

1. Ποια είναι τα χαρακτηριστικά του λεξικού στην Python;

2. Περιγράψτε τον τρόπο που ελέγχεται η ισότητα στην περίπτωση των λεξικών

3. Περιγράψτε τον έλεγχο συμπερίληψης σε λεξικό.

4. Περιγράψτε τους τρεις κύριους τρόπους διάσχισης ενός λεξικού. Μπορείτε να εντοπίσετε πλεονεκτήματα ή και μειονεκτήματα;

7.4.6 Ασκήσεις

1. Στο πρώτο παράδειγμα της παραγράφου (7.3) χρησιμοποιήσαμε τα ονόματα υπαλλήλων ως κλειδιά και τους ρόλους τους στην επιχείρηση ως τιμές. Προφανώς παραπάνω από ένα άτομα μπορούν να έχουν τον ίδιο ρόλο. Έτσι αν χρησιμοποιηθούν οι ρόλοι ως κλειδιά μπορούμε να έχουμε ως τιμές μια λίστα με τα ονόματα. Σκεφτείτε μια επιχείρηση με 10 υπαλλήλους, προσδιορείστε τους πιθανούς ρόλους και ενημερώστε το σχετικό λεξικό έτσι ώστε οι εργαζόμενοι με τον ίδιο ρόλο να εισάγονται σε μία τιμή λίστα για το κλειδί αυτό. Τυπώστε το ενημερωμένο λεξικό σας.

2. Υποθέστε ότι δύο ομάδες ποδοσφαίρου ενοποιούνται. Επιλέξτε τις αγαπημένες σας. Ευτυχώς τα λεξικά για τους παίχτες τους είναι οργανωμένα με τον ίδιο τρόπο, που περιγράφεται στην προηγούμενη άσκηση. Τα κλειδιά είναι και στις δύο περιπτώσεις μέλη της λίστας [‘επιθετικός’, ‘μέσος’, ‘αμυντικός’, ‘τερματοφύλακας’]. Οι παίκτες είναι τουλάχιστον 15 για την κάθε ομάδα.

α. Ενημερώστε τα λεξικά των ομάδων και ενώστε τα για να εξυπηρετείται η νέα.

β. Αν ο προπονητής έχει εντοπίσει έναν ‘λίμπερο’ με το όνομα ‘Ταχυπόδης’ μπορούμε να τον προσθέσουμε και πώς;

γ. τυπώστε το τελικό λεξικό.

3. Κατασκευάστε ένα λεξικό με κλειδί μια πλειάδα με το ονοματεπώνυμο και τιμή μια λίστα με τα τηλέφωνα των φίλων σας. Εισάγετε τους πλησιέστερους. Είναι εξυπηρετική η οργάνωση αυτή;

7.5 Η δομή του συνόλου

Ένα σύνολο έχει μία κύρια διαφορά από μία πλειάδα: είναι αδιάτακτο. Όχι μη διατεταγμένο αλλά αδιάτακτο, επειδή αυτή είναι μια επιλογή που έχει γίνει με πρόθεση. Η επιλογή προσφέρει μεγάλα πλεονεκτήματα σε επίπεδο ταχύτητας αλλά εδώ δεν θα ασχοληθούμε με αυτά. Μας ενδιαφέρει περισσότερο να εντοπίσουμε πράγματα που γίνονται ευκολότερα με ένα σύνολο από ότι με μια πλειάδα ή μια λίστα.

Στην πραγματικότητα υπάρχουν περισσότερες διαφορές, αν και κάποιες είναι άμεση συνέπεια αυτής της πρώτης. Για παράδειγμα σε ένα σύνολο δεν επιτρέπονται επαναλήψεις. Καθόλου. Σε βαθμό που το True και το 1, πιθανότατα επειδή λειτουργούν με τον ίδιο τρόπο σε λογικές εκφράσεις, θεωρούνται επανάληψη και δεν μπορούν να υπάρχουν ταυτόχρονα σε ένα σύνολο.

Υπάρχουν όμως και άλλες. Για παράδειγμα το σύνολο είναι ανάποδο από την πλειάδα σε τούτο: το ίδιο είναι τροποποιήσιμο (mutable) αφού μπορείς να αφαιρέσεις ή να προσθέσεις στοιχεία. Τα στοιχεία του όμως είναι μη τροποποιήσιμα: από τη στιγμή που θα βάλεις ένα στοιχείο σε ένα σύνολο μπορείς μόνο να το βγάλεις, όχι να το τροποποιήσεις. Γι αυτό και θα βρεις πολλές αναφορές να προσπαθούν να το περιγράψουν με μια λέξη όπως παράδειγμα 'αμετάβλητο' (unchangeable).

Με βάση τα παραπάνω το σύνολο είναι ένας ενσωματωμένος σύνθετος τύπος δεδομένων που περιλαμβάνει ένα μεταβλητό αριθμό αμετάβλητων στοιχείων. Επιπλέον τα στοιχεία (μέλη) ενός συνόλου οφείλουν να είναι και συγκρίσιμα – μεταξύ τους και με άλλα δεδομένα. Μπορούμε να πάμε σε μεγάλο βάθος περιγραφής χρησιμοποιώντας όρους όπως hashable και μεθόδους **__hash__** και **__eq__** αλλά η ουσία του πράγματος είναι αυτή.

Αν θέλουμε να είμαστε πιο τυπικοί το σύνολο είναι μια δομή δεδομένων με τα εξής χαρακτηριστικά:

- ✓ είναι σύνθετη, δηλαδή χρησιμοποιείται για την αποθήκευση πολλαπλών δεδομένων ταυτόχρονα
- ✓ είναι αδιάτακτη, δηλαδή η σειρά των μελών της είναι αδιάφορη
- ✓ είναι τροποποιήσιμη (mutable), με την έννοια ότι μπορούμε να προσθέσουμε και να αφαιρέσουμε στοιχεία όποτε θέλουμε
- ✓ τα στοιχεία/ δεδομένα της, από την άλλη, είναι μη τροποποιήσιμα (immutable)
- ✓ δεν επιτρέπεται η επανάληψη στοιχείου
- ✓ τα στοιχεία της πρέπει να είναι συγκρίσιμα (ισχύει και για τις ακολουθίες αλλά όχι για τα λεξικά)
- ✓ τα στοιχεία/ δεδομένα της μπορούν να είναι ετερογενή

Αυτό τα δύο τελευταία χαρακτηριστικά, σε συνδυασμό με μια πληθώρα μεθόδων αλλά και την ταχύτητα των υλοποιήσεών τους, είναι που καθιστούν το σύνολο ένα πολυεργαλείο για την Python.

Ας δούμε μερικά παραδείγματα διαφορετικών τρόπων δημιουργίας συνόλων (sets):

```
# Τρόποι δημιουργίας συνόλων (sets)
set1 = {1, 2, 3, 4, 5, 6}    # Κλασική δημιουργία συνόλου (set)
# με αγκύλες και απαρίθμηση των στοιχείων του
```

```
set2 = set([10, 20, 30])    # Εναλλακτικός τρόπος δημιουργίας
# συνόλου χρησιμοποιώντας τη συνάρτηση set() και
# κάποια ακολουθία.
set3 = {5}    # Αρχική δημιουργία ενός υποσυνόλου
# Προσοχή: Δεν μπορεί να είναι το κενό γιατί,
# περιέργως, θα το θεωρήσει λεξικό
set3.add(10)
set3.add(15)    # Διαδοχικές προσθήκες στοιχείων με τη
# μέθοδο .add()
print(set1)    # Ακολουθούν εκτυπώσεις των τριών συνόλων
{1, 2, 3, 4, 5, 6}
print(set2)
{10, 20, 30}
print(set3)
{10, 5, 15}
print(type(set3))
<class 'set'>
```

Προσοχή! Όπως αναφέραμε τα στοιχεία ενός συνόλου δεν μπορεί να είναι τροποποιήσιμα. Δεδομένου ότι ένα σύνολο είναι τροποποιήσιμο συμπεραίνουμε ότι δεν μπορούμε να έχουμε π.χ. ένα σύνολο συνόλων. Επίσης, δεν μπορούμε να χρησιμοποιήσουμε ένα σύνολο ως κλειδί σε ένα λεξικό. Επειδή όμως και οι δύο αυτές ιδιαίτερες περιπτώσεις έχουν ένα ενδιαφέρον υπάρχει ένας επιπλέον τύπος συνόλων που είναι μη τροποποιήσιμος (immutable) και χρησιμοποιείται ειδικά για αυτές τις περιπτώσεις. Τα σύνολα αυτά ονομάζονται πολύ σωστά frozensets. Δεν θα μας απασχολήσουν άλλο εδώ απλά αν συναντήσουμε κάπου απρόοπτα ένα σύνολο, γνωρίζουμε ότι αυτό θα ανήκει στη συγκεκριμένη κατηγορία.

7.5.1 Στοιχεία ενός συνόλου (πρόσβαση, διάσχιση)

Εφόσον το σύνολο είναι αδιάτακτο δεν μπορούμε να έχουμε πρόσβαση στα στοιχεία του μέσα από ένα δείκτη ή κλειδί ή κάποιο άλλο προσδιοριστικό θέσης. Τα στοιχεία του δεν έχουν κάποια δεδομένη σειρά. Μπορούμε όμως να χρησιμοποιήσουμε το γνωστό βρόγχο for σε συνδυασμό με τον τελεστή αναζήτησης in. Κάθε στοιχείο του συνόλου, με τυχαία σειρά, θα ανατεθεί διαδοχικά στη μεταβλητή που ορίστηκε για αυτό το σκοπό, επιτρέποντας τη χρησιμοποίησή του μέσα στο βρόγχο:

```
new_set = {'pencil', 'pen', 'rubber', 'book'}
for x in new_set:
      print(x)
pen rubber pencil book
```

Η σειρά, εάν το ξανατρέξουμε, θα είναι διαφορετική.

Ένας άλλος τρόπος για πρόσβαση σε όλα τα στοιχεία ενός συνόλου είναι η συνάρτηση iter( ). Τη συνάρτηση αυτή μπορούμε να χρησιμοποιήσουμε σε κάθε συλλογή. Δεν την αναφέραμε στις περιπτώσεις πλειάδας, λίστας και λεξικού γιατί εκεί είχαμε ποικιλία τρόπων πρόσβασης. Ας δούμε πώς δουλεύει. Υποθέτουμε το ίδιο σύνολο new_set.

```
new_set_iter = iter(new_set)
for i in range(len(new_set)):    # Χρησιμοποιούμε το μέγεθος του
# συνόλου - που ονομάζεται μήκος (length) μόνο για λόγους
# ομοιομορφίας – γιατί ο iterator που κατασκευάσαμε με
# τη συνάρτηση iter( ) δεν έχει τέτοιο χαρακτηριστικό.
      print(next(new_set_iter))    # Η next δίνει το επόμενο στοιχείο
# του iterator
pen rubber pencil book
```

7.5.2 Βασικές λειτουργίες συνόλου, πράξεις και συναρτήσεις που εφαρμόζονται σε αυτό

Για τα σύνολα υπάρχει ένας μεγάλος αριθμός πράξεων που τα αφορά. Επίσης, όπως είδαμε ήδη στην περίπτωση του μεγέθους (len( )), οι συναρτήσεις που εφαρμόζονται στις ακολουθίες συχνά εφαρμόζονται και στα σύνολα. Σε ένα βαθμό αφορούν όλες τις συλλογές απλά έχουν πάρει τα ονόματά τους από τις ακολουθίες. Λέμε μήκος και εννοούμε μέγεθος, μεγαλύτερο και εννοούμε επόμενο στην κατάταξη (συχνά αλφαβητική) και ούτω καθεξής. Τέλος υπάρχει ένας μεγάλος αριθμός δημόσιων μεθόδων των συνόλων που κάποιες από αυτές αποτελούν εναλλακτικές υλοποιήσεις των πράξεων μεταξύ συνόλων, κάποιες εξυπηρετούν τη διαχείριση των στοιχείων τους, δημιουργούν αντίγραφα και λοιπά.

Πράξεις για σύνολα

Οι πράξεις μεταξύ συνόλων, όπως η ένωση, η τομή, οι έλεγχοι για υποσύνολα και υπερσύνολα αλλά και άλλες εκδοχές λογικών πράξεων που υλοποιούνται με σύνολα έχουν όπως είπαμε τουλάχιστον δύο τρόπους υλοποίησης: με τελεστές και με μεθόδους (της κλάσης του) συνόλου. Ας δούμε μερικά παραδείγματα της πρώτης περίπτωσης:

```
set1 = { 'pencil', 'pen', 'rubber', 'book' }
set2 = { 'diary', 'book'}
set3 = { 'pen', 'ruler'}
set3 = set1 | set2    # Η ένωση των δύο συνόλων
set4 = set1 & set2    # Η τομή των δύο συνόλων
print(set4)
{'book'}
for x in set3:
     print(x)
diary pencil pen rubber book
print(max(set1))    # Επιστρέφει τη μέγιστη αλφαριθμητικά
# τιμή στο πρώτο σύνολο
rubber
set5 = set1 - set2 - set3
print(set5)    # Το σύνολο είναι κενό οπότε θα μας το γράψει
# αντί να τυπώσει ένα κενό
set()
set6 = { 1, 'one'}
print(max(set5))    # Θα επιστρέψει ValueError αφού στη max() δώσαμε όρισμα
κενό σύνολο
print(max(set6))    # Θα επιστρέψει TypeError: αφού ανάμεσα σε
# 'str' και 'int' τύπου στοιχεία δεν προβλέπεται
# σύγκριση
```

Να σημειωθεί ότι τελεστές όπως | της ένωσης, - της διαφοράς, ^ της συμμετρικής διαφοράς και & της τομής, εφαρμόζονται και επαναληπτικά μέσα στις συνολοθεωρητικές εκφράσεις της Python.

Έλεγχος συμπερίληψης

Μπορούμε επίσης να ελέγξουμε αν ένα στοιχείο ανήκει σε ένα σύνολο ή και το αντίθετο όπως και για τις υπόλοιπες δομές που αποτελούν συλλογές στοιχείων. Αυτό γίνεται με την εφαρμογή των γνωστών τελεστών in και not in για έλεγχο συμπερίληψης ή μη αντίστοιχα. Κατά τα γνωστά, οι δύο τελεστές εφαρμόζονται ανάμεσα σε ένα πιθανό στοιχείο και ένα σύνολο, δημιουργώντας μία παράσταση που μπορεί να είναι είτε αληθής είτε ψευδής. Όπως και για κάθε άλλη συλλογή στοιχείων, μπορεί να χρησιμοποιηθεί ως συνθήκη, για παράδειγμα σε μία εντολή επιλογής ή επανάληψης (κυρίως όταν πρόκειται για in). Ας δούμε άλλο ένα παράδειγμα:

```
for x in set1
     print(x)  # Θα τυπώσει διαδοχικά 'pencil','pen','rubber','book'
```

Συγκρίσεις

Οι συγκριτικοί τελεστές μπορούν να εφαρμοστούν και στα σύνολα. Έχουν όμως μια διαφορετική σημασία – στην πραγματικότητα ελέγχουν τη σχέση των συνόλων. Συγκεκριμένα, θεωρώντας τους ορισμούς συνόλων από την παράγραφο «Πράξεις για σύνολα», ας δούμε μερικά παραδείγματα:

```
set1 == set2    # Ελέγχει αν τα δύο σύνολα έχουν ακριβώς τα
# ίδια στοιχεία
set1 != set2    # Ελέγχει αν τα δύο σύνολα έχουν τουλάχιστον
# ένα στοιχείο διαφορετικό.
set1 <= set2    # Ελέγχει αν το set1 είναι υποσύνολο του set2,
# δηλαδή αν κάθε στοιχείο του set1 περιέχεται στο set2
set1 < set2    # Ελέγχει αν το set1 είναι γνήσιο υποσύνολο
# του set2, δηλαδή αν set1 <= set2 και ταυτόχρονα set1 != set2
set1 >= set2    # Ελέγχει αν το set2 είναι υποσύνολο του set1,
# δηλαδή αν κάθε στοιχείο του set2 περιέχεται στο set1
set1 > set2    # Ελέγχει αν το set2 είναι γνήσιο υποσύνολο
# του set1, δηλαδή αν set1 >= set2 και ταυτόχρονα set1 != set2
print(set1 == set2)
False
print(set1 <= set3)
True
print(set1 > set4)
True
```

Συναρτήσεις

Υπάρχουν όπως έχουμε ήδη αναφέρει μια σειρά από συναρτήσεις που εφαρμόζονται στις ακολουθίες, όμως εφαρμόζονται επίσης και σε άλλες συλλογές. Αυτές περιλαμβάνουν την εύρεση του μεγέθους του συνόλου με τη συνάρτηση len( ) και, εφόσον οι τιμές είναι συγκρίσιμες, το μέγιστο και ελάχιστο στοιχείο με τις συναρτήσεις max( )και min( ) αντίστοιχα. Ακόμα η, επίσης

ενσωματωμένη, συνάρτηση sorted( ) επιστρέφει μια ταξινομημένη λίστα με τα ίδια στοιχεία με το σύνολό μας και τέλος η iter( ) που είδαμε νωρίτερα. Ας δούμε κι εδώ κάποια παραδείγματα:

```
set1 = {'pencil','pen','rubber','book' }
print(len(set1))    # Θα τυπώσει το μέγεθος (αριθμό στοιχείων)
# του συνόλου
4
print(max(set1))    # Θα τυπώσει τη «μεγαλύτερη» τιμή

rubber

print(min(set1))    # Θα τυπώσει τη «μικρότερη» τιμή

book
print(sorted(set1 ))    # Θα τυπώσει μια λίστα με τα στοιχεία
# του συνόλου ταξινομημένα.

['book', 'pen', 'pencil', 'rubber']
print(iter(set1 ))    # Θα μας επιστρέψει τη θέση του
# αντικειμένου που δημιούργησε η iter( ) στη μνήμη
# καθώς δεν προβλέπεται εκτύπωση για έναν iterator που
# κατασκευάστηκε έτσι. Π.χ.
<set_iterator object at 0x7f380f6d6640>
for x in iter(set1):    #
     print(x)    # Αντίθετα τυπώνει κανονικά τα επί μέρους
# στοιχεία του αντικειμένου (σε τυχαία εννοείται
# σειρά)
book rubber pen pencil
```

Μέθοδοι

Όπως έχουμε και αλλού αναφέρει τα σύνολα έχουν μια πληθώρα δημόσιων μεθόδων που κάποιες από αυτές αποτελούν εναλλακτικές υλοποιήσεις των πράξεων και συγκρίσεων μεταξύ συνόλων, κάποιες εξυπηρετούν τη διαχείριση των στοιχείων τους και φυσικά η ενσωματωμένη συνάρτηση `dir( )`, η οποία επιστρέφει μια λίστα με τα ονόματα των μεθόδων (δημόσιων και ιδιωτικών) του αντικειμένου μας.

Μέθοδοι υλοποίησης συγκρίσεων

`.isdisjoint( )` Δέχεται ως όρισμα ένα άλλο σύνολο και επιστρέφει True αν και μόνο αν το σύνολο αυτό δεν έχει κοινά στοιχεία με το αρχικό. Τα δύο σύνολα πρέπει να έχουν κενή τομή.

`.issubset( )` Δέχεται ως όρισμα ένα άλλο σύνολο και επιστρέφει True αν και μόνο αν κάθε στοιχείο στο αρχικό σύνολο ανήκει στο σύνολο όρισμα. Αποτελεί εναλλακτική υλοποίηση της σύγκρισης υποσύνολο (set1 <= set2).

`.issuperset( )` Δέχεται ως όρισμα ένα άλλο σύνολο και επιστρέφει True αν και μόνο αν κάθε στοιχείο στο σύνολο όρισμα ανήκει στο αρχικό σύνολο. Αποτελεί εναλλακτική υλοποίηση της σύγκρισης υπερσύνολο (set1 >= set2).

Μέθοδοι υλοποίησης πράξεων

`.union( )` Δέχεται ως όρισμα ένα ή περισσότερα σύνολα χωρισμένα με κόμμα (,) και επιστρέφει ένα νέο σύνολο που περιλαμβάνει όλα τα στοιχεία που ανήκουν είτε στο αρχικό σύνολο είτε σε οποιοδήποτε από τα σύνολα ορίσματα. Αποτελεί εναλλακτική υλοποίηση της ένωσης συνόλων ή αν θέλετε του λογικού ΄Η (set1 | set2 | set3 ...).

`.intersection( )` Δέχεται ως όρισμα ένα ή περισσότερα σύνολα χωρισμένα με κόμμα (,) και επιστρέφει ένα νέο σύνολο που περιλαμβάνει όλα τα στοιχεία που ανήκουν τόσο στο αρχικό σύνολο όσο και σε οποιοδήποτε από τα σύνολα ορίσματα. Αποτελεί εναλλακτική υλοποίηση της τομής συνόλων ή αλλιώς του λογικού ΚΑΙ (set1 & set2 & set3 ...).

`.difference( )` Δέχεται ως όρισμα ένα ή περισσότερα σύνολα χωρισμένα με κόμμα (,) και επιστρέφει ένα νέο σύνολο που περιλαμβάνει όλα τα στοιχεία που ανήκουν στο αρχικό σύνολο αλλά όχι σε κάποιο από τα σύνολα ορίσματα. Αποτελεί εναλλακτική υλοποίηση της set1 – set2 – set3 ...

`.symmetric_difference( )` Δέχεται ως όρισμα ένα σύνολο και επιστρέφει ένα νέο σύνολο που περιλαμβάνει όλα τα στοιχεία που ανήκουν είτε στο αρχικό σύνολο είτε στο σύνολο όρισμα αλλά όχι κα στα δύο. Αποτελεί εναλλακτική υλοποίηση της set1 ^ set2.

`.update( )` Δέχεται ως όρισμα ένα ή περισσότερα σύνολα χωρισμένα με κόμμα (,) και ενημερώνει το αρχικό σύνολο ώστε να περιλαμβάνει όλα τα στοιχεία που ανήκουν είτε στο αρχικό σύνολο είτε σε οποιοδήποτε από τα σύνολα ορίσματα. Αποτελεί εναλλακτική υλοποίηση της ένωσης συνόλων ή αν θέλετε του λογικού ΄Η που όμως αποθηκεύει το αποτέλεσμα στο αρχικό σύνολο(set1 |= set2 |= set3 ...).

`.intersection_update( )` Δέχεται ως όρισμα ένα ή περισσότερα σύνολα χωρισμένα με κόμμα (,) και ενημερώνει το αρχικό σύνολο ώστε να περιλαμβάνει όλα τα στοιχεία που ανήκουν τόσο στο αρχικό σύνολο όσο και σε οποιοδήποτε από τα σύνολα ορίσματα. Αποτελεί εναλλακτική υλοποίηση

της τομής συνόλων ή αλλιώς του λογικού ΚΑΙ που όμως αποθηκεύει το αποτέλεσμα στο αρχικό σύνολο (set1 &= set2 &= set3 ...).

`.difference_update( )` Δέχεται ως όρισμα ένα ή περισσότερα σύνολα χωρισμένα με κόμμα (,) και ενημερώνει το αρχικό σύνολο αφαιρώντας όλα τα στοιχεία που ανήκουν και σε κάποιο από τα σύνολα ορίσματα. Αποτελεί εναλλακτική υλοποίηση της set1 –= set2 | set3 ...

`.symmetric_difference_update( )` Δέχεται ως όρισμα ένα σύνολο και ενημερώνει το αρχικό σύνολο ώστε να περιλαμβάνει όλα τα στοιχεία που ανήκουν σε οποιοδήποτε από τα δύο σύνολα αλλά όχι και στα δύο. Αποτελεί εναλλακτική υλοποίηση της set1 ^= set2.

Διαχείριση στοιχείων

`.add( )` Δέχεται ως όρισμα ένα στοιχείο και το προσθέτει στο σύνολο.

`.remove( )` Δέχεται ως όρισμα ένα στοιχείο και το απομακρύνει από το σύνολο. Εάν το στοιχείο δεν περιλαμβάνεται στο σύνολο επιστρέφει KeyError.

`.discard( )` Δέχεται ως όρισμα ένα στοιχείο και το απομακρύνει από το σύνολο αν ανήκει σε αυτό.

`.pop( )` Απομακρύνει ένα τυχαίο στοιχείο από το σύνολο. Εάν το σύνολο είναι κενό επιστρέφει KeyError.

`.clear( )` Απομακρύνει όλα τα στοιχεία από το σύνολο.

`.copy( )` Επιστρέφει ένα «ρηχό» (by value) αντίγραφο του συνόλου.

.

7.5.3 Να θυμάμαι

Σύνολα

Το σύνολο είναι μια δομή δεδομένων με τα εξής χαρακτηριστικά:

- ✓ είναι σύνθετη, δηλαδή χρησιμοποιείται για την αποθήκευση πολλαπλών δεδομένων ταυτόχρονα
- ✓ είναι αδιάτακτη, δηλαδή η σειρά των μελών της είναι αδιάφορη
- ✓ είναι τροποποιήσιμη (mutable), με την έννοια ότι μπορούμε να προσθέσουμε και να αφαιρέσουμε στοιχεία όποτε θέλουμε
- ✓ τα στοιχεία/ μέλη της, από την άλλη, είναι μη τροποποιήσιμα (immutable)
- ✓ δεν επιτρέπεται η επανάληψη στοιχείου
- ✓ τα στοιχεία της πρέπει να είναι συγκρίσιμα
- ✓ τα στοιχεία/ μέλη της μπορούν να είναι ετερογενή

Τρόποι δημιουργίας συνόλων

Με απαρίθμηση των στοιχείων του:

```
set1 = {1, 2, 3, 4, 5, 6}
```

Με εφαρμογή της συνάρτησης σε κατάλληλη ακολουθία (όχι λεξικό στο σύνολό του):

```
set2 = set([10, 20, 30])
```

Με αρχική δημιουργία υποσυνόλου και διαδοχικές προσθήκες των μελών/ στοιχείων του (π.χ. με τη μέθοδο `.add( )`:

```
set3 = {5}
set3.add(10)
set3.add(15)
```

Προσοχή! Δεν μπορεί να είναι το κενό γιατί θα το θεωρήσει λεξικό

Πράξεις ανάμεσα σε σύνολα

Στα σύνολα εφαρμόζονται οι κλασσικές συνολοθεωρητικές πράξεις με τελεστές όπως `|` της ένωσης, `-` της διαφοράς, `^` της συμμετρικής διαφοράς και `&` της τομής, δεν εφαρμόζονται σε αυτά η συνένωση (concatenation) και ο πολλαπλασιασμός, για προφανείς λόγους.

Έλεγχος συμπερίληψης

Μπορούμε να ελέγξουμε αν ένα στοιχείο ανήκει σε ένα σύνολο ή και το αντίθετο. Αυτό γίνεται με την εφαρμογή των τελεστών `in` και `not in` για έλεγχο συμπερίληψης ή μη αντίστοιχα.

Συγκρίσεις

Και οι συγκριτικοί τελεστές μπορούν να εφαρμοστούν στα σύνολα, αν και έχουν διαφορετική σημασία:

```
set1 == set2    # Ελέγχει αν τα δύο σύνολα έχουν ακριβώς τα  ίδια στοιχεία
set1 != set2    # Ελέγχει αν έχουν τουλάχιστον ένα στοιχείο διαφορετικό.
set1 <= set2    # Ελέγχει αν το set1 είναι υποσύνολο του set2.
set1 < set2    # Ελέγχει αν το set1 είναι γνήσιο υποσύνολο του set2
set1 >= set2    # Ελέγχει αν το set2 είναι υποσύνολο του set1.
set1 > set2    # Ελέγχει αν το set2 είναι γνήσιο υποσύνολο του set1.
```

Συναρτήσεις

Κάποιες συναρτήσεις που αφορούν στις ακολουθίες εφαρμόζονται με διασταλτικό τρόπο και σε άλλες συλλογές όπως τα σύνολα. Εξυπηρετείται έτσι η εύρεση του μεγέθους του συνόλου (το πλήθος των στοιχείων του) από τη συνάρτηση `len( )` και, εφόσον οι τιμές είναι συγκρίσιμες, το μέγιστο και ελάχιστο στοιχείο με τις συναρτήσεις `max( )`και `min( )` αντίστοιχα. Ακόμα η, επίσης ενσωματωμένη, συνάρτηση `sorted( )` επιστρέφει μια ταξινομημένη λίστα με τα ίδια στοιχεία με το σύνολό μας και τέλος η `iter( )` επιτρέπει τη δημιουργία ενός iterator τόσο για τη διάσχιση του συνόλου μας όσο και για άλλες χρήσεις.

Μέθοδοι

Ένα σύνολο έχει τις περισσότερες δημόσιες μεθόδους από όλους τους ενσωματωμένους σύνθετους τύπους. Και αυτό γιατί περιλαμβάνει πολλαπλές κατηγορίες όπως:

Μεθόδους υλοποίησης συγκρίσεων,

Μεθόδους υλοποίησης πράξεων,

Μεθόδους διαχείριση στοιχείων.

7.5.4 Ερωτήσεις Κατανόησης

1. Το σύνολο στην Python είναι μια σύνθετη δομή δεδομένων.

   ΝΑΙ ΟΧΙ

2. Το σύνολο στην Python είναι διατεταγμένο.

   ΝΑΙ ΟΧΙ

3. Υπάρχει μόνο ένας τρόπος εκτέλεσης συνολοθεωρητικών πράξεων στην Python.

   ΝΑΙ ΟΧΙ

4. Σε σύνολα αποθηκεύουμε υποχρεωτικά ομοειδή στοιχεία.

   ΝΑΙ ΟΧΙ

5. Η γλώσσα Python διαθέτει συναρτήσεις που εφαρμόζονται εκτός από τις ακολουθίες και σε άλλες συλλογές, όπως οι . `len( )`, `max( )` και `min( )`

   ΝΑΙ ΟΧΙ

6. Οι συγκρίσεις δεν εφαρμόζονται στα σύνολα.

   ΝΑΙ ΟΧΙ

7.5.5 Ερωτήσεις Ανάπτυξης

1. Ποια είναι τα χαρακτηριστικά του συνόλου στην Python;
2. Με ποιους τρόπους κατασκευάζουμε σύνολα στην Python;
3. Μπορούν οι συνολοθεωρητικές πράξεις να χρησιμοποιηθούν για το σκοπό αυτό;
4. Με ποιο τρόπο μπορούμε να δούμε όλα τα στοιχεία ενός συνόλου;
5. Προτείνετε τέσσερις τρόπους για να προσθέσουμε στοιχεία σε ένα σύνολο.
6. Προτείνετε τρεις τρόπους για να απομακρύνουμε ένα στοιχείο από ένα σύνολο

7.5.6 Ασκήσεις

1. Δεδομένου ότι κάθε φορά που χρησιμοποιώ ένα σύνολο η σειρά των αριθμών είναι τυχαία, αν έχω ένα σύνολο set = {1,2,3,4,5,6} κάθε εκτύπωσή π.χ. θα μας δίνει πρώτο έναν διαφορετικό αριθμό. Άρα ο επόμενος κώδικας θεωρητικά προσομοιώνει τη ρήψη ενός ζαριού:

```
set = {1,2,3,4,5,6}

zari = list(set)

print(zari[0])
```

Αν αντί για print σώσουμε την κάθε τιμή σε μια θέση μιας νέας λίστας num μπορούμε να φτιάξουμε μια επανάληψη π.χ. 1000 ή 10000 επαναλήψεων και τυπώνοντας, όταν τελειώσει, τη συχνότητα εμφάνισης κάθε αριθμού στο set στη νέα μας λίστα μπορούμε να ελέγξουμε πόσο καλά προσομοιώνει την τυχαιότητα. Για τη συχνότητα μπορούμε να διαιρέσουμε το .count( ) του στοιχείου με τον αριθμό των επαναλήψεων.

Υλοποιείστε και ελέγξτε.

2. Αν θελήσουμε να προσομοιώσουμε με παρόμοιο τρόπο τη λειτουργία του ΛΟΤΤΟ πρέπει να αντιμετωπίσουμε το πρόβλημα ότι τα 6 νούμερα που θα επιλέξει το πρόγραμμά μας πρέπει να είναι διαφορετικά. Σκέψου έναν τρόπο να το παρακάμψουμε και φτιάξε ένα πρόγραμμα που όσο του ζητάμε θα μας επιστρέφει εξάδες.

Κεφάλαιο 8

8. Λάθη, Παγίδευση Λαθών, Αρχεία, Διαχείριση Αρχείων

Τελειώνοντας αυτό το μάθημα θα έχεις μάθει

- Να αναγνωρίζεις τα είδη σφαλμάτων που παρουσιάζονται στον κώδικα
- Να προφυλάσσεις τον κώδικα από λάθη με τις εντολές παγίδευσης
- Τις διαφορές φύλαξης δεδομένων στην κύρια και στη δευτερεύουσα μνήμη
- Τους τύπους κωδικοποίησης των αρχείων κειμένου
- Να εκτελείς τις βασικές λειτουργίες με εντολές γλώσσας σε αρχεία κειμένου
- Να λύνεις απλά προβλήματα με δεδομένα που αποθηκεύονται σε αρχεία
- Να αλλάζεις την κωδικοποίηση των περιεχομένων σε αρχείο κειμένου
- Να χρησιμοποιείς τις βασικές μεθόδους του αρθρώματος λογισμικού os
- Τη χρήση διάφορων προγραμματιστικών τεχνικών για τις εργασίες στα αρχεία
- Να συγκρίνεις προγραμματιστικές προσεγγίσεις επίλυσης προβλημάτων

8.1 Λάθη στον κώδικα

Τα προγράμματα που υλοποιούνται από έναν προγραμματιστή ειδικά στις πρώτες εκδόσεις τους σπάνια λειτουργούν απολύτως σωστά. Μεγάλοι τίτλοι λογισμικού περνούν από πολλές εκδόσεις (versions), διορθώσεις (patches), ενημερώσεις (updates), προκειμένου να αποκτήσουν σχετικά αξιόπιστη λειτουργία. Πλήθος λαθών (συντακτικών, χρόνου εκτέλεσης ή λογικών) διορθώνονται στις διαδικασίες ελέγχου και δοκιμών αλλά και σε όλη τη διάρκεια ζωής ενός λογισμικού. Η διαδικασία αυτή, δηλαδή του ελέγχου, εντοπισμού και διόρθωσης των λαθών σε ένα πρόγραμμα ονομάζεται εκσφαλμάτωση ή αποσφαλμάτωση (debugging).

Τα λάθη ή σφάλματα (errors) όπως αναφέρθηκε και στο κεφάλαιο 2 μπορεί να είναι:

Τα συντακτικά λάθη (syntax errors), τα οποία εμφανίζονται όταν παραβιάζονται οι κανόνες σύνταξης των εντολών της γλώσσας. Είναι τα πιο συχνά σφάλματα αλλά και τα πιο εύκολα να διορθωθούν αφού εντοπίζονται από το μεταγλωττιστή ή το διερμηνέα ο οποίος εμφανίζει προειδοποιητικά μηνύματα προς τον προγραμματιστή. Το πρόγραμμα δεν εκτελείται εάν δε διορθωθούν όλα τα συντακτικά του λάθη. Παραδείγματα τέτοιων σφαλμάτων προκύπτουν: σε λανθασμένη σύνταξη εντολών, στη στοίχιση του κώδικα, σε λανθασμένη χρήση παρενθέσεων στις εκφράσεις υπολογισμού, ορθογραφικά (στις δεσμευμένες λέξεις) κ.ά.

Τα λάθη χρόνου εκτέλεσης (run time errors) εμφανίζονται κατά την εκτέλεση του προγράμματος και προκαλούν τη βίαιη διακοπή του (crash). Αυτά δεν εντοπίζονται από το περιβάλλον της γλώσσας που χρησιμοποιούμε (διερμηνέα ή μεταγλωττιστή) και είναι πιο δύσκολο να αντιμετωπιστούν από ότι τα συντακτικά λάθη διότι πολλές φορές οφείλονται σε καταστάσεις που δεν είναι εύκολο να προβλεφθούν. Αυτά τα σφάλματα μπορεί να συμβούν: κατά την εισαγωγή ασύμβατων δεδομένων από το χρήστη, σε μη έγκυρες μαθηματικές πράξεις (διαίρεση με το 0), αστοχίες στο υλικό (σφάλματα στην επιφάνεια ανάγνωσης ενός δίσκου) κ.ά. Οι γλώσσες προγραμματισμού διαθέτουν εντολές διαχείρισης που «παγιδεύουν τα λάθη αυτά» αποφεύγοντας έτσι και την ξαφνική (και εκνευριστική) διακοπή του προγράμματος. Τέτοιες εντολές θα δούμε στη συνέχεια του κεφαλαίου.

Τα λογικά λάθη (logical errors) είναι λάθη σχεδιασμού στο πρόγραμμα, δεν προκαλούν αντίδραση από το περιβάλλον της γλώσσας ή κάποια ξαφνική διακοπή του προγράμματος, αλλά εμφανίζουν λάθος αποτελέσματα. Κάποιες φορές είναι εύκολο να τα αντιληφθούμε (πχ αναμένουμε μία θετική τιμή και λαμβάνουμε ως αποτέλεσμα μία αρνητική τιμή), άλλες φορές όμως είναι πολύ δύσκολο ή και αδύνατο να τα αντιληφθούμε απλά βλέποντας ένα αποτέλεσμα (πχ ένα λάθος σε μία υπολογιστική διαδικασία εντοπισμού ενός μαθηματικού ορίου, υπολογισμός ενός μεγάλου αθροίσματος). Γι' αυτά υπάρχουν εργαλεία από το προγραμματιστικό περιβάλλον που βοηθούν τον προγραμματιστή στην ανεύρεσή τους και ονομάζονται debuggers καθώς και διάφορες τεχνικές ελέγχου ορθότητας των αποτελεσμάτων.

Η διαδικασία ελέγχου του προγράμματος μπορεί να περιλαμβάνει:

- Πολλές δοκιμαστικές εκτελέσεις με κατάλληλα επιλεγμένα **έγκυρα δεδομένα**, όπου συγκρίνονται τα αναμενόμενα με τα παραγόμενα αποτελέσματα του προγράμματος. Εάν αυτά πχ δεν είναι ίδια η ύπαρξη λογικού λάθους είναι βέβαιη. Εάν είναι ίδια είναι μία καλή ένδειξη αλλά δυστυχώς ΟΧΙ απόδειξη ΜΗ ύπαρξης λογικού λάθους γιατί η απουσία του

μπορεί να οφείλεται στα μη αντιπροσωπευτικά δεδομένα της δοκιμής μας και όχι στον αλάνθαστο κώδικα.

- Εκτελέσεις με δεδομένα εισόδου που **δεν ανταποκρίνονται στις προδιαγραφές των απαιτήσεων** του προβλήματος και εξαρτώνται από τους προσεκτικούς ή μη, χειρισμούς του χρήστη ή από κάποιο αισθητήρα που τροφοδοτεί με δεδομένα το πρόγραμμα και λόγω αστοχίας υλικού στέλνει τιμές εκτός προβλεπόμενων ορίων. Οι δοκιμές αυτές ελέγχουν τον αμυντικό σχεδιασμό του κώδικα δηλαδή τη συμπεριφορά του σε μη έγκυρες τιμές δεδομένων εισόδου.

Σε προγράμματα ζωτικής σημασίας η διαδικασία της εκσφαλμάτωσης είναι μείζονος σημασίας, διότι τα λάθη που μπορεί να διαφύγουν από αυτήν μπορούν να προκαλέσουν μεγάλες οικονομικές απώλειες ή δυστυχώς και απώλειες σε ανθρώπινες ζωές. Σε τέτοια προγράμματα η φάση δοκιμών και ελέγχου της ορθής και απρόσκοπτης λειτουργίας τους μπορεί να ξεπερνάει σε χρονική διάρκεια τη φάση σχεδίασης και ανάπτυξής τους.

Η ανάλυση ενός προβλήματος, ο προσεκτικός σχεδιασμός της λύσης, η διάσπασή του προγράμματος σε μικρότερα μέρη και η εκσφαλμάτωση αυτών σταδιακά, ο αμυντικός σχεδιασμός, η τεκμηρίωση του κώδικα, επιδρούν θετικά στον περιορισμό, στον εντοπισμό και τη διόρθωση τελικά των σφαλμάτων στον κώδικα του προγράμματος.

8.2 Χειρισμός λαθών στην Python

8.2.1 Μηνύματα λαθών στην Python

Όταν συμβαίνει κάποιο λάθος στον κώδικα και δεν είναι δυνατή η εκτέλεσή του, η γλώσσα Python προκαλεί/ υψώνει (raise) μία εξαίρεση (exception) ως ένδειξη ότι παραβιάζεται κάποιος κανόνας σύνταξης ή μαθηματικού περιορισμού ή περιορισμού της λύσης του προβλήματος. Ακολουθούν κάποια παραδείγματα τέτοιων εξαιρέσεων.

8.2.2 Συντακτικά Λάθη (Syntax Errors)

Εάν στο κέλυφος δοθούν οι παρακάτω εντολές `print` ο διερμηνέας της γλώσσας Python στην προσπάθεια να εκτελέσει τις εντολές αυτές θα διαπιστώσει και θα εντοπίσει την πρώτη κατηγορία λαθών που αναφέρθηκαν στην προηγούμενη παράγραφο δηλαδή τα συντακτικά λάθη που υπάρχουν. Θα χρωματιστούν τα σημεία που παραβιάζονται οι κανόνες σύνταξης προκαλώντας και αντίστοιχα μηνύματα λάθους (raise syntax errors) με κόκκινο χρώμα προτρέποντας τον προγραμματιστή στη διόρθωσή τους ή ακόμα και με πιο συγκεκριμένα μηνύματα (ανάλογα το είδος του συντακτικού λάθους). Οι εντολές και συνεπώς και το πρόγραμμα δεν μπορεί να εκτελεστεί εάν δεν διορθωθούν ΟΛΑ τα συντακτικά λάθη.

Παράδειγμα 8.1

Ακολουθούν παραδείγματα συντακτικών λαθών

```
# Συντακτικό Λάθος 1
>>> print(3 + 4) / 5)
SyntaxError: unmatched ')'

# Συντακτικό Λάθος 2
>>> print("hello')
SyntaxError: incomplete input
```

Παρατηρήσεις

- Για την απούσα παρένθεση εμφανίζει : `unmatched ')'`
- Για τους διαφορετικούς τόνους στο μήνυμα εμφανίζει: `incomplete input`

8.2.3 Λάθη χρόνου εκτέλεσης (Run Time Errors)

Παράδειγμα 8.2

Εάν στο κέλυφος αυτή τη φορά δοθούν οι παρακάτω εντολές/ συναρτήσεις `sqrt`, `divmod`, `div` ο διερμηνέας αφού ελέγξει τη σύνταξη των εντολών που δόθηκαν και τις βρει συντακτικά ορθές θα τις εκτελέσει. Όμως και οι τρείς εντολές αν και συντακτικά ορθές, παραβιάζουν κανόνες υπολογισμού (τετραγωνική ρίζα αρνητικού αριθμού, διαίρεση με το μηδέν, διαίρεση συμβολοσειράς. Έτσι προκαλούνται αντίστοιχες εξαιρέσεις που αναφέρονται σε λάθη χρόνου εκτέλεσης δίνοντας όμως ακριβώς **και** το είδος του λάθους που προκύπτει:

```
# Λάθος Χρόνου Εκτέλεσης 1
>>> from math import sqrt
```

```
>>> sqrt(-25)

Traceback (most recent call last):
  File "<pyshell#157>", line 1, in <module>
    sqrt(-25)
ValueError: math domain error

# Λάθος Χρόνου Εκτέλεσης 2
>>> divmod(8, 0)

Traceback (most recent call last):
  File "<pyshell#158>", line 1, in <module>
    divmod(8, 0)
ZeroDivisionError: integer division or modulo by zero

# Λάθος Χρόνου Εκτέλεσης 3
def div(a, b):
    return a / b

div(3, 5)
0.6
div('cut in half', 2)
Traceback (most recent call last):
  File "<pyshell#207>", line 1, in <module>
    div('cut in half', 2)
  File "<pyshell#205>", line 2, in div
    return a / b
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

- Για την τετραγωνική Ρίζα Αρνητικού αριθμού: `ValueError: math domain error`
- Για τη διαίρεση με το μηδέν (0) :`ZeroDivisionError: integer division or modulo by zero`
- Για τη διαίρεση με τη συμβολοσειρά:`TypeError: unsupported operand type(s) for /: 'str' and 'int'`

8.2.4 Λογικά Λάθη

Τα λογικά λάθη όπως αναφέρθηκε είναι λάθη σχεδιασμού, είναι δύσκολο να εντοπιστούν και προκαλούν λανθασμένα αποτελέσματα.

Παράδειγμα 8.3

Ζητείται ο υπολογισμός του μέσου όρου τριών (3) βαθμολογιών. Η παρακάτω λύση περιέχει λογικό λάθος.

```
a, b, c = map(int,input('Δώστε 3 βαθμολογίες α, β, γ:').split(','))
```

```
mo = a + b + c / 3
print('Μέσος όρος των βαθμών',a , b, c, 'είναι',mo)
```

Δοκιμάστε να εκτελέσετε τον κώδικα με κάποιες τιμές παρατηρήστε τα αποτελέσματα. Μπορείτε να εντοπίσετε και να διορθώσετε το λογικό λάθος;

Σημείωση

Η συνάρτηση map η οποία αναφέρεται στο παραπάνω παράδειγμα ανήκει σε μία κατηγορία συναρτήσεων της Python που ονομάζονται high-order functions και οποίες δέχονται ως ορίσματα άλλες συναρτήσεις παρόμοιες συναρτήσεις είναι η filter και η reduce.

Η συνάρτηση map που χρησιμοποιείται εδώ περνάει τη συνάρτηση του 1ου ορίσματος (mapping function στο παράδειγμα η συνάρτηση int) σε κάθε ένα από τα στοιχεία της διασχίσιμης ακολουθίας του 2ου ορίσματος εδώ η συμβολοσειρά που επιστρέφεται από τη συνάρτηση input)

Παράδειγμα 8.4

Ζητείται η αριθμητική έκφραση σε γλώσσα Python για τον υπολογισμό της ρίζας που προκύπτει από την παράσταση $\frac{-\beta+\sqrt{\Delta}}{2\alpha}$. *Η παρακάτω παράσταση έχει λογικό λάθος.*

```
root = (-b + d ** 0.5) / 2 * a
print(root)
```

Δοκιμάστε να εκτελέσετε τον κώδικα με κάποιες τιμές παρατηρήστε τα αποτελέσματα. Μπορείτε να εντοπίσετε και να διορθώσετε το λογικό λάθος;

8.2.5 Δραστηριότητα

Μπορείτε να εντοπίσετε και να διορθώσετε το λογικό λάθος στο παράδειγμα 6.24;

8.2.6 Εντολές παγίδευσης/διαχείρισης λαθών και αμυντικός προγραμματισμός

Αμυντικός προγραμματισμός (Defensive programming) καλείται η προγραμματιστική τεχνική κατά την οποία ο προγραμματιστής ενσωματώνει στο πρόγραμμα εντολές ή και ολόκληρα τμήματα κώδικα μέσω των οποίων προσπαθεί να διασφαλίσει ότι ΜΗ έγκυρες τιμές εισόδου δεδομένων από το χρήστη δε θα προκαλέσουν απροσδόκητες καταστάσεις όπως σφάλματα χρόνου εκτέλεσης ή λογικά σφάλματα.

Για τα συντακτικά λάθη, ο διερμηνέας αναλαμβάνει να δώσει οδηγίες διόρθωσης στον προγραμματιστή (προκειμένου πρώτα να προβεί σε διορθώσεις και έπειτα να εκτελέσει τον κώδικά του).

Για τα λάθη χρόνου εκτέλεσης, τα λογικά λάθη και τη βελτίωση του **αμυντικού σχεδιασμού** του προγράμματος η γλώσσα Python διαθέτει τις εντολές παγίδευσης και διαχείρισης λαθών `try ... except :`

Σύνταξη της εντολής:

```
try:
    <Εντολές κώδικα προς εκτέλεση>
```

```
except:
    <Εντολές Διαχείρισης Λάθους>
```

Όταν κάποια από τις εντολές που περιέχονται στην εντολή `try` υψώσει(προκαλέσει) λάθος (Raise Exception Error) ο έλεγχος του προγράμματος μεταφέρεται στις εντολές διαχείρισης λάθους της εντολής `except`, εάν δεν υπάρχει λάθος εκτελούνται όλες οι εντολές της ενότητας `try` εκτελούνται κανονικά.

Παράδειγμα 8.5

Συντάξετε και καλέστε την παρακάτω συνάρτηση:

```
>>> def division(a, b):
        try:
            z = a / b
            print('Η διαίρεση του',a ,'με το',b,'δίνει', z)
        except:
            print('Προσοχή διαίρεση με το μηδέν (0)')

>>> division(3.123,5)
Η διαίρεση του 3.123 με το 5 δίνει 0.6246

>>> division(3.123,0)
Προσοχή διαίρεση με το μηδέν (0)
```

Πλήρης ανάπτυξη των εντολών παγίδευσης σφαλμάτων

Οι εντολές παγίδευσης μπορούν να έχουν περισσότερες από μία εντολές `except` ελέγχοντας ταυτόχρονα και το είδος λάθους (εξαίρεσης), όπως επίσης και την (προαιρετική) εντολή `else` όπου η ενότητα των εντολών που περιέχει εκτελείται όταν δεν υπάρχει `exception error` και την (προαιρετική) εντολή `finally` με την ενότητα των εντολών που περιέχει να εκτελείται σε κάθε περίπτωση.

```
try:
    <Εντολές κώδικα προς εκτέλεση>
except <Είδος Λάθους>:
    <Εντολές Διαχείρισης Λάθους>
except <Είδος Λάθους>:
    <Εντολές Διαχείρισης Λάθους>
else:
    <Εντολές Εάν δεν υπάρχει λάθος>
finally:
    <Εντολές που εκτελούνται σε κάθε περίπτωση>
```

Παράδειγμα 8.6

Τροποποιήσετε την συνάρτηση του προηγούμενου παραδείγματος ως εξής:

```
>>>def division(a, b):
        try:
            z = a / b
        except ZeroDivisionError:
            print('Προσοχή διαίρεση με το μηδέν (0)')
        except TypeError:
            print('Δεν γίνεται διαίρεση με αυτόν τον τύπο δεδομένων')
        else:
            print('Η διαίρεση του',a ,'με το',b,'δίνει', z)
        finally:
            print('Ευχαριστούμε !!')

>>> division(3, 5)
Η διαίρεση του 3 με το 5 δίνει 0.6
Ευχαριστούμε !!

>>> division(3, 0)
Προσοχή διαίρεση με το μηδέν (0)
Ευχαριστούμε !!

>>> division(3, 'Hello World')
Δεν γίνεται διαίρεση με αυτόν τον τύπο δεδομένων
Ευχαριστούμε !!
```

Παράδειγμα 8.7

Ακολουθεί ένα παράδειγμα απλού αμυντικού προγραμματισμού.

Ας υποθέσουμε ότι ζητείται πρόγραμμα το οποίο να δέχεται από το χρήστη έναν ακέραιο αριθμό και στη συνέχεια να κάνει κάποια λειτουργία με αυτόν.

Πολλά από τα σφάλματα χρόνου εκτέλεσης συμβαίνουν όπως αναφέρθηκε σε περιπτώσεις εισαγωγής ασύμβατων δεδομένων από το χρήστη (πχ ζητείται αριθμός και ο χρήστης εισάγει χαρακτήρες). Όπως συμβαίνει στην παρακάτω εντολή:

α) Ζητείται ακέραιος αριθμός και ο χρήστης εισάγει πραγματικό

```
>>> n = int(input('Δώσε έναν ακέραιο αριθμό: '))
Δώσε έναν ακέραιο αριθμό: 34.5        # α) Δίνεται αριθμός τύπου float
```

εμφανίζεται μήνυμα λάθους χρόνου εκτέλεσης:

```
Traceback (most recent call last):
  File "<pyshell#65>", line 1, in <module>
    n = int(input('Δώσε έναν ακέραιο αριθμό: '))
```

```
ValueError: invalid literal for int() with base 10: '34.5'
```

β) Ζητείται ακέραιος αριθμός και ο χρήστης εισάγει 12,

```
>>> n = int(input('Δώσε έναν ακέραιο αριθμό: '))
Δώσε έναν ακέραιο αριθμό: 12,          # β) Δίνεται & κόμμα ,
```

εμφανίζεται μήνυμα λάθους χρόνου εκτέλεσης:

```
Traceback (most recent call last):
  File "<pyshell#68>", line 1, in <module>
    n = int(input('Δώσε έναν ακέραιο αριθμό: '))
ValueError: invalid literal for int() with base 10: '12,'
```

β) Ζητείται ακέραιος αριθμός και ο χρήστης εισάγει enter,

```
>>> n = int(input('Δώσε έναν ακέραιο αριθμό: '))
Δώσε έναν ακέραιο αριθμό:              # γ) Δίνεται εκ λάθους το enter
```

εμφανίζεται μήνυμα λάθους χρόνου εκτέλεσης:

```
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    n = int(input('Δώσε έναν ακέραιο αριθμό: '))
ValueError: invalid literal for int() with base 10: ''
```

Παρατήρηση

Όπου δε δοθούν ακέραιοι αριθμοί θα προκληθεί exception error αφού τα δεδομένα που δίνονται από τον χρήστη η συνάρτηση `int()` *αδυνατεί να τα μετατρέψει σε ακέραιο αριθμό και ως συνέπεια προκαλείται κατάρρευση του προγράμματος (crash). Το μήνυμα λάθους είναι το ίδιο σε όλες τις περιπτώσεις* `ValueError`.

Παράδειγμα 8.8

Υλοποιήστε κώδικα ο οποίος θα εξασφαλίζει ότι στο προηγούμενο παράδειγμα δεν θα προκληθεί λάθος χρόνου εκτέλεσης από μη έγκυρη είσοδο δεδομένων.

Για να εξασφαλιστεί η εγκυρότητα των δεδομένων σε αυτή την περίπτωση ακολουθούν δύο ενδεικτικές λύσεις:

Λύση Α – με χρήση της μεθόδου συμβολοσειρών `isdigit()`

```
>>> while True:
        x = input('Δώσε έναν ακέραιο αριθμό: ')
        if x.isdigit():
            x = int(x)
            break
        else:
            print(x, 'δεν είναι ακέραιος αριθμός')
```

```
Δώσε έναν ακέραιο αριθμό: 12.34
12.34 δεν είναι ακέραιος αριθμός
Δώσε έναν ακέραιο αριθμό: 11,
11, δεν είναι ακέραιος αριθμός
Δώσε έναν ακέραιο αριθμό: adb
adb δεν είναι ακέραιος αριθμός
Δώσε έναν ακέραιο αριθμό:
 δεν είναι ακέραιος αριθμός
Δώσε έναν ακέραιο αριθμό: 12
```

Λύση Β - με χειρισμό εξαιρέσεων `try .. except`

```
while True:
    try:
        x = input('Δώσε έναν ακέραιο αριθμό: ')
        x = int(x)
        break
    except ValueError:
        print(x, 'δεν είναι ακέραιος αριθμός')

Δώσε έναν ακέραιο αριθμό: 12.34
12.34 δεν είναι ακέραιος αριθμός
Δώσε έναν ακέραιο αριθμό: 11,
11, δεν είναι ακέραιος αριθμός
Δώσε έναν ακέραιο αριθμό: abc
abc δεν είναι ακέραιος αριθμός
Δώσε έναν ακέραιο αριθμό:
 δεν είναι ακέραιος αριθμός
Δώσε έναν ακέραιο αριθμό: 12
```

Σημείωση

Παρατηρείστε ότι οι παραπάνω λύσεις Α, Β έχουν παρόμοια αποτελέσματα όπως θα δείτε όμως στη συνέχεια, στη διαχείριση αρχείων όπου η παγίδευση λαθών είναι απαραίτητη προτείνεται ο τύπος της λύσης Β με χειρισμό μέσω εξαιρέσεων.

8.2.7 Δραστηριότητα, παράδειγμα σύνθετου αμυντικού προγραμματισμού

Ζητείται να υλοποιήσετε μία συνάρτηση η οποία θα διαβάζει και θα επιστρέφει μία λίστα βαθμολογιών μαθητών στην κλίμακα [1..20] με προσέγγιση δεκάτου πχ 12.3, 17.8, 8.0 κλπ. την οποία θα εισάγει με κατάλληλα μηνύματα ο χρήστης. Πρέπει να εξασφαλίσετε ότι δεν θα είναι δυνατή η επιστροφή λίστας με μη έγκυρες βαθμολογίες (πχ βαθμούς εκτός κλίμακας, ή με περισσότερα δεκαδικά ψηφία από 1, ή χαρακτήρες κλπ.). Η είσοδος βαθμολογιών θα σταματά όταν πατηθεί (enter) χωρίς να δοθεί καμία βαθμολογία.

Σημειώσεις:

- Σε περίπτωση έγκυρης βαθμολογίας θα εμφανίζεται μήνυμα «βαθμός αποδεκτός»
- Σε περίπτωση εισαγωγής μη αριθμητικών τιμών (εκτός του enter) θα εμφανίζεται μήνυμα «μη έγκυρος βαθμός»
- Σε περίπτωση αριθμητικών τιμών εκτός ορίων θα εμφανίζεται μήνυμα «βαθμός εκτός ορίων»
- Σε περίπτωση τιμών εντός ορίων αλλά με περισσότερα από ένα δεκαδικά ψηφία θα εμφανίζεται μήνυμα «ο βαθμός έχει πολλά δεκαδικά ψηφία»
- Σε περίπτωση που θα πατηθεί (enter) θα σταματάει η εισαγωγή βαθμών και θα επιστρέφεται η λίστα με τις έγκυρες βαθμολογίες

Προσπαθήστε να επιλύσετε τη δραστηριότητα χωρίς να δείτε τη ενδεικτική λύση που ακολουθεί.

Ενδεικτική λύση της δραστηριότητας

```
def grades():
    grades = []
    while True:
        try:
            grade = input('Δώσε βαθμό [1..20] με προσέγγιση δεκάτου: ')
            grade = float(grade)
        except:
            if grade == '':
                break
            else:
                print('Μη έγκυρος βαθμός')
        else:
            if 1 <= grade <= 20:
                if int(grade * 10) == grade * 10:
                    print('Βαθμός αποδεκτός')
                    grades.append(grade)
                else:
                    print('Ο βαθμός έχει πολλά δεκαδικά ψηφία')
            else:
                print('Βαθμός εκτός ορίων')
    return grades
```

8.3 Κύρια και δευτερεύουσα μνήμη

Μέχρι τώρα για τα δεδομένα εισόδου των εφαρμογών χρησιμοποιήθηκαν οι βασικοί τύποι δεδομένων της Python `int()`, `float()`, `str()`, `bool()` ή οι δομές δεδομένων (lists, tuples, dictionaries). Αυτά (τα δεδομένα) δίνονταν συνήθως από το χρήστη της εφαρμογής χρησιμοποιώντας κάποια εντολή εισόδου σε χρόνο εκτέλεσης (run time).

Επίσης τα αποτελέσματα των προγραμμάτων για τα οποία χρησιμοποιήθηκαν παρόμοιοι τύποι δεδομένων ή και δομές όπως και για τα δεδομένα εισόδου εμφανίζονταν στην οθόνη του υπολογιστή χρησιμοποιώντας την συνάρτηση εξόδου `print()`.

Τα παραπάνω δεδομένα είτε αφορούσαν σε είσοδο είτε σε έξοδο λειτουργιών της εφαρμογής διατηρούνταν προσωρινά και παρέμεναν αποθηκευμένα στην κύρια και ταχύτατη μνήμη (RAM) του υπολογιστή μας, μόνο για όσο διάστημα διαρκούσε η εκτέλεση του προγράμματος.

Το χαρακτηριστικό της κύριας μνήμης που αφορά στην προσωρινή διατήρηση των δεδομένων αλλά και αυτό του περιορισμένου μεγέθους της (λόγω του υψηλού κόστους της) την καθιστούν ακατάλληλη για χρήση από απαιτητικές εφαρμογές (εμπορικές ή επιστημονικές) με μεγάλο όγκο δεδομένων που έχουν ανάγκες μόνιμης αποθήκευσής τους.

Έτσι για τις ανάγκες αυτές (μεγάλου όγκου δεδομένων και μόνιμης αποθήκευσης δεδομένων) χρησιμοποιείται η δευτερεύουσα μνήμη η οποία κατά κύριο λόγο στις εφαρμογές μας αποτελείται από μονάδες σκληρών δίσκων (HDD, SSD) οι οποίες είναι μεν πιο αργές από την κύρια μνήμη αλλά καλύπτουν τις αυξημένες απαιτήσεις της χωρητικότητας και της μόνιμης αποθήκευσης.

Εκτός από τα αρχεία τα οποία περιέχουν τον κώδικα της εφαρμογής μας σε γλώσσα Python (`.py`) μία εφαρμογή μπορεί να συνοδεύεται και από άλλα αρχεία που περιέχουν δεδομένα εισόδου ή αποτελέσματα εξόδου. Τα αρχεία αυτά των δεδομένων στη δευτερεύουσα μνήμη αποθηκεύονται με τη μορφή **αρχείων κειμένου** (text files) ή **δυαδικών αρχείων** (binary files).

Τα αρχεία κειμένου είναι αρχεία που το περιεχόμενό τους αποτελείται από χαρακτήρες κειμένου όπως αυτούς των συμβολοσειρών και μπορεί να αναγνωστεί από τον χρήστη χρησιμοποιώντας και έναν απλό συντάκτη κειμένου. Αυτός είναι και ο πιο κοινός τύπος αρχείων και μπορούν να χρησιμοποιηθούν εύκολα για δεδομένα αντικειμένων των βασικών τύπων `int`, `float`, `str`, `bool` .

Τα δυαδικά αρχεία τα οποία περιέχουν δεδομένα σε δυαδική μορφή μη αναγνώσιμη από το χρήστη μπορούν να χρησιμοποιηθούν για κάθε τύπο αντικειμένων ακόμα και εικόνας, ήχου κλπ.

Σημείωση

Τα δυαδικά αρχεία δεν θα αποτελέσουν αντικείμενο του κεφαλαίου. Όποτε αναφέρεται από εδώ και πέρα η λέξη αρχείο θα εννοείται πως αφορά σε αρχείο κειμένου.

8.4 Αρχεία κειμένου

Τα αρχεία δεδομένων κειμένου που χρησιμοποιούνται στην Python όπως και τα υπόλοιπα αρχεία του ΛΣ αποθηκεύονται σε κάποιο φάκελο του συστήματος αρχείων μέσα στο σκληρό δίσκο. Για να είναι εφικτό το άνοιγμα ή η εγγραφή σε ένα αρχείο πρέπει να είναι γνωστά:

α. το όνομα του αρχείου καθώς και
β. η απόλυτη διαδρομή μέσα στο σύστημα αρχείων του ΛΣ

πχ

- σε ΛΣ windows : `c:\python311\my_files\data.txt`
- σε ΛΣ linux : `/home/user/my_files/data.txt`

Θυμίζουμε ότι τα αρχεία κώδικα της Python τα οποία δημιουργούνται από το συντάκτη στο περιβάλλον IDLE αλλά και στους άλλους συντάκτες είναι και αυτά αρχεία που περιέχουν κείμενο (που αποτελείται όμως από εντολές της γλώσσας Python).

- τα αρχεία κώδικα της Python έχουν επέκταση `.py`
- τα αρχεία δεδομένων κειμένου έχουν συνήθως επέκταση `.txt, .csv`

8.4.1 Διαχείριση αρχείων κειμένου

Για τη διαχείριση ενός αρχείου κειμένου της Python πρέπει πρώτα να δημιουργηθεί ένα αντίστοιχο αντικείμενο <TextIOWrapper> (θα καλείται χάρη απλότητας αντικείμενο διαχείρισης αρχείου κειμένου).

Για να δημιουργηθεί ένα τέτοιο αντικείμενο θα χρησιμοποιηθεί η συνάρτηση `open` η οποία συντάσσεται ως εξής:

```
<όνομα αναφοράς> = open(<διαδρομή και όνομα αρχείου>,
mode=<κατάσταση/mode>, encoding=<κωδικοποίηση>)
```

Στο πρώτο όρισμα της συνάρτησης open, τοποθετείται μία συμβολοσειρά στην οποία περιγράφεται το όνομα και η διαδρομή που βρίσκεται το αρχείο κειμένου, στο δεύτερο όρισμα τοποθετείται ένας χαρακτήρας ο οποίος δηλώνει για ποια λειτουργία δημιουργείται το αντικείμενο, και στο τρίτο όρισμα ορίζεται η κωδικοποίηση των χαρακτήρων του αρχείου, εδώ εάν δε τοποθετηθεί τίποτα χρησιμοποιείται η προκαθορισμένη κωδικοποίηση του λειτουργικού συστήματος.

Κατάσταση mode	Λειτουργία
`mode='r'`	Άνοιγμα αρχείου για ανάγνωση κειμένου
`mode='w'`	Άνοιγμα αρχείου για εγγραφή, εάν δεν υπάρχει το αρχείο δημιουργείται, εάν υπάρχει ήδη, διαγράφονται πρώτα τα παλαιά περιεχόμενά του για να εγγραφούν τα νέα
`mode='a'`	Άνοιγμα νέου αρχείου για εγγραφή ή εάν υπάρχει ήδη, για προσάρτηση νέων περιεχομένων στο τέλος του

`mode='x'`	Άνοιγμα νέου αρχείου για εγγραφή, εάν υπάρχει ήδη επιστρέφεται εξαίρεση `FileExistsError`
`mode='r+'`	Άνοιγμα αρχείου για ανάγνωση και εγγραφή
`mode='w+'`	Άνοιγμα αρχείου για ανάγνωση και εγγραφή, τα προηγούμενα περιεχόμενα διαγράφονται
Κωδικοποιήσεις παράμετρος encoding	**Λειτουργία**

`Εάν παραληφθεί`	Χρησιμοποιείται η κωδικοποίηση του ΛΣ.
	Η κωδικοποίηση συστήματος μπορεί να εμφανιστεί φορτώνοντας το module `locale` και καλώντας τη μέθοδο `.getencoding()` ή τη μέθοδο `.getlocale()`
	`import locale`
	`locale.getencoding()`
	`locale.getlocale()`

`encoding='cp1253'`	Χρησιμοποιείται από τη Microsoft και συνήθως είναι η default τιμή σε ΛΣ windows
`encoding='utf-8'`	Κωδικοποίηση μεταβλητού μήκους από 1 έως 4 bytes. Μπορούν να κωδικοποιηθούν οι χαρακτήρες όλων των υπαρκτών γλωσσών. Είναι η προκαθορισμένη (default)τιμή για ΛΣ Linux
`encoding='iso 8859-7'`	Διεθνές πρότυπο κωδικοποίησης με 8-bits σχεδιάστηκε για να καλύπτει την ελληνική γλώσσα.

Παραδείγματα δημιουργίας αντικειμένου διαχείρισης αρχείου:

```
> file1 = open('data.txt', 'w')
> file2 = open('c:\\python311\\data.txt', 'r')
> file3 = open('/home/user/my_files/data.txt', 'a')
```

Τα αντικείμενα αρχείων κειμένου, έχουν και αυτά λειτουργικότητα και συμπεριφορά, όπως και τα άλλα αντικείμενα που συναντήθηκαν σε προηγούμενα κεφάλαια (πχ συμβολοσειρών). Αυτή η λειτουργικότητα εκφράζεται με μεθόδους του αντικειμένου `(.read()`, `.write()` κ.ά. θυμίζουμε ότι οι μέθοδοι στα αντικείμενα είναι συναρτήσεις ειδικής μορφής και θα αναλυθούν σε επόμενο μάθημα).

Τα αντικείμενα αρχείων κειμένου έχουν όμως και χαρακτηριστικά, (ιδιότητες, attributes) που εκφράζουν την κατάσταση στην οποία βρίσκεται το αντικείμενο. Ως τέτοια χαρακτηριστικά εδώ έχουμε:

Χαρακτηριστικό	**Επιστρέφει την τιμή του/της**

`file.name`	ονόματος του αρχείου
`file.mode`	κατάστασης προσπέλασης που άνοιξε
`file.encoding`	κωδικοποίησης των περιεχομένων του
`file.closed`	**False** (Εάν είναι ανοικτό), **True** (Εάν είναι κλειστό)

Σημείωση

Παρατηρήστε ότι τα χαρακτηριστικά ακολουθούν τη σημειογραφία τελείας (dot notation) όπως οι μέθοδοι, δεν έχουν όμως παρενθέσεις (διότι δεν είναι συναρτήσεις αφού δεν κάνουν κάποια λειτουργία και βέβαια δεν δέχονται παραμέτρους).

Παράδειγμα 8.9

Παράδειγμα λειτουργιών διαχείρισης αρχείων κειμένου

```
# Δημιουργία αρχείου κειμένου file και άνοιγμα για εγγραφή σε αυτό ('w')
file = open('data.txt', 'w')
type(file)
<class '_io.TextIOWrapper'>
# Κλείσιμο αρχείου κειμένου file με τη μέθοδο close()
file.close()
```

Παρατηρήσεις

- *Το όνομα file είναι μία αναφορά στο αντικείμενο αρχείου (όπως και με τις συμβολοσειρές), ενώ η μέθοδος `.close()` κλείνει το αρχείο ώστε να μην προκληθεί κάποιο σφάλμα όσο είναι ανοιχτό. ΠΡΟΣΟΧΗ εάν δεν κλείσει το αρχείο ΔΕΝ εξασφαλίζεται ότι τα δεδομένα θα εγγραφούν σωστά σε αυτό.*
- *Οι παραπάνω εντολές δημιούργησαν ένα νέο αρχείο με όνομα data.txt στην τρέχουσα τοποθεσία (θα δούμε παρακάτω που βρίσκεται αυτή) και στη συνέχεια το έκλεισαν χωρίς να εγγράψουν τίποτα μέσα σε αυτό. Εάν το αρχείο προϋπήρχε και είχε δεδομένα, αυτά διαγράφηκαν.*

8.4.2 Εύρεση τρέχοντος καταλόγου

Τρέχοντας κατάλογος εργασίας (current working directory) είναι ο κατάλογος ο οποίος θα αναζητήσει η Python κάποιο αρχείο εάν δοθεί μία εντολή ανάγνωσης ή εγγραφής ή δημιουργίας ή μία άλλη εντολή που αφορά σε εργασία σε κάποιο αρχείο χωρίς να προσδιοριστεί κάποια άλλη διαδρομή (path) που θα βρεθεί αυτό.

Παράδειγμα 8.10

1. *Ανοίξτε το κέλυφος και δώστε τις επόμενες εντολές:*

```
# Σε ΛΣ windows θα δείτε κάτι παρόμοιο με το παρακάτω
>>> import os # Φόρτωσε το άρθρωμα λογισμικού os
>>> os.getcwd() # Εμφάνισε τον τρέχοντα κατάλογο
'C:\\Python311'
```

```
# Σε ΛΣ linux θα δείτε κάτι παρόμοιο με το παρακάτω
>>> import os # Φόρτωσε το άρθρωμα λογισμικού os
>>> os.getcwd() # Εμφάνισε τον τρέχοντα κατάλογο
'/home/user' # όπου user το όνομα του χρήστη στο linux
```

Μπορείτε να καταλάβετε τι βρέθηκε από τις προηγούμενες εντολές;

Απάντηση: *βρέθηκε χρησιμοποιώντας τη μέθοδο* `os.getcwd()` *(get current working directory) ο τρέχον κατάλογος που αποθηκεύτηκε το αρχείο data.txt.*

2. *Ανοίξτε το κέλυφος και δώστε την επόμενη εντολή για να εμφανίσετε το zen της Python: στη συνέχεια ανοίξτε με κάποιον συντάκτη κειμένου (πχ Notepad σε περιβάλλον Windows ή Pluma σε περιβάλλον Ubuntu(Mate)) το αρχείο data.txt και αντιγράψτε σε αυτό με αντιγραφή και επικόλληση από το κέλυφος το zen της Python. Στη συνέχεια αποθηκεύστε το αρχείο στον τρέχοντα κατάλογο που βρήκατε παραπάνω.*

```
>>> 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!
```

Το άρθρωμα λογισμικού (module) `os` *(operating system) θα μελετηθεί αναλυτικότερα παρακάτω.*

8.4.3 Ανάγνωση των περιεχομένων ενός αρχείου κειμένου μέθοδοι .read(), .seek(), .tell()

Για την ανάγνωση των περιεχομένων ενός αρχείου κειμένου δημιουργείται πρώτα ένα αντικείμενο με παράμετρο `mode='r'` στη συνέχεια καλώντας τη μέθοδο `.read()` δημιουργείται ένα αντικείμενο string με όλα τα περιεχόμενα του αρχείου κειμένου.

Παράδειγμα 8.11

Άνοιγμα του αρχείου data.txt και εμφάνισης ΟΛΩΝ των περιεχομένων του

```
>>> f = open('data.txt', 'r')
>>> text = f.read()
>>> print(text)
>>> f.close()
```

Το όρισμα της κωδικοποίησης στο παραπάνω παράδειγμα δεν είναι απαραίτητο εφόσον η εφαρμογή συντάκτη μέσω της οποίας δημιουργήθηκε το αρχείο όπως και η εντολή `open` χρησιμοποίησαν τις ίδιες προκαθορισμένες ρυθμίσεις κωδικοποίησης χαρακτήρων του λειτουργικού συστήματος.

`file.read(size)`: Επιστρέφει ως συμβολοσειρά τα περιεχόμενα μεγέθους size σε bytes του αρχείου

Σημειώσεις

α. Όταν ένα αρχείο ανοίγει είτε για εγγραφή είτε για ανάγνωση δημιουργείται ένας εσωτερικός δείκτης στο αρχείο ο οποίος μετακινείται κάθε φορά που γίνεται μια εγγραφή ή μία ανάγνωση σε αυτό.

β. Η μέθοδος `.read()` εάν εκτελεστεί αμέσως μόλις ανοίξει το αρχείο διαβάζει όλα τα περιεχόμενα και μετακινεί το δείκτη του αρχείου στο τέλος του.

γ. Για να διατηρήσουμε τα δεδομένα πρέπει να δημιουργήσουμε μία αναφορά σε αυτά (μία μεταβλητή string) στο προηγούμενο παράδειγμα είναι η μεταβλητή `text`.

δ. Στη μέθοδο `.read()` τοποθετώντας ένα ακέραιο ως όρισμα ορίζεται το μέγεθος του κειμένου (size) που θέλουμε να διαβάσουμε σε χαρακτήρες. 'Ετσι με τη κλήση της μεθόδου `.read(size)` σε κάποιο αντικείμενο αρχείου μπορούμε να διαβάσουμε συγκεκριμένο πλήθος χαρακτήρων (ο δείκτης του αρχείου τότε μετακινείται αναλόγως). Εάν στη συνέχεια εκτελεστεί η μέθοδος `.read()` διαβάζεται το περιεχόμενο του αρχείου από τη θέση του δείκτη και μέχρι το τέλος του.

ε. Παραδείγματα κλήσης της μεθόδου `.read()`:

> `data = f.read()` : Επιστρέφει σε συμβολοσειρά data όλα τα περιεχόμενα του αρχείου από τη θέση του δείκτη και μετά.

> `data = f.read(20)` : Επιστρέφει 20 χαρακτήρες από τη θέση του δείκτη και μετά στη συμβολοσειρά με όνομα data.

Μέθοδοι .seek(), .tell()

Για να μετακινηθεί ο δείκτης του αρχείου σε κάποια θέση του κειμένου χρησιμοποιείται η μέθοδος `.seek()`. ΠΡΟΣΟΧΗ η μέθοδος μετράει σε bytes όχι σε χαρακτήρες οπότε για αρχεία κειμένου συνιστάται η χρήση της μόνο σε κωδικοποιήσεις αρχείων όπου 1 char = 1 byte. (πχ ansi, cp1253, iso 8859-7 και όχι για utf-8 όπου χρησιμοποιούνται από 1-4 bytes ανάλογα το χαρακτήρα).

`file.seek(position)` :Μετακίνησε το δείκτη στη θέση position.

Παραδείγματα κλήσης της μεθόδου `.seek()`

> `f.seek(0)` : Μετακίνησε το δείκτη στην αρχή

> `f.seek(15)` : Μετακίνησε το δείκτη στη 15η θέση από την αρχή

> `f.seek(0, 2)` : Μετακίνησε το δείκτη στην τελευταία θέση του αρχείου
> `f.tell()` : Επιστρέφει τη θέση του δείκτη αρχείου

Παρατηρήσεις

α. Κάθε λάθος στη διαδρομή του δίσκου που βρίσκεται το αρχείο ή στο όνομα του αρχείου ή στην κωδικοποίηση θα έχει ως συνέπεια κάποιο λάθος χρόνου εκτέλεσης (run time error). Έτσι μία καλή πρακτική όταν δουλεύουμε με αρχεία είναι η χρήση των εντολών διαχείρισης λαθών `try - except – else - finally`.

```
try:
    f = open('data.txt', 'r')
except:
    print('Το αρχείο','data.txt','δεν βρέθηκε ή έχει λάθος κωδικοποίηση')
else:
    text = f.read()
    print(text)
finally:
    f.close()
```

β. Ένας άλλος τρόπος χειρισμού των αρχείων είναι χρησιμοποιώντας την εντολή `with`. Το πλεονέκτημα με τη χρήση της εντολής αυτής είναι ότι κλείνει το αρχείο όταν τελειώσει η εκτέλεσή της αυτόματα (άρα δεν χρειάζεται η μέθοδος `.close()`) ακόμα και εάν συμβεί κάποιο σφάλμα.

```
with open('data', 'r') as f:
    text = f.read()
    print(text)
```

γ. Ή ακόμα καλύτερα συνδυασμός των α, β με εντολές διαχείρισης λαθών για αποφυγή run time errors:

```
try:
    with open('data', 'r') as f:
        text = f.read()
        print(text)
except:
    print('Λάθος στην τοποθεσία-όνομα-κωδικοποίηση του αρχείου')
else:
    print('Η λειτουργία εκτελέστηκε χωρίς σφάλματα')
```

8.4.4 Άνοιγμα για ανάγνωση των περιεχομένων ενός αρχείου κειμένου ανά γραμμή

Κάθε αρχείο ενός ΛΣ έχει συνήθως 3 μέρη πληροφορίας:

α. την κεφαλίδα με πληροφορίες για το αρχείο,
β. τα δεδομένα κειμένου,

γ. το τέλος του αρχείου το οποίο σηματοδοτείται με έναν ειδικό χαρακτήρα που υποδηλώνει το τέλος του, το τέλος του αρχείου στην Python υποδηλώνεται με μία κενή συμβολοσειρά ''

```
>>> eof = ''
>>> not eof
True
```

Η ανάγνωση ενός αρχείου ανά γραμμή είναι ίσως η πιο συνηθισμένη εργασία διότι πολύ συχνά τα αρχεία αυτά ανά γραμμή κειμένου περιέχουν δεδομένα εγγραφών. Σε αυτές τις περιπτώσεις η επεξεργασία τους είναι σκόπιμο να γίνει με αυτόν τον τρόπο (ανά γραμμή). Το τέλος κάθε γραμμής υποδηλώνεται με τον ειδικό χαρακτήρα αλλαγής γραμμής '\n'
Για τη λειτουργία ανάγνωσης των περιεχομένων ενός αρχείου κειμένου και επεξεργασίας **ανά γραμμή** υπάρχουν περισσότερες από μία τεχνικές τις οποίες θα περιγράψουμε παρακάτω.

ΠΡΟΒΛΗΜΑ
Ανάγνωση περιεχομένων αρχείου κειμένου ανά γραμμή και υπολογισμός του μήκους κάθε γραμμής.

Α. Τρόπος απευθείας διάσχιση με χρήση της εντολής for
Κάθε αντικείμενο διαχείρισης αρχείου κειμένου είναι iterable (διασχίσιμο) με κάθε γραμμή του (line) να αποτελεί ένα στοιχείο (element/ item) οπότε η διάσχισή του γίνεται όπως ακριβώς στις συμβολοσειρές, στις λίστες, στις πλειάδες την εντολή ορισμένων επαναλήψεων for.

Παράδειγμα 8.12

Στο κέλυφος ή στο IDLE εκτελέστε τις παρακάτω εντολές:

Το αρχείο data.txt περιέχει το zen της Python

```
α.Υλοποίηση ανάγνωσης ανά γραμμή με τις εντολές open/close
# Άνοιγμα του αρχείου για ανάγνωση
f = open('data.txt', 'r')
# Διάσχιση του αρχείου ανά γραμμή
for line in f:
    # Καθαρισμός αρχής και τέλους κάθε γραμμής από whitespaces
    line = line.strip()
    # Εμφάνιση μήκους γραμμής και περιεχομένων της
    print(len(line), line)
f.close()

32 The Zen of Python, by Tim Peters
0
0
30 Beautiful is better than ugly.
33 Explicit is better than implicit.
30 Simple is better than complex.
```

```
35 Complex is better than complicated.
27 Flat is better than nested.
28 Sparse is better than dense.
19 Readability counts.
...
```

β.Υλοποίηση ανάγνωσης ανά γραμμή με την εντολή with

```
# Διάσχιση αρχείου με for αλλά
# με χρήση της εντολής with χωρίς f.close()
with open('data.txt', 'r') as f:
    for line in f:
        line = line.strip()
        print(len(line), line)
```

γ.Υλοποίηση ανάγνωσης ανά γραμμή με την εντολή with & try – except

```
# Κώδικας με διαχείριση σφαλμάτων
try:
    with open('data.txt', 'r') as f:
        for line in f:
            line = line.strip()
            print(len(line), line)
except IOError:
    print('Το αρχείο',f.name,'δεν βρέθηκε')
except UnicodeError:
    print('Το αρχείο',f.name,'έχει λάθος κωδικοποίηση')
else:
    print(f.name,'closed status',f.closed)
```

Παρατηρήσεις

- *Ο α τρόπος ανάγνωσης θεωρείται ο πιο «κλασσικός»*
- *Ο β τρόπος ανάγνωσης είναι πιο «μοντέρνος»*
- *Ο γ τρόπος ανάγνωσης πιο «επαγγελματικός», αφού τα run time errors όταν γίνονται εργασίες με αρχεία είναι πολύ συχνά*
- *Δοκιμάστε να εκτελέσετε τα παραπάνω τμήματα κώδικα χωρίς τη χρήση της μεθόδου* `.strip()` *τι παρατηρείτε ;*

Β. Τρόπος με χρήση της μεθόδου readlines()

Όταν καλείται η μέθοδος `.readlines()` σε ένα αντικείμενο αρχείου κειμένου αυτή επιστρέφει μία λίστα όπου τα στοιχεία της λίστας είναι οι γραμμές του αρχείου κειμένου.

`file.readlines()` : Η μέθοδος με μία κλήση της επιστρέφει μία λίστα με στοιχεία τις γραμμές του κειμένου

Παράδειγμα 8.13

Υλοποίηση ανάγνωσης ανά γραμμή με την μέθοδο `.readlines()` *και την εντολή επανάληψης* `for`

```
with open('data.txt', 'r') as f:
    # Η λίστα lines περιέχει όλες τις γραμμές του αρχείου
    lines = f.readlines()
    # διάσχιση της λίστας lines
    for line in lines:
        line = line.strip()
        print(len(line), line)
```

Γ. Τρόπος με χρήση της μεθόδου readline()

Η μέθοδος `.readline()` επιστρέφει κάθε φορά που καλείται μία γραμμή από το αρχείο μετακινώντας το δείκτη του αρχείου στο τέλος της γραμμής.

`file.readline()`: Η μέθοδος επιστρέφει με κάθε κλήση της μία γραμμή από το αρχείο κειμένου μετακινώντας ταυτόχρονα και το δείκτη ανάγνωσης του αρχείου

Παράδειγμα 8.14

Υλοποίηση ανάγνωσης ανά γραμμή με την μέθοδο `.readline()` *και την εντολή επανάληψης* `while`

```
with open('data.txt', 'r') as f:
    # Διάβασε μία γραμμή
    line = f.readline()
    # επανάλαβε όσο η γραμμή δεν είναι άδεια(δηλαδή όσο δεν έχεις βρει EOF)
    while line: # ισοδύναμο με την εντολή while line != ''
        print(line.rstrip())
        line = f.readline()
```

Παρατηρήσεις

Πολλές φορές η επιλογή της τεχνικής εξαρτάται:

- *από τις προτιμήσεις του προγραμματιστή,*
- *τις συνήθειές του,*
- *τις εργασίες που θέλει να εκτελέσει,*
- *τον τρόπο που οργανώνονται τα δεδομένα του προβλήματος στο αρχείο*

8.4.5 Άνοιγμα αρχείου και προσάρτηση περιεχομένων

Μία άλλη συχνή λειτουργία στα αρχεία είναι η προσάρτηση (προσθήκη) νέων περιεχομένων σε αυτά. Η προσθήκη αυτή γίνεται στο τέλος του αρχείου. Σε αυτή την περίπτωση χρησιμοποιείται η κατάσταση ανοίγματος με προσάρτηση (append: `'a'`).

```
> file = open('data.txt', 'a')
```

Παράδειγμα 8.15

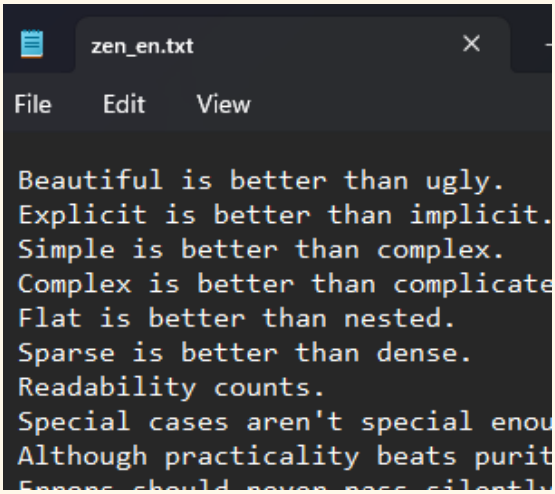
Διατίθενται τα αρχεία:

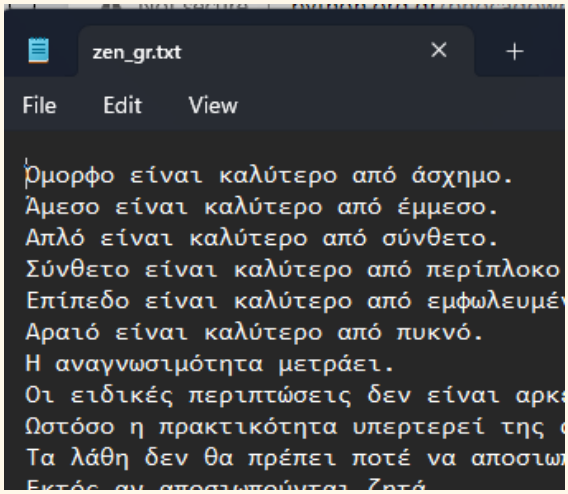
1. *zen_en.txt το οποίο περιέχει το zen της Python στα αγγλικά*
2. *zen_ gr.txt το οποίο περιέχει το zen της Python στα ελληνικά*
3. *zen.txt το οποίο περιέχει τις πηγές του περιεχομένου των περιεχομένων αρχείων αυτών*

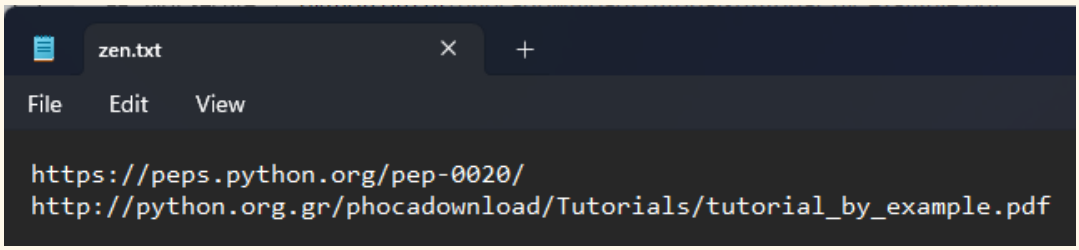
Και ζητείται:

α. Στο ήδη υπάρχον αρχείο zen.txt να τοποθετηθούν στο τέλος του τα περιεχόμενα των αρχείων 1 & 2. Τα περιεχόμενα των 3 αρχείων να χωρίζονται με μία κενή γραμμή

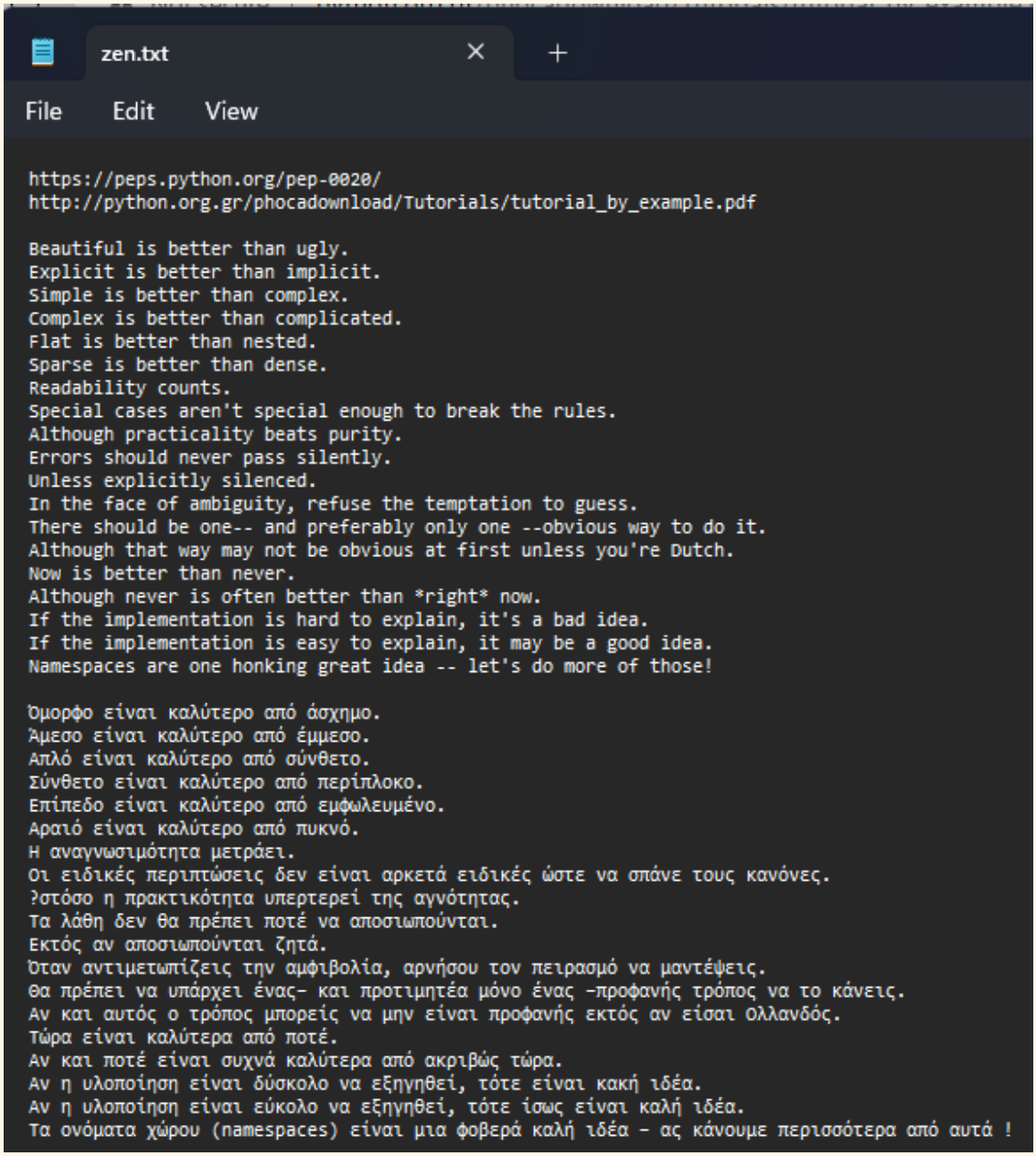
β. Να δημιουργηθεί νέο αρχείο zen_en_gr.txt το οποίο να περιέχει τις γραμμές των αρχείων 1 & 2 εναλλάξ.







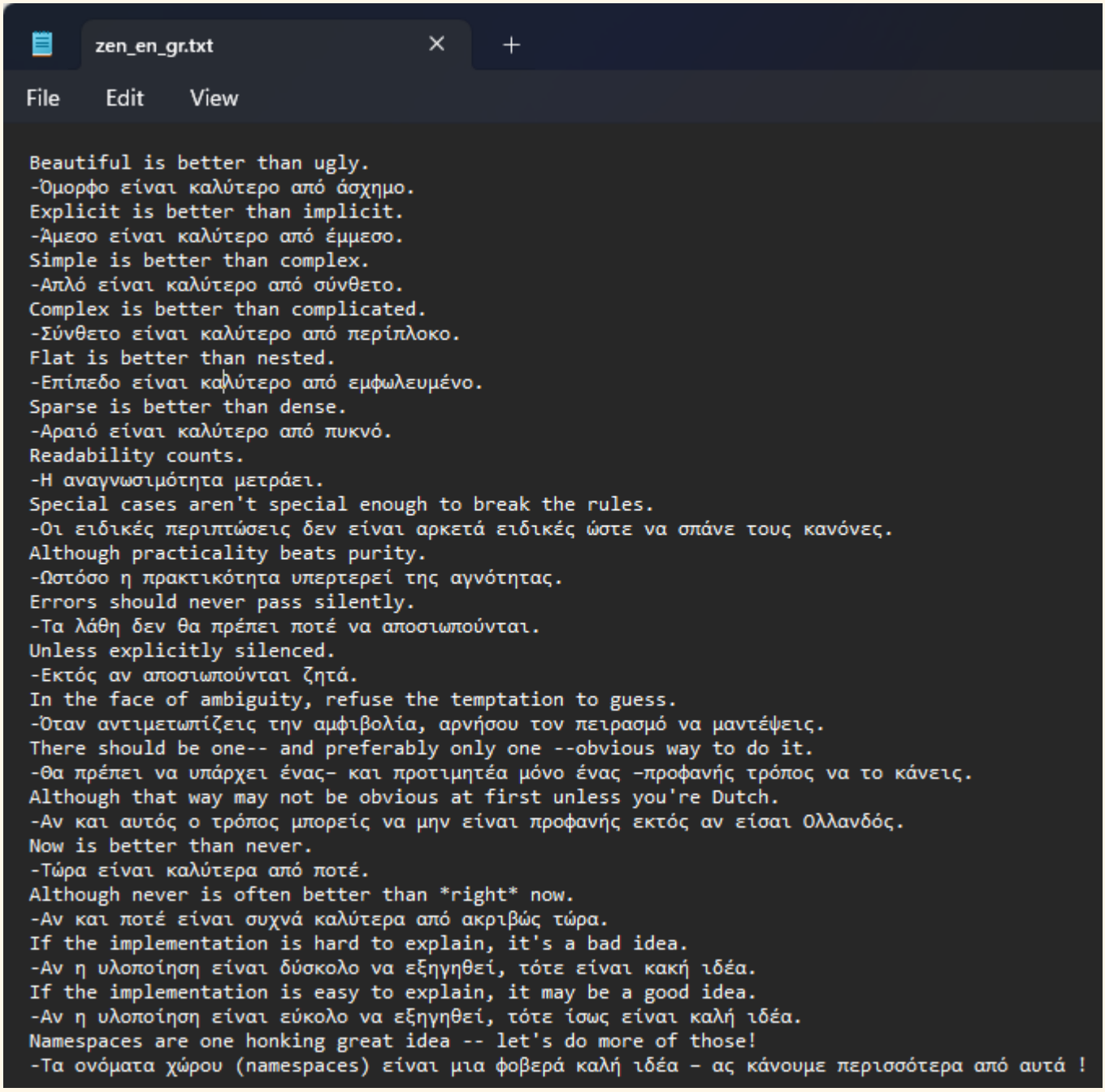
Από τα παραπάνω αρχεία ζητείται να δημιουργηθεί το επόμενο



Λύση του ερωτήματος α

```
file1 = open('c:\\python311\\my_scripts\\sample_files\\zen_en.txt', 'r')
file2 = open('c:\\python311\\my_scripts\\sample_files\\zen_gr.txt', 'r')
file_a = open('c:\\python311\\my_scripts\\sample_files\\zen.txt', 'a')
text1 = file1.read()
text2 = file2.read()
file_a.write('\n' + text1)
file_a.write('\n' + text2)
file1.close()
file2.close()
file_a.close()
```

Ζητείται τώρα από τα αρχεία zen_en.txt και zen_gr.txt να δημιουργηθεί το επόμενο αρχείο zen_en_gr.txt



Λύση του ερωτήματος β

```
file1 = open('c:\\python311\\my_scripts\\sample_files\\zen_en.txt', 'r')
file2 = open('c:\\python311\\my_scripts\\sample_files\\zen_gr.txt', 'r')
file_b = open('c:\\python311\\my_scripts\\sample_files\\zen_en_gr.txt',
'w')
while True:
    line1 = file1.readline()
    line2 = file2.readline()
    if line1 and line2:
        file_b.write(line1)
        file_b.write('-' + line2)
    else:
        break
```

```
file1.close()
file2.close()
file_b.close()
```

8.4.6 Ανάγνωση αρχείου και επεξεργασία περιεχομένου ανά χαρακτήρα

Πολύ συχνά χρειάζεται η επεξεργασία ενός κειμένου χαρακτήρα προς χαρακτήρα για διάφορους λόγους:

- για στατιστικά στοιχεία
- για διορθώσεις και αλλαγές
- για εντοπισμό γραμμών με συγκεκριμένα χαρακτηριστικά
- για διαχωρισμό πεδίων όταν μία γραμμή αντιστοιχεί σε μία εγγραφή κ.ά.

Ακολουθεί ένα τέτοιο παράδειγμα το οποίο ανήκει στην κατηγορία εξαγωγής στατιστικών στοιχείων.

Παράδειγμα 8.16

Ανοίξτε το αρχείο data.txt και δημιουργήστε και εμφανίστε ένα πίνακα συχνοτήτων εμφάνισης ανά λατινικό γράμμα των περιεχομένων του αρχείου κειμένου.

Σημειώσεις

- *Στον υπολογισμό των συχνοτήτων θεωρήστε τους κεφαλαίους και τους πεζούς χαρακτήρες ταυτόσημους.*
- *Τα πεζά γράμματα του λατινικού αλφαβήτου μπορείτε να τα λάβετε είτε χειροκίνητα (πληκτρολογώντας τα) είτε εισάγοντας το module* `string` *και το χαρακτηριστικό του* `ascii_lowercase`.

Περιγραφική ανάλυση της λύσης

1. *Δημιουργία μίας δομής δεδομένων με μετρητές για τον πίνακα συχνοτήτων (επιλέξτε dictionary/ list)*
2. *Άνοιγμα του αρχείου για ανάγνωση*
3. *Ανάγνωση του περιεχομένου του με κάποια από τις τεχνικές που αναφέρθηκαν προηγούμενα (απευθείας διάσχιση, μέθοδος* `.read()`, `.read(size)`, `.readline()`, `.readlines()`)
4. *Διάσχιση του περιεχομένου του αρχείου ανά χαρακτήρα*
   4.1. *Κάθε φορά που ένας χαρακτήρας αποτελεί ένα κεφαλαίο ή ένα πεζό λατινικό γράμμα θα καταμετρείται στη δομή των μετρητών.*
5. *Εμφάνιση των περιεχομένων της δομής του πίνακα συχνοτήτων*

Η γλώσσα Python δίνει τη δυνατότητα για πολλές προγραμματιστικές τεχνικές και αλγοριθμικές προσεγγίσεις. Πριν δείτε μία από τις ενδεικτικές λύσεις προσπαθήστε να δώσετε τη δική σας.

Ακολουθούν εναλλακτικές ενδεικτικές προσεγγίσεις οι δύο πρώτες χρησιμοποιούν ως δομή δεδομένων το λεξικό και τη μέθοδο `.read()` *ενώ η τρίτη λύση τη δομή δεδομένων της λίστας.*

Σημειώσεις

- *Για διδακτικούς και για λόγους απλοποίησης του κώδικα δε χρησιμοποιούνται εντολές διαχείρισης λαθών.*
- *Σε πραγματικές συνθήκες και σε εφαρμογές που έχουν λειτουργίες με αρχεία η χρήση τέτοιων εντολών είναι επιβεβλημένη.*

α. Λύση με Λεξικό και Ανάγνωση του κειμένου ανά χαρακτήρα

```
import string # Εισαγωγή του αρθρώματος
ab = string.ascii_lowercase # Οι λατινικοί πεζοί χαρακτήρες
frequency = {} # Δημιουργία κενού λεξικού συχνοτήτων
# Αρχικοποίηση λεξικού με
# δημιουργία κλειδιών(Πεζό Λατινικό Γράμμα): Τιμή (0)
for letter in ab:
    frequency[letter] = 0
# Ανοιγμα αρχείου για ανάγνωση
with open('c:\\python311\\data.txt', 'r') as f:
    # Διάβασε ένα χαρακτήρα και μετακίνησε το δείκτη ανάγνωσης κατά 1
    char = f.read(1)
    # Όσο ο χαρακτήρας έχει περιεχόμενο (στην ουσία while not EOF)
    while char:
        if char.lower() in ab: # Εάν ο χαρακτήρας είναι πεζό λατ. γράμμα
            # Βρες το κλειδί και αύξησε τη συχνότητα κατά 1
            frequency[char.lower()] += 1
        char = f.read(1) # διάβασε τον επόμενο χαρακτήρα
# Εμφάνισε τα περιεχόμενα του λεξικού συχνοτήτων
for letter in ab:
    print (letter, frequency[letter])
```

β. Λύση με Λεξικό και Ανάγνωση ΟΛΟΥ του κειμένου μαζί, χρήση μεθόδου count

```
import string # Εισαγωγή του αρθρώματος
ab = string.ascii_lowercase # Οι λατινικοί πεζοί χαρακτήρες
# Δημιουργία & αρχικοποίηση λεξικού συχνοτήτων με dict comprehension
frequency = {letter:0 for letter in ab}
# Άνοιγμα αρχείου για ανάγνωση
with open('c:\\python311\\data.txt', 'r') as f:
    text = f.read() # Διάβασε όλο το περιεχόμενο σε μία συμβολοσειρά
    for letter in ab: # Για κάθε λατινικό γράμμα
        # Τοποθέτησε ως τιμή του κλειδιού
        # το πλήθος εμφάνισης του πεζού + του κεφαλαίου γράμματος
        frequency[letter] = text.count(letter) + text.count(letter.upper())
# Εμφάνισε τα περιεχόμενα του λεξικού συχνοτήτων
for letter in ab:
```

```
    print(letter, frequency[letter])
```

γ. Λύση με λίστα και διάσχιση με for ανά γραμμή και χρήση της μεθόδου find

```
from string import ascii_lowercase as ab
file = open('c:\\python311\\data.txt', 'r')
# Δημιούργησε μία λίστα μετρητών
frequency = [0 for letter in ab]
for line in file: # Διάσχισε το αρχείο ανά γραμμή
    for char in line: # Διάσχισε τη γραμμή ανά χαρακτήρα
        if char.lower() in ab: # Εάν ο χαρακτήρας είναι γράμμα
            letter = char.lower() # Κάνε πεζό τον χαρακτήρα
            pos = ab.find(letter) # Ψάξε τη θέση του στο string ab
            # Οι χαρακτήρες του ab είναι σε αντίστοιχες θέσεις
            # με τους μετρητές εμφάνισης γραμμάτων της frequency
            frequency[pos] += 1 # Αύξησε τον αντίστοιχο μετρητή γράμματος
file.close() # Κλείσε το αρχείο
# Εμφάνισε τη συχνότητα εμφάνισης των γραμμάτων
for i in range(len(ab)):
    print(ab[i], frequency[i])
```

Παρατηρήσεις

- *Αφού τις εκτελέσετε και τις μελετήσετε επιλέξτε αυτή που σας αρέσει περισσότερο*
- *Στη συνέχεια μπορείτε να συζητήσετε τους λόγους της επιλογής σας*
- *Σε ποιες περιπτώσεις δεδομένων κειμένου θα ήταν προτιμότερη η κάθε λύση*

8.4.7 Εγγραφή σε αρχεία κειμένου, μέθοδος write()

Η μέθοδος `.write(<συμβολοσειρά>)` εγγράφει τη συμβολοσειρά σε ένα αρχείο που ανοίχθηκε για εγγραφή (`mode='w'`)ή για προσάρτηση (`mode='a'`) όπως είδαμε και σε προηγούμενο παράδειγμα.

`file.write(string)` : Εγγράφεται η συμβολοσειρά στο αρχείο

Σημειώσεις

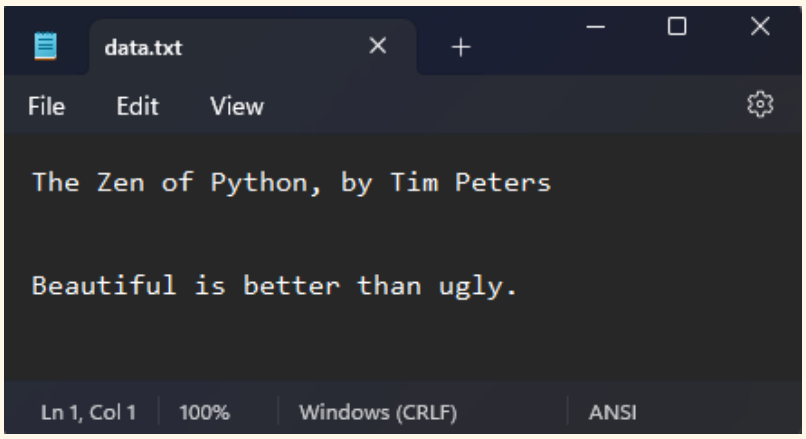
- *Κάθε φορά χρησιμοποιείται η εντολή print για την εμφάνιση μίας συμβολοσειράς μετά το πέρας της εμφάνισης ο δρομέας μετακινείται αυτόματα στην επόμενη γραμμή.*
- *Κάθε φορά που εγγράφεται μία γραμμή όμως σε ένα αρχείο αυτό δεν συμβαίνει έτσι πρέπει η συμβολοσειρά που επιθυμεί ο προγραμματιστής να εγγράψει σε μία γραμμή του αρχείου στο τέλος της να περιλαμβάνει το χαρακτήρα αλλαγής γραμμής (`'\n'`). Αλλιώς η επόμενη εγγραφή θα γίνει στην ίδια γραμμή του αρχείου.*

Παράδειγμα 8.17

Το αποτέλεσμα των εντολών:

```
f = open('data.txt', 'w')
line = 'The Zen of Python, by Tim Peters' + '\n'
# Εγγράφεται η γραμμή χωρίς να μετακινηθεί ο δρομέας στην επόμενη
f.write(line)
line = '\n'
f.write(line) # Γίνεται αλλαγή γραμμής
f.write(line) # Εγγράφεται μία κενή γραμμή
line = 'Beautiful is better than ugly.' + '\n'
f.close()
```

Θα είναι:



8.4.8 Πρόβλημα, εγγραφή βαθμολογίων σπουδαστή σε αρχείο

Ακολουθεί ένα τυπικό πρόβλημα εγγραφής δεδομένων σε αρχείο.

Έστω ότι μία λίστα grades περιέχει τις βαθμολογίες ενός σπουδαστή του Γεώργιου Αναγνωστόπουλου, grades = [7, 8, 6, 9, 10] και ζητείται η δημιουργία ενός αρχείου κειμένου όπου η 1η γραμμή θα περιέχει τα δεδομένα του σπουδαστή ως μία εγγραφή (record) δηλαδή θα έχει τη μορφή:

`name,grade1,grade2,grade3,…<αλλαγή γραμμής>`

Ζητείται δηλαδή η δημιουργία ενός csv (comma-separated values) αρχείου.

Σημείωση

Υπενθυμίζεται ότι εφόσον πρόκειται για αρχείο κειμένου οι ακέραιοι βαθμοί πρέπει να μετατραπούν πρώτα σε μορφή string.

1η Λύση

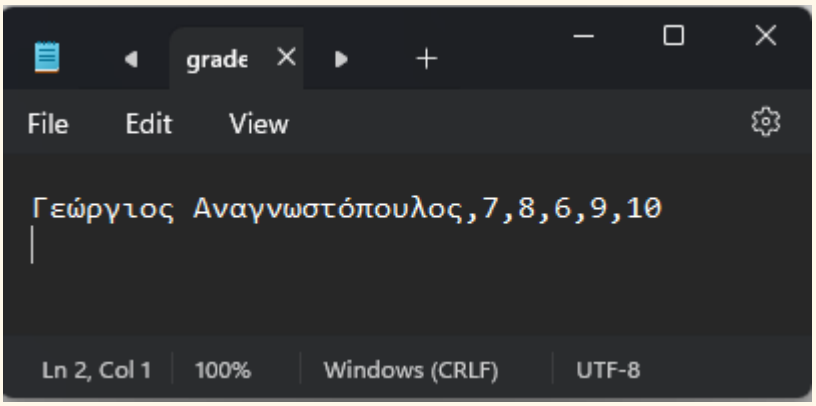
```
name = 'Γεώργιος Αναγνωστόπουλος'
grades = [7, 8, 6, 9, 10]
s = ','
for grade in grades:
```

```
    s += str(grade) + ','
file = open('grades.txt', mode='w', encoding='utf-8')
line = name + s[:-1] + '\n'
file.write(line)
file.close()
```

2η Λύση

```
name = 'Γεώργιος Αναγνωστόπουλος'
grades = [7, 8, 6, 9, 10]
file = open('grades.txt', mode='w', encoding='utf-8')
line = name + ','.join(map(str,grades)) + '\n'
file.write(line)
file.close()
```

Το αποτέλεσμα των εντολών και των δύο λύσεων θα είναι:



8.4.9 Επιλυμένη δραστηριότητα, εισαγωγή βαθμολογίας προόδου σπουδαστών

Τελειώνοντας το 1ο εξάμηνο της ειδικότητας «Τεχνικός Συστημάτων Ανοιχτού Λογισμικού» η γραμματεία του ΔΙΕΚ ζητά μία εφαρμογή καταχώρησης των βαθμολογιών της προόδου των σπουδαστών της ειδικότητας. Σε αυτή την εφαρμογή θα ζητούνται τα ονόματα των σπουδαστών και οι βαθμολογίες προόδου για κάθε ένα σπουδαστή. Τα δεδομένα αυτά θα αποθηκεύονται σε ένα αρχείο κειμένου με όνομα OpenSoftwareTechnician1.txt.

Σημειώσεις

1. Η εισαγωγή σπουδαστών θα σταματά όταν ως όνομα δοθεί το κενό.
2. Η βαθμολογία δίνεται με έλεγχο εγκυρότητας δεδομένων στα όρια [1, 10] (1 λαμβάνουν οι απόντες Παρ 17, Άρθρο 10, ΦΕΚ 5837/2021).
3. Επειδή ο χρήστης στη γραμματεία είναι αρχάριος η εφαρμογή πρέπει να έχει αμυντικό σχεδιασμό στα δεδομένα εισόδου ώστε να αποτρέπονται τα runtime errors από λανθασμένες καταχωρήσεις.
4. Η πρώτη γραμμή του αρχείου θα περιέχει τη δομή των εγγραφών (Όνομα Σπουδαστή, ΟνΜαθηματος1, ΟνΜαθήματος2, ..., ΜέσοςΌρος).
5. Τα δεδομένα θα διαχωρίζονται με , (csv file).
6. Στο τέλος του προγράμματος θα εμφανίζεται και η τοποθεσία αποθήκευσης του αρχείου.

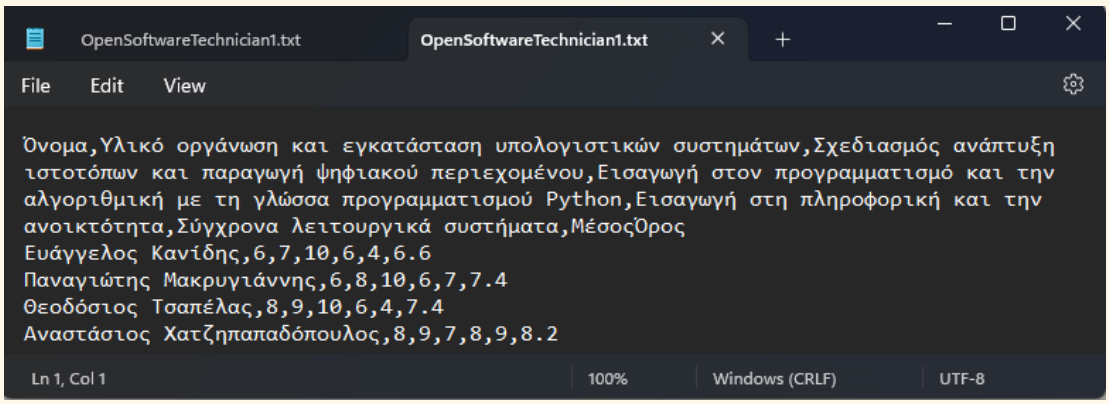
```
import os # Άρθρωμα Διαχείρισης ΛΣ

# Δηλώσεις σταθερών
FILE_NAME = 'OpenSoftwareTechnician1.txt'
COURSE_NAMES = (
    'Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων',
    'Σχεδιασμός ανάπτυξη ιστοτόπων και παραγωγή ψηφιακού περιεχομένου',
    'Εισαγωγή στον προγραμματισμό και την αλγοριθμική με τη γλώσσα
προγραμματισμού Python',
    'Εισαγωγή στη πληροφορική και την ανοιχτότητα',
    'Σύγχρονα λειτουργικά συστήματα'
    )
# Άνοιγμα Αρχείου
file = open(FILE_NAME, mode='w', encoding='utf-8')
# Προετοιμασία της επικεφαλίδας του αρχείου (1ης γραμμής)
header = ','.join(COURSE_NAMES)
header = 'Όνομα,' + header + ',ΜέσοςΌρος\n'
file.write(header)

# Επανάλαβε για πάντα
while True:
    student = input('Δώστε το όνομα του σπουδαστή: ')
    if not student:
        file.close()
        # Εμφάνισε πληροφορίες για το αρχείο
        print('Οι βαθμολογίες καταχωρήθηκαν στο αρχείο',file.name)
        print('το οποίο αποθηκεύτηκε στο φάκελο',os.getcwd())
        print('Ευχαριστούμε !!!')
        break # Όταν δε δοθεί όνομα σπουδαστή σταμάτα την επανάληψη
    else:
        print('\tΔώστε τη βαθμολογία του σπουδαστή στα μαθήματα')
        sum_grades = 0 # Το άθροισμα των βαθμολογιών
        line = student # Η εγγραφή για κάθε σπουδαστή
        i = 0 # Μετρητής μαθημάτων 0..4
        for course in COURSE_NAMES:
            while True:
                # Εάν δε δοθεί ακέραιος αριθμός δεν δέχεται τη
                # βαθμολογία για να προχωρήσει
                try:
                    grade = int(input('\t' + COURSE_NAMES[i] + ' :'))
                except:
                    print('Άκυρη Βαθμολογία')
```

```
            else:
                # Μόλις δοθεί ακέραιος βαθμός
                break
        # Έλεγχος εγκυρότητας βαθμού εντός κλίμακας 1..10
        while not (1 <= int(grade) <= 10):
            print('Λάθος βαθμολογία ξαναδώστε βαθμό [1..10]')
            grade = input('\t' + COURSE_NAMES[i] + ' :')
        # Τοποθέτησε στην εγγραφή τους βαθμούς
        line += ',' + str(grade)
        # Αύξησε το άθροισμα για κάθε βαθμό
        sum_grades += int(grade)
        i += 1
    # Υπολόγισε το ΜΟ βαθμολογίας
    avg = sum_grades / len(COURSE_NAMES)
    line += ',' + str(avg) # Πρόσθεσε το ΜΟ στην εγγραφή
    line += '\n'    # Βάλε στη γραμμή το χαρακτήρα αλλαγής
    file.write(line) # Γράψε την γραμμή στο αρχείο
```

Εκτέλεση του προγράμματος:

```
=================== RESTART: C:\Python311\my_scripts\DIEK\kef_08\student_grades_in_file.py ==========
Δώστε το όνομα του σπουδαστή: Ευάγγελος Κανίδης
        Δώστε τη βαθμολογία του σπουδαστή στα μαθήματα
        Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων :αβ
Άκυρη Βαθμολογία
        Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων :12
Λάθος βαθμολογία ξαναδώστε βαθμό [1..10]
        Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων :6
        Σχεδιασμός ανάπτυξη ιστοτόπων και παραγωγή ψηφιακού περιεχομένου :7
        Εισαγωγή στον προγραμματισμό και την αλγοριθμική με τη γλώσσα προγραμματισμού Python :10
        Εισαγωγή στη πληροφορική και την ανοικτότητα :6
        Σύγχρονα λειτουργικά συστήματα :4
Δώστε το όνομα του σπουδαστή: Παναγιώτης Μακρυγιάννης
        Δώστε τη βαθμολογία του σπουδαστή στα μαθήματα
        Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων :6
        Σχεδιασμός ανάπτυξη ιστοτόπων και παραγωγή ψηφιακού περιεχομένου :8
        Εισαγωγή στον προγραμματισμό και την αλγοριθμική με τη γλώσσα προγραμματισμού Python :10
        Εισαγωγή στη πληροφορική και την ανοικτότητα :6
        Σύγχρονα λειτουργικά συστήματα :7
Δώστε το όνομα του σπουδαστή: Θεοδόσιος Τσαπέλας
        Δώστε τη βαθμολογία του σπουδαστή στα μαθήματα
        Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων :
Άκυρη Βαθμολογία
        Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων :8,5
Άκυρη Βαθμολογία
        Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων :8.5
Άκυρη Βαθμολογία
        Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων :8
        Σχεδιασμός ανάπτυξη ιστοτόπων και παραγωγή ψηφιακού περιεχομένου :9
        Εισαγωγή στον προγραμματισμό και την αλγοριθμική με τη γλώσσα προγραμματισμού Python :10
        Εισαγωγή στη πληροφορική και την ανοικτότητα :6
        Σύγχρονα λειτουργικά συστήματα :4
Δώστε το όνομα του σπουδαστή: Αναστάσιος Χατζηπαπαδόπουλος
        Δώστε τη βαθμολογία του σπουδαστή στα μαθήματα
        Υλικό οργάνωση και εγκατάσταση υπολογιστικών συστημάτων :8
        Σχεδιασμός ανάπτυξη ιστοτόπων και παραγωγή ψηφιακού περιεχομένου :9
        Εισαγωγή στον προγραμματισμό και την αλγοριθμική με τη γλώσσα προγραμματισμού Python :7
        Εισαγωγή στη πληροφορική και την ανοικτότητα :8
        Σύγχρονα λειτουργικά συστήματα :9
Δώστε το όνομα του σπουδαστή:
Οι βαθμολογίες καταχωρήθηκαν στο αρχείο OpenSoftwareTechnician1.txt
το οποίο αποθηκεύτηκε στο φάκελο C:\Python311\my_scripts\DIEK\kef_08
Ευχαριστούμε !!!
```

Τα περιεχόμενα του αρχείου για τους 4 σπουδαστές είναι:



8.5 Κωδικοποίηση των αρχείων κειμένου

Ένα σημαντικό θέμα το οποίο πρέπει να ληφθεί υπόψη όταν γίνεται δημιουργία ή ανάγνωση ενός αρχείου κειμένου είναι η κωδικοποίηση των περιεχομένων του. Όταν τα αρχεία κειμένου περιέχουν μόνο λατινικούς χαρακτήρες δε χρειάζεται συνήθως κάποια ιδιαίτερη μέριμνα διότι οι προκαθορισμένες (default) ρυθμίσεις των εντολών διαχείρισης αρχείων της Python είναι αρκετές. Στα αρχεία κειμένου που περιέχουν χαρακτήρες και άλλων γλωσσών (πχ χαρακτήρες της ελληνικής γλώσσας) ή το αρχείο πρόκειται να αποσταλεί για επεξεργασία σε άλλη χώρα (όπου τα ΛΣ των χρηστών της θα έχουν διαφορετικές ρυθμίσεις) χρειάζεται η κατανόηση της κωδικοποίησης των περιεχομένων και των ρυθμίσεων των εντολών που χρησιμοποιούμε.

Οι πιο συνηθισμένες κωδικοποιήσεις αρχείων κειμένου που περιέχουν ελληνικούς χαρακτήρες είναι οι:

- cp1253 : ή αλλιώς windows-1253 κωδικοποίηση γνωστή και ως greek ANSI η οποία χρησιμοποιεί 8-bits και χρησιμοποιείται από τη Microsoft και συνήθως είναι η default ρύθμιση σε ΛΣ windows
- iso 8859-7 : διεθνές πρότυπο κωδικοποίησης με 8-bits σχεδιάστηκε για να καλύπτει την ελληνική γλώσσα συμβατό με το πρότυπο ΕΛΟΤ – 928 και πλην λίγων χαρακτήρων (πχ Ά) κοινή κωδικοποίηση με το cp1253
- utf-8 : Κωδικοποίηση μεταβλητού μήκους από 1 έως 4 bytes. Μπορούν να κωδικοποιηθούν οι χαρακτήρες όλων των υπαρκτών γλωσσών. Για τους χαρακτήρες του κώδικα ASCII χρησιμοποιείται μόνο ένα byte οπότε ως υπερσύνολό του είναι συμβατός με αυτόν και οι 128 πρώτοι χαρακτήρες έχουν την ίδια κωδικοποίηση (πχ οι λατινικοί χαρακτήρες). Είναι η default ρύθμιση για ΛΣ linux.

```
PS C:\Users\Administrator> [System.Text.Encoding]::Default


IsSingleByte      : True
BodyName          : iso-8859-7
EncodingName      : Greek (Windows)
HeaderName        : windows-1253
WebName           : windows-1253
WindowsCodePage   : 1253
IsBrowserDisplay  : True
IsBrowserSave     : True
IsMailNewsDisplay : True
IsMailNewsSave    : True
EncoderFallback   : System.Text.InternalEncoderBestFitFallback
DecoderFallback   : System.Text.InternalDecoderBestFitFallback
IsReadOnly        : True
CodePage          : 1253
```

Εύρεση προκαθορισμένης κωδικοποίησης σε περιβάλλον windows

8.5.1 Κωδικοποίηση αρχείων σε διαφορετικά ΛΣ

Παράδειγμα 8.18

Εκτελέστε από το Python shell σε περιβάλλον Linux τις παρακάτω εντολές. Αυτές δημιουργούν ένα αρχείο κειμένου με όνομα data.txt και αποθηκεύουν τις παρακάτω φράσεις χρησιμοποιώντας λατινικούς

και ελληνικούς χαρακτήρες, στη συνέχεια φορτώνουν ένα νέο άρθρωμα λογισμικού και καλούν μία μέθοδό του:

```
>>> f = open('data.txt', 'w')
>>> f.write('Hello\n')
>>> f.write('Κόσμε')
>>> f.close()
>>> import locale
>>> locale.getlocale()
('el_GR', 'UTF-8')
```

Έπειτα αφού μεταφέρετε το αρχείο `data.txt` *σε κατάλληλη τοποθεσία σε ΗΥ με ΛΣ Windows προσπαθήστε με τις παρακάτω εντολές να διαβάσετε τα περιεχόμενα αυτά.*

```
>>> f = open('data.txt', 'r')
>>> f.read()
>>> f.close()
```

Ερωτήσεις:

1. Πως εξηγούνται τα παρακάτω μηνύματα λάθους
2. Τι δείχνουν οι εντολές `locale.getlocale()`

```
File "<pyshell#20>", line 1, in <module> f.read()
File "C:\Python311\Lib\encodings\cp1253.py", line 23, in decode
    return codecs.charmap_decode(input,self.errors,decoding_table)[0]
UnicodeDecodeError: 'charmap' codec can't decode byte 0x9a in position 7:
character maps to <undefined>
>>> import locale
>>> locale.getlocale()
('Greek_Greece', '1253')
```

Απάντηση :

1. *Η προκαθορισμένη (default) κωδικοποίηση των αρχείων κειμένου σε περιβάλλον Windows είναι* `cp1253` *ενώ σε περιβάλλον Linux είναι* `utf-8`*. Όταν δημιουργείται ένα αντικείμενο διαχείρισης αρχείου κειμένου με την εντολή* `open` *και με την προκαθορισμένη ρύθμιση του ορίσματος* `encoding` *τότε χρησιμοποιείται ως ρύθμιση κωδικοποίησης αυτή του ΛΣ. Έτσι ενώ το αρχείο δημιουργήθηκε για εγγραφή από το ΛΣ Linux με κωδικοποίηση utf-8 η ανάγνωσή του από το ΛΣ Windows γίνεται χρησιμοποιώντας την προκαθορισμένη κωδικοποίηση cp1253. Η ασύμβατη (προς τα κάτω δηλαδή από utf → cp1253) κωδικοποίηση προκαλεί το μήνυμα λάθους που φαίνεται στις εντολές που εκτελέστηκαν.*
2. *Η εντολή* `locale.getlocale()` *είναι διαθέσιμη φορτώνοντας το άρθρωμα λογισμικού* `locale` *και δείχνει σε περιβάλλον python-shell την default κωδικοποίηση του συστήματος.*

Παράδειγμα 8.19

Δοκιμάστε να εκτελέσετε την αντίστροφη διαδικασία, δηλαδή: δημιουργήστε ένα αρχείο χρησιμοποιώντας εντολές Python σε ΛΣ Windows με περιεχόμενα ελληνικούς και λατινικούς

χαρακτήρες και στη συνέχεια δοκιμάστε να το διαβάσετε με εντολές Python σε ΛΣ Linux. Τι παρατηρείτε; Μπορείτε να το αιτιολογήσετε;

8.5.2 (**) Δραστηριότητα, αποθήκευση περιεχομένων αρχείου κειμένου σε διαφορετική κωδικοποίηση

Να υλοποιηθεί μια εφαρμογή η οποία θα ζητά από το χρήστη: α) τη διαδρομή που βρίσκεται κάποιο αρχείο κειμένου, β) το όνομα του αρχείου γ) την κωδικοποίησή του περιεχομένου του σε 3 διαθέσιμες επιλογές (`cp1253, iso 8859-7, utf-8`) και στη συνέχεια εάν το αρχείο υπάρχει θα το ανοίγει θα διαβάζει τα περιεχόμενά του και θα ζητά το νέο τύπο κωδικοποίησης του περιεχομένου του αρχείου στο οποίο ζητείται η αποθήκευσή του. Στη συνέχεια θα αποθηκεύει το αρχείο στη νέα κωδικοποίηση.

Σημείωση

- Η μετατροπή από μία κωδικοποίηση σε μία άλλη δεν είναι ιδιαίτερα απλή και πάντα εφικτή λειτουργία. Εξαρτάται από το πηγαίο περιεχόμενο (πχ ένας χαρακτήρας που υπάρχει στο πηγαίο αρχείο (source) και δεν υφίσταται στον πίνακα κωδικοποίησης του αρχείου προορισμού (target) θα προκαλέσει λάθος.
- Η ενδεικτική λύση δεν έχει ως στόχο τις δεξιότητες στις τεχνικές μετατροπής περιεχομένων που απαιτούν και άλλες γνώσεις αλλά τη χρήση βασικών λειτουργιών των αρχείων (άνοιγμα, κωδικοποίηση, χρήση μενού επιλογών, αποφυγή runtime errors κλπ). Έτσι μετατροπές με τον τρέχον κώδικα σε κάποιες περιπτώσεις (όπως πχ utf (με ελληνικούς χαρακτήρες)→ cp1253 ή iso 8859-7) θα προκαλέσουν λάθη χρόνου εκτέλεσης.

```
import os

# Η συνάρτηση source διαβάζει και επιστρέφει τη διαδρομή(θέση)
# και το όνομα του αρχείου προς μετατροπή
def source():
    print(
        "Δώστε τη διαδρομή (path) που βρίσκεται\n\
το αρχείο που θέλετε να μετατρέψετε σε άλλη κωδικοποίηση"
    )
    print("πχ σε περιβάλλον Linux /home/user/my_files/")
    print("πχ σε περιβάλλον Windows c:\\python311\\my_files\\")
    path = input("Διαδρομή: ")
    file1 = input("Δώστε το όνομα του αρχείου: ")
    return path, file1

# Η συνάρτηση menu εμφανίζει το μενού επιλογών της επιθυμητής κωδικοποίησης
# α) του πηγαίου αρχείου παράμετρος(0) και της κωδικοποίησης του
# αρχείου προορισμού παράμετρος (1) επιστρέφει την επιθυμητή κωδικοποίηση
# ως συμβολοσειρά: 'cp1253', 'iso 8859-7', 'utf-8'
def menu(direction=0):
```

```
    """direction=0 (0:ΑΠΟ, 1:ΣΕ)"""
    if direction == 0:
        d = 'ΑΠΟ'
    else:
        d = 'ΣΕ'
    print('Μετατροπή περιεχομένων',d, 'κωδικοποίηση(Encoding)')
    print('1.cp1253')
    print('2.iso 8859-7')
    print('3.utf-8')
    choice = int(input('Δώστε επιλογή: '))
    while choice not in [1, 2, 3]:
        print('Η επιλογή πρέπει να είναι 1, 2, 3')
        choice = int(input('Δώστε επιλογή: '))
    codes = {1:'cp1253', 2:'iso 8859-7', 3:'utf-8'}
    return codes[choice]

# Η συνάρτηση main δημιουργεί τα αντικείμενα f1 ως πηγαίο αρχείο
# και το αντικείμενο f2 ως το αρχείο προορισμού με τις επιλεχθείσες
# κωδικοποιήσεις, διαβάζει τα περιεχόμενα του αντικειμένου f1
# και τα αντιγράφει στο f2, με όνομα αρχείου file2
# εμφανίζει επίσης και τη διαδρομή του αρχείου προορισμού
def main():

    while True:
        try:
            path, file1 = source()
            code1 = menu(0)
            f1 = open(path+file1, mode='r', encoding=code1)
        except IOError:
            print('!' * 30)
            print('Το αρχείο',file1,'δε βρέθηκε')
            print('!' * 30)
        else:
            try:
                text = f1.read()
            except UnicodeDecodeError:
                print('!' * 30)
                print('Το αρχείο',file1,'δεν έχει τη σωστή κωδικοποίηση')
                print('!' * 30)
            else:
                print('Τα περιεχόμενα του αρχείου',file1,'διαβάστηκαν')
                print(text)
```

```
                f1.close()
                code2 = menu(1)
                file2 = file1[:-4] + '_' + code2 + '.' + file1[-1:-4:-1]
                f2 = open(file2, mode='w', encoding=code2)
                try:
                    f2.write(text)
                except:
                    print('Ανέφικτη η μετατροπή')
                    print('Δημιουργήθηκε το',file2,'χωρίς περιεχόμενα')
                else:
                    print('Δημιουργήθηκε το',file2,'κωδικοποίηση',code2)
                    print('Η μετατροπή της κωδικοποίησης ήταν επιτυχής')
                finally:
                    print('Το αρχείο βρίσκεται στη διαδρομή', os.getcwd())
                    f2.close()
                break

main()
```

Ως αρχεία δοκιμής ανάλογα με το ΛΣ που χρησιμοποιείτε μπορείτε να δημιουργήσετε με τον ενσωματωμένο συντάκτη κειμένου του ΛΣ (βλ παρακάτω Συντάκτες Κειμένου & Κωδικοποιήσεις), δύο αρχεία με περιεχόμενο σε λατινικούς χαρακτήρες και διαφορετικές κωδικοποιήσεις και 2 αρχεία με περιεχόμενο σε ελληνικούς χαρακτήρες και διαφορετικές κωδικοποιήσεις.

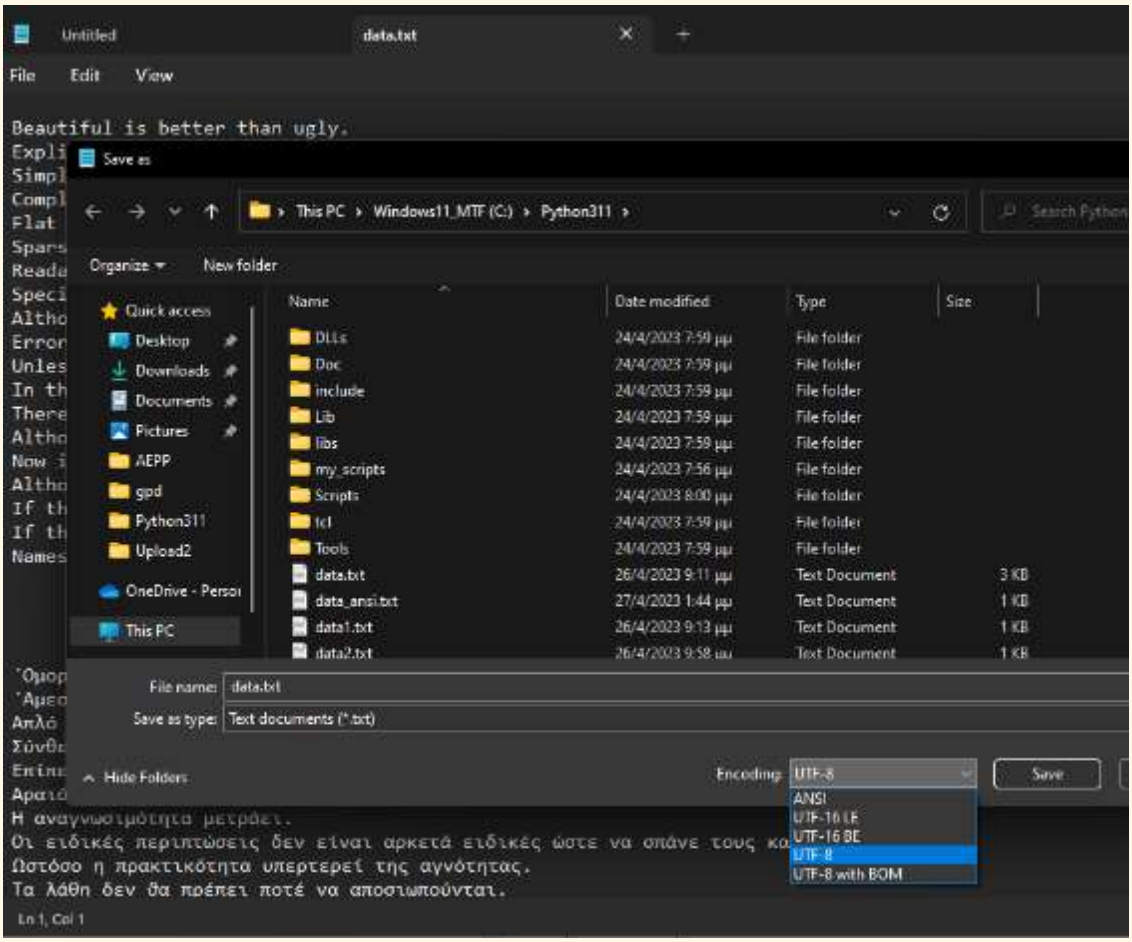
8.5.3 Συντάκτες Κειμένου & Κωδικοποιήσεις

Από τους απλούς συντάκτες κειμένου είναι δυνατόν να επιλεχθεί ο τρόπος κωδικοποίησης ενός αρχείου. Αυτό γίνεται από τις επιλογές:

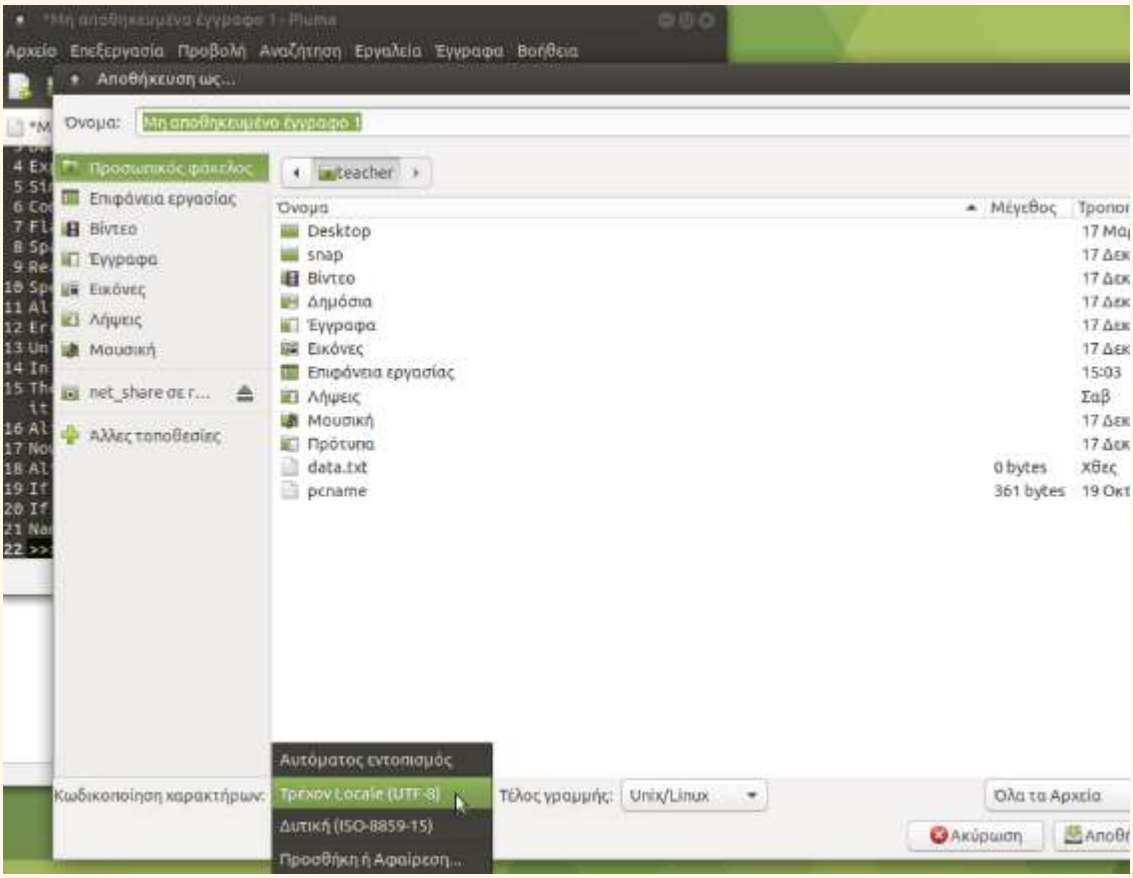
- File/Save As → Encoding (Windows/Notepad) ή
- Αρχείο/Αποθήκευση Ως → Κωδικοποίηση Χαρακτήρων (Ubuntu(Mate)/ Pluma)

Όταν δημιουργείται ένα αρχείο από περιβάλλον Linux μέσω του συντάκτη υπάρχει η δυνατότητα και της επιλογής του τέλους γραμμής (Line Feed) ώστε να είναι συμβατή με το περιβάλλον Windows

Windows - Συντάκτης Κειμένου notepad



Ubuntu Linux (Mate) Συντάκτης Κειμένου Pluma



8.6 Άρθρωμα λογισμικού os

Το module `os` παρέχει δυνατότητες ώστε μέσα από το σενάριο σε γλώσσα Python να δίνονται εντολές που διαχειρίζονται αρχεία και καταλόγους του ΛΣ. Σε προηγούμενη παράγραφο αναφέρθηκε ότι αφού φορτωθεί το άρθρωμα `os` (`import os`) η εντολή: `os.getcwd()` επιστρέφει τον τρέχοντα κατάλογο εργασίας. Ακολουθούν οι πιο συχνές εντολές (μέθοδοι) του αρθρώματος `os`, η λειτουργία τους και παραδείγματα εκτέλεσης.

Εντολή	Λειτουργία	Παράδειγμα

`os.getcwd()`	Εμφάνιση τρέχοντος καταλόγου εργασίας	`>>> os.getcwd()`
		`'c:\\python311'`
		`'/home/teacher'`

`os.chdir(path)`	Αλλαγή τρέχοντος καταλόγου στο συγκεκριμένο path	`>>> os.chdir('c:\\python27')` `>>> os.chdir('/home/teacher/Έγγραφα')`

`os.chdir('../ ')`	Μετάβαση στον προηγούμενο κατάλογο	`>>> os.getcwd()`
		`'/home/teacher/Έγγραφα'`
		`>>> os.chdir('../ ')`
		`>>> os.getcwd()`
		`'/home/teacher'`

`os.mkdir(path)`	Δημιουργία τελικού καταλόγου (my_files) στο συγκεκριμένο path	`>>> os.chdir('c:\\python311\my_files)` `>>> os.chdir('/home/teacher/Έγγραφα/my_files')`
`os.mkdir(folder)`	Δημιουργία καταλόγου (my_files) στον τρέχοντα κατάλογο	`>>> os.chdir('my_files')`
`os.rename(name1, name2)`	Μετονομασία αρχείου name1 σε name2	`>>> os.rename('old_name.txt', 'new_name.txt')`
`os.remove(file)`	Διαγραφή του αρχείου (old_data.txt)	`>>> os.remove('old_data.txt')`
`os.listdir(path)`	Επιστρέφει τα περιεχόμενα του τελικού καταλόγου σε μία λίστα	`>>> os.listdir('c:\\python311\\os_module')` `['data1.txt', 'data2.txt', 'data_ansi.txt']`

8.6.1 Δραστηριότητα - Εφαρμογές OS

α. Να γίνει συνάρτηση η οποία θα δέχεται ένα path και θα διαγράφει όλα τα αρχεία .txt που βρίσκονται στον τελικό φάκελο

β. Να γίνει συνάρτηση η οποία θα δέχεται ένα path και ένα string και θα εμφανίζει όλα τα ονόματα των αρχείων κειμένου στον τελικό φάκελο του path τα οποία περιέχουν αυτό το string με φθίνουσα σειρά εμφάνισης (πρώτα το όνομα του αρχείου που περιέχει περισσότερες φορές το string).

8.7 Επιλυμένη - Κλιμακωτή Εφαρμογή

Στην εφαρμογή που ακολουθεί θα υλοποιηθούν κλιμακωτά ορισμένες συναρτήσεις οι οποίες υλοποιούν διάφορες λειτουργίες με αρχεία. Αυτές στη συνέχεια θα χρησιμοποιηθούν στη λύση ενός πιο σύνθετου προβλήματος.

1. Να γίνει συνάρτηση η οποία θα δέχεται ως όρισμα μία διαδρομή (path) και θα επιστρέφει τα περιεχόμενα του τελικού φακέλου ως μία λίστα αρχείων.

```
def create_file_list_from(path):
    """Επιστρέφει λίστα με τα όνοματα των αρχείων του path"""
    file_list = os.listdir()
    return file_list
```

2. Να γίνει συνάρτηση η οποία θα δέχεται το όνομα ενός αρχείου και θα εμφανίζει όλα τα περιεχόμενά του.

```
def show_contents_of_file(file):
    """Εμφανίζει τα περιεχόμενα του αρχείου κειμένου"""
    file_obj = open(file, "r")
    text = file_obj.read()
    print(text)
    file_obj.close()
```

3. Έστω ότι έχουμε μία συμβολοσειρά της μορφής: Ονοματεπώνυμο <**όνομα**>,<**τιμή**><**αλλαγή γραμμής**>. Δηλαδή θα μπορούσε να είναι μία γραμμή από ένα αρχείο κειμένου πχ:

   ```
   George Anagnostopoulos,456'\n'
   ```

   και ζητείται:

   α. Συνάρτηση η οποία θα δέχεται τη γραμμή και θα επιστρέφει το ονοματεπώνυμο (δεν γνωρίζουμε το μήκος του ονόματος)

   β. Συνάρτηση η οποία θα δέχεται τη γραμμή και θα επιστρέφει την τιμή ως αριθμητική ποσότητα (δεν γνωρίζουμε το πλήθος των ψηφίων της τιμής)

```
def get_name_from_line(line):
    """Επιστρέφει το όνομα του υπαλλήλου από ένα string της μορφής
    <ονομα><διαχωριστικό: sep><...>"""
    pos = line.find(sep)
    return line[:pos]
```

```
def get_ammount_from_line(line):
    """Επιστρέφει τις πωλήσεις του υπαλλήλου από ένα string της μορφής
    <ονομα><διαχωριστικό: sep><πωλήσεις>'\n'"""
    pos = line.find(sep)
    num = line[pos + 1 :].strip()
    return float(num)
```

4. Να γίνει συνάρτηση η οποία θα δέχεται μία λίστα αρχείων και θα διαγράφει από τον τρέχοντα κατάλογο εργασίας όλα τα αρχεία της λίστας το όνομα των οποίων αρχίζει από τη λέξη 'day' εμφανίζοντας και τα κατάλληλα μηνύματα.

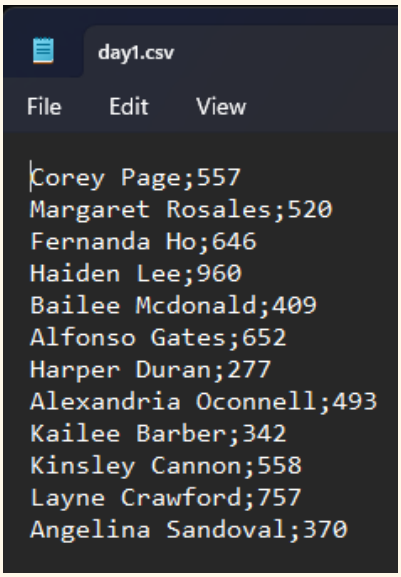
```
def delete_files(file_list):
    """Εντοπισμός των αρχικών αρχείων δεδομένων
    και διαγραφή τους από το path"""
    for file in file_list:
        # Αν το όνομα του αρχείου αρχίζει από day διέγραψέ το
        if file[:3] == "day":
            os.remove(file)
            print(file, "deleted")
```

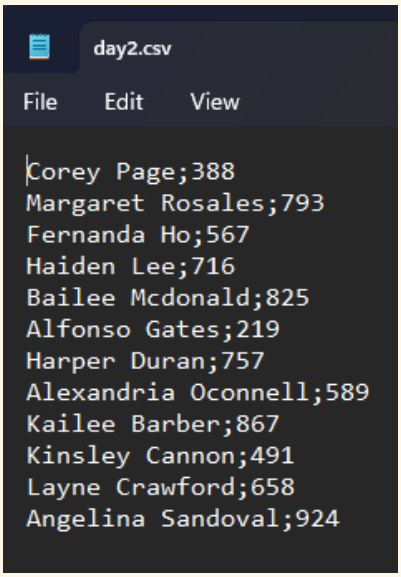
5. Ας υποθέσουμε τώρα ότι έχουμε ένα αρχείο με πολλές τέτοιες γραμμές <**όνομα**>,<**τιμή**><**αλλαγή γραμμής**> όπου κάθε γραμμή έχει και ένα διαφορετικό όνομα. Ζητείται να κάνετε μία συνάρτηση η οποία χρησιμοποιώντας τη συνάρτηση 3α θα δέχεται το όνομα ενός αρχείου και θα επιστρέφει μία λίστα με όλα τα ονόματα που περιέχει.

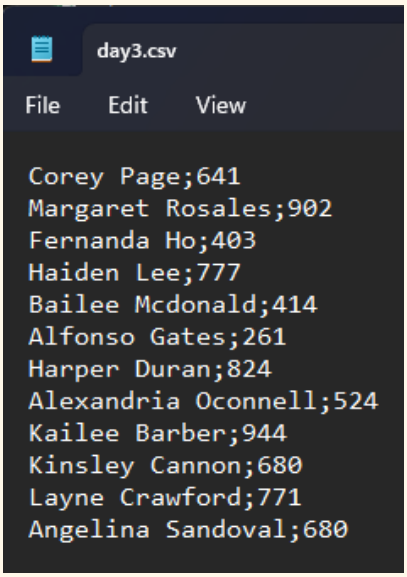
```
def get_names_list_from_file(file):
    """Επιστρέφει λίστα με τα ονόματα των υπαλλήλων που βρίσκονται
    σε ένα αρχείο της μορφής:
    Corey Page;557
    Margaret Rosales;520
    Fernanda Ho;646
    Haiden Lee;960
    ..............."""
    file_obj = open(file, "r")  # Άνοιξε το αρχείο για ανάγνωση
    names = []  # Δημιούργησε μία κενή λίστα για τα ονόματα πωλητών
    # Για κάθε γραμμή του αρχείου (που αντιστοιχεί σε κάθε πωλητή)
    for line in file_obj:
        # Διαχώρισε το όνομα από τη γραμμή
        name = get_name_from_line(line) # Η συνάρτηση του Ερωτήματος 3α
        names.append(name)  # Βάλε το όνομα στη λίστα
    file_obj.close()  # Κλείσε το αρχείο
    return names  # Επέστρεψε τη λίστα με τα ονόματα
```

Πρόβλημα

Μία εταιρεία με γραφεία στην ΠΟΛΗ1 απασχολεί κάποιους πωλητές. Για κάθε ημέρα λειτουργίας της εταιρείας καταγράφονται σε ένα αρχείο (όχι από την εφαρμογή που θέλουμε να υλοποιήσουμε) το οποίο αποθηκεύεται σε κάποιο φάκελο στο δίσκο τα ονόματα των πωλητών της ΠΟΛΗΣ1 και οι πωλήσεις τους για αυτήν την ημέρα, την επόμενη καταγράφονται τα ονόματα των πωλητών και οι πωλήσεις τους σε ένα 2ο αρχείο κ.ο.κ. Η σειρά καταγραφής των ονομάτων των πωλητών παραμένει η ίδια σε όλα τα αρχεία. Τα ονόματα των πωλητών και οι πωλήσεις σε ευρώ χωρίζονται από κάποιο διαχωριστικό χαρακτήρα. Τα αρχεία δηλαδή έχουν την παρακάτω μορφή:







Παρατηρήστε ότι όλα αφορούν στους ίδιους πωλητές και κάθε μέρα αποθηκεύεται ένα αρχείο με τις πωλήσεις του καθενός (έχουν όλα το ίδιο μοτίβο περιεχομένου και δεν απολύονται, ούτε συνταξιοδοτούνται, ούτε παραιτούνται, ούτε προσλαμβάνονται πωλητές).

Ζητείται να δίνεται ως είσοδο σε ένα πρόγραμμα τη διαδρομή του φακέλου που βρίσκονται τα αρχεία αυτά και στη συνέχεια:

α. να εμφανίζεται στην οθόνη όλα τα περιεχόμενα όλων των αρχείων
β. να εμφανίζονται τα ονόματα των πωλητών και οι συνολικές πωλήσεις του καθενός
γ. να υπολογίζονται και να εμφανίζονται οι συνολικές εισπράξεις από όλους τους πωλητές μαζί
δ. τα αποτελέσματα του ερωτήματος γ να αποθηκεύονται σε αρχείο με το όνομα total_sales.txt
ε. να διαγράφονται από τον αρχικό φάκελο όλα τα αρχεία εκτός από το αρχείο total_sales.txt

Σημειώσεις

α. Είναι άγνωστο το πλήθος των αρχείων του φακέλου
β. Είναι άγνωστα τα ονόματα των αρχείων που περιέχει, αλλά περιέχει μόνο τα αρχεία days1.txt, days2.txt, ..
γ. Είναι άγνωστα τα ονόματα των πωλητών
δ. Είναι άγνωστο το πλήθος των πωλητών
ε. Είναι γνωστό ότι όλα τα αρχεία περιέχουν τους ΙΔΙΟΥΣ πωλητές και το ΙΔΙΟ μοτίβο

στ. Θα πρέπει η εφαρμογή να λειτουργεί χωρίς αλλαγές (εκτός από τη διαδρομή του φακέλου που αποτελεί είσοδο) και για την ΠΟΛΗ2, ΠΟΛΗ3 κ.ο.κ. όπου η εταιρεία έχει άλλους πωλητές (άλλα ονόματα & άλλο πλήθος πωλητών).

Ακολουθεί η πλήρης (ενδεικτική λύση) όχι με απόλυτα αμυντικό προγραμματισμό προκειμένου να μην αυξηθεί η δυσκολία του κώδικα, τα δοκιμαστικά αρχεία (day1.txt, day2.txt, …, day7.txt) θα τα βρείτε στο σύνδεσμο. Πρέπει να τα κατεβάσετε και να τα αντιγράψετε σε φάκελο εργασίας στον ΗΥ σας, πχ στην ενδεικτική λύση στις πρώτες γραμμές ο φάκελος βρίσκεται στη διαδρομή c:\python311\sales.

Ενδεικτική Λύση

Οι συναρτήσεις σε γκρι φόντο στη λύση έχουν ήδη υλοποιηθεί στα ερωτήματα (1 – 4).

```
import os  # Φόρτωσε το άρθρωμα διαχείρισης του ΛΣ

##path = 'c:\\python311\\sales\\'
##sep = ';'
while True:
    try:
        path = input("Δώσε τη διαδρομή με τα αρχεία της εφαρμογής: ")
        os.chdir(path)  # άλλαξε τον τρέχοντα κατάλογο στο path
    except FileNotFoundError:
        print(" Η διαδρομή", path, "δεν βρέθηκε")
    else:
        print("Η διαδρομή", path, "βρέθηκε")
        sep = input("Δώσε το διαχωριστικό των αρχείων δεδομένων: ")
        break


def create_file_list_from(path):
    """Επιστρέφει λίστα με τα ονόματα των αρχείων του path"""
    file_list = os.listdir()
    return file_list


def show_contents_of_file(file):
    """Εμφανίζει τα περιεχόμενα του αρχείου κειμένου"""
    file_obj = open(file, "r")
    text = file_obj.read()
    print(text)
    file_obj.close()


def get_name_from_line(line):
    """Επιστρέφει το όνομα του υπαλλήλου από ένα string της μορφής
    <ονομα><διαχωριστικό: sep><...>"""
    pos = line.find(sep)
    return line[:pos]


def get_ammount_from_line(line):
    """Επιστρέφει τις πωλήσεις του υπαλλήλου από ένα string της μορφής
    <ονομα><διαχωριστικό: sep><πωλήσεις>'\n'"""
```

```
    pos = line.find(sep)
    num = line[pos + 1 :].strip()
    return float(num)
```

```
def get_names_list_from_file(file):
    """Επιστρέφει λίστα με τα ονόματα των υπαλλήλων που βρίσκονται
    σε ένα αρχείο της μορφής:
    Corey Page;557
    Margaret Rosales;520
    Fernanda Ho;646
    Haiden Lee;960
    ..............."""
    file_obj = open(file, "r")  # Άνοιξε το αρχείο για ανάγνωση
    names = []  # Δημιουργησε μία κενή λίστα για τα ονόματα πωλητών
    # Για κάθε γραμμή του αρχείου (που αντιστοιχεί σε κάθε πωλητή)
    for line in file_obj:
        # Διαχώρισε το όνομα από τη γραμμή
        name = get_name_from_line(line)
        names.append(name)  # Βάλε το όνομα στη λίστα
    file_obj.close()  # Κλείσε το αρχείο
    return names  # Επέστρεψε τη λίστα με τα ονόματα
```

```
def get_total_sales_from_file_list(file_list):
    """Υπολογίζει και επιστρέφει λίστα με τις συνολικές πωλήσεις
    κάθε πωλητή που βρίσκεται στα αρχεία πωλήσεων"""
    # Αρχικοποίησε τη λίστα συνολικών πωλήσεων με 0
    sales_list = [0 for i in range(12)]
    n = len(file_list)  # Το μήκος της λίστας
    for i in range(n):  # Για κάθε αρχείο της λίστας αρχείων
        # Φτιάξε ένα αντικείμενο για κάθε αρχείο στη λίστα
        # και άνοιξέ το για διάβασμα
        file_obj = open(file_list[i], "r")
        line_num = 0  # Δείκτης εντοπισμού γραμμών (άρα και πωλητών)
        for line in file_obj:  # Για κάθε γραμμή του αρχείου
            # η οποία αντιστοιχεί σε κάποιον πωλητή
            # Αύξησε τον αντίστοιχο μετρητή συνολικών πωλήσεων
            # κατά την τιμή που δείχνει τις πωλήσεις του πωλητή
            sales_list[line_num] += get_ammount_from_line(line)
            line_num += 1  # Πήγαινε στον επόμενο πωλητή
        file_obj.close()  # Κλείσε το αρχείο
```

```
    return sales_list  # Επέστρεψε τη λίστα των συνολικών πωλήσεων


def export_results(names, sales):
    """Εξαγωγή των αποτελεσμάτων των λιστών των ονομάτων των
    πωλητών και των συνολικών πωλήσεων στην οθόνη και επιστροφή
    των αποτελεσμάτων και σε συμβολοσειρά"""
    total = 0  # Ποσό συνολικών πωλήσεων από όλους τους πωλητές
    text = ""  # Τα αποτελέσματα στην οθόνη ως συμβολοσειρά
    for i in range(len(names)):  # Γιά κάθε πωλητή
        total += sales[i]  # Άθροισε τις πωλήσεις του
        # Γραμμή εμφάνισης (στοιχισμένη) όνομα -- πωλήσεις
        out = names[i].ljust(40) + str(sales[i]).rjust(10)
        print(out)
        # Πρόσθεσε τη γραμμή εμφάνισης στο string επιστροφής
        text += out + "\n"
    out = (40 + 10) * "-"  # Φτιάξε μία οριζόντια γραμμή -
    print(out)
    # Πρόσθεσε τη γραμμή εμφάνισης στο string επιστροφής
    text += out + "\n"
    # Γραμμή εμφάνισης (στοιχισμένη) Συνολικές -- πωλήσεις
    out = "Total".ljust(40) + str(total).rjust(10)
    print(out)
    text += out + "\n"
    # Επέστρεψε ότι εμφανίστηκε στην οθόνη ως string
    return text


def delete_files(file_list):
    """Εντοπισμός των αρχικών αρχείων δεδομένων
    και διαγραφή τους από το path"""
    for file in file_list:
        # Αν το όνομα του αρχείου αρχίζει από day διέγραψέ το
        if file[:3] == "day":
            os.remove(file)
            print(file, "deleted")


def main():
    """Κύρια συνάρτηση κλήσης των υπολοίπων"""
    # Δημιούργησε τη λίστα αρχείων
    file_list = create_file_list_from(path)
```

```
    for file in file_list:  # Για κάθε αρχείο στη λίστα
        show_contents_of_file(file)  # Εμφάνισε τα περιεχόμενα στην οθόνη
    # Φτιάξε την λίστα με τα ονόματα των πωλητών από το 1ο αρχείο
    names = get_names_list_from_file(file_list[0])
    # Φτιάξε για κάθε πωλητή τη λίστα των συνολικών του πωλήσεων
    sales = get_total_sales_from_file_list(file_list)
    # Εμφάνισε τα δεδομένα και τα αποτελέσματα στην οθόνη
    text = export_results(names, sales)
    # Δημιουργησε αρχείο με τις συνολικές πωλήσεις για κάθε πωλητή
    file = open("total_sales.txt", "w")
    file.write(text)
    file.close()
    # Διαγραφή αρχικών αρχείων
    a = input("Θέλετε να διαγράψετε τα αρχικά αρχεία δεδομένων N/O: ")
    if a.lower() in "vNnN":
        delete_files(file_list)
    else:
        print("Τα αρχεία δεν διαγράφηκαν")


if __name__ == "__main__":
    main()
```

8.8 Να θυμάμαι

Λάθη στον κώδικα

Τα προγράμματα που υλοποιούνται από έναν προγραμματιστή ειδικά στις πρώτες εκδόσεις τους σπάνια λειτουργούν απολύτως σωστά. Μεγάλοι τίτλοι λογισμικού περνούν από πολλές εκδόσεις (versions), διορθώσεις (patches), ενημερώσεις (updates), προκειμένου να αποκτήσουν σχετικά αξιόπιστη λειτουργία.

Εκσφαλμάτωση

Πλήθος λαθών (συντακτικών, χρόνου εκτέλεσης ή λογικών) διορθώνονται στις διαδικασίες ελέγχου και δοκιμών αλλά και σε όλη τη διάρκεια ζωής ενός λογισμικού. Η διαδικασία αυτή, δηλαδή του ελέγχου, εντοπισμού και διόρθωσης των λαθών σε ένα πρόγραμμα ονομάζεται εκσφαλμάτωση ή αποσφαλμάτωση (debugging).

Κατηγορίες Λαθών

Τα συντακτικά λάθη (syntax errors), τα οποία εμφανίζονται όταν παραβιάζονται οι κανόνες σύνταξης των εντολών της γλώσσας. Είναι τα πιο συχνά σφάλματα αλλά και τα πιο εύκολα να διορθωθούν αφού εντοπίζονται από το μεταγλωττιστή ή το διερμηνέα ο οποίος εμφανίζει προειδοποιητικά μηνύματα προς τον προγραμματιστή. Το πρόγραμμα δεν εκτελείται εάν δε διορθωθούν όλα τα συντακτικά του λάθη. Παραδείγματα τέτοιων σφαλμάτων προκύπτουν: σε λανθασμένη σύνταξη εντολών, στη στοίχιση του κώδικα, σε λανθασμένη χρήση παρενθέσεων στις εκφράσεις υπολογισμού, ορθογραφικά (στις δεσμευμένες λέξεις) κ.ά.

Τα λάθη χρόνου εκτέλεσης (run time errors) εμφανίζονται κατά την εκτέλεση του προγράμματος και προκαλούν τη βίαιη διακοπή του (crash). Αυτά δεν εντοπίζονται από το περιβάλλον της γλώσσας που χρησιμοποιούμε (διερμηνέα ή μεταγλωττιστή) και είναι πιο δύσκολο να αντιμετωπιστούν από ότι τα συντακτικά λάθη διότι πολλές φορές οφείλονται σε καταστάσεις που δεν είναι εύκολο να προβλεφθούν. Αυτά τα σφάλματα μπορεί να συμβούν: κατά την εισαγωγή ασύμβατων δεδομένων από το χρήστη, σε μη έγκυρες μαθηματικές πράξεις (διαίρεση με το 0), αστοχίες στο υλικό (σφάλματα στην επιφάνεια ανάγνωσης ενός δίσκου) κ.ά. Οι γλώσσες προγραμματισμού διαθέτουν εντολές διαχείρισης που «παγιδεύουν τα λάθη αυτά» αποφεύγοντας έτσι και την ξαφνική (και εκνευριστική) διακοπή του προγράμματος. Τέτοιες εντολές θα δούμε στη συνέχεια του κεφαλαίου.

Τα λογικά λάθη (logical errors) είναι λάθη σχεδιασμού στο πρόγραμμα, δεν προκαλούν αντίδραση από το περιβάλλον της γλώσσας ή κάποια ξαφνική διακοπή του προγράμματος, αλλά εμφανίζουν λάθος αποτελέσματα. Κάποιες φορές είναι εύκολο να τα αντιληφθούμε (πχ αναμένουμε μία θετική τιμή και λαμβάνουμε ως αποτέλεσμα μία αρνητική τιμή), άλλες φορές όμως είναι πολύ δύσκολο ή και αδύνατο να τα αντιληφθούμε απλά βλέποντας ένα αποτέλεσμα (πχ ένα λάθος σε μία υπολογιστική διαδικασία εντοπισμού ενός μαθηματικού ορίου, υπολογισμός ενός μεγάλου αθροίσματος). Γι' αυτά υπάρχουν εργαλεία από το προγραμματιστικό περιβάλλον που βοηθούν τον προγραμματιστή στην ανεύρεσή τους και ονομάζονται debuggers καθώς και διάφορες τεχνικές ελέγχου ορθότητας των αποτελεσμάτων.

Μηνύματα λαθών στην Python

Όταν συμβαίνει κάποιο λάθος στον κώδικα και δεν είναι δυνατή η εκτέλεσή του, η γλώσσα Python προκαλεί/ υψώνει (raise) μία εξαίρεση (exception) ως ένδειξη ότι παραβιάζεται κάποιος κανόνας σύνταξης ή μαθηματικού περιορισμού ή περιορισμού της λύσης του προβλήματος

Αμυντικός προγραμματισμός

(Defensive programming) καλείται η προγραμματιστική τεχνική κατά την οποία ο προγραμματιστής ενσωματώνει στο πρόγραμμα εντολές ή και ολόκληρα τμήματα κώδικα μέσω των οποίων προσπαθεί να διασφαλίσει ότι ΜΗ έγκυρες τιμές εισόδου δεδομένων από το χρήστη δε θα προκαλέσουν απροσδόκητες καταστάσεις όπως σφάλματα χρόνου εκτέλεσης ή λογικά σφάλματα.

Εντολές παγίδευσης σφαλμάτων

Οι εντολές παγίδευσης μπορούν να έχουν περισσότερες από μία εντολές `except` ελέγχοντας ταυτόχρονα και το είδος λάθους (εξαίρεσης), όπως επίσης και την (προαιρετική) εντολή `else` όπου η ενότητα των εντολών που περιέχει εκτελείται όταν δεν υπάρχει `exception error` και την (προαιρετική) εντολή `finally` με την ενότητα των εντολών που περιέχει να εκτελείται σε κάθε περίπτωση.

```
try:
    <Εντολές κώδικα προς εκτέλεση>
except <Είδος Λάθους>:
    <Εντολές Διαχείρισης Λάθους>
except <Είδος Λάθους>:
    <Εντολές Διαχείρισης Λάθους>
else:
    <Εντολές Εάν δεν υπάρχει λάθος>
finally:
    <Εντολές που εκτελούνται σε κάθε περίπτωση>
```

Κύρια και δευτερεύουσα μνήμη

Το χαρακτηριστικό της κύριας μνήμης που αφορά στην προσωρινή διατήρηση των δεδομένων αλλά και αυτό του περιορισμένου μεγέθους της (λόγω του υψηλού κόστους της) την καθιστούν ακατάλληλη για χρήση από απαιτητικές εφαρμογές (εμπορικές ή επιστημονικές) με μεγάλο όγκο δεδομένων που έχουν ανάγκες μόνιμης αποθήκευσής τους.

Έτσι για τις ανάγκες αυτές (μεγάλου όγκου δεδομένων και μόνιμης αποθήκευσης δεδομένων) χρησιμοποιείται η δευτερεύουσα μνήμη η οποία κατά κύριο λόγο στις εφαρμογές μας αποτελείται από μονάδες σκληρών δίσκων (HDD, SSD) οι οποίες είναι μεν πιο αργές από την

κύρια μνήμη αλλά καλύπτουν τις αυξημένες απαιτήσεις της χωρητικότητας και της μόνιμης αποθήκευσης.

Αρχεία

Εκτός από τα αρχεία τα οποία περιέχουν τον κώδικα της εφαρμογής μας σε γλώσσα Python (`.py`) μία εφαρμογή μπορεί να συνοδεύεται και από άλλα αρχεία που περιέχουν δεδομένα εισόδου ή αποτελέσματα εξόδου. Τα αρχεία αυτά των δεδομένων στη δευτερεύουσα μνήμη αποθηκεύονται με τη μορφή **αρχείων κειμένου** (text files) ή **δυαδικών αρχείων** (binary files).

Τα αρχεία κειμένου είναι αρχεία που το περιεχόμενό τους αποτελείται από χαρακτήρες κειμένου όπως αυτούς των συμβολοσειρών και μπορεί να αναγνωστεί από τον χρήστη χρησιμοποιώντας και έναν απλό συντάκτη κειμένου. Αυτός είναι και ο πιο κοινός τύπος αρχείων και μπορούν να χρησιμοποιηθούν εύκολα για δεδομένα αντικειμένων των βασικών τύπων `int`, `float`, `str`, `bool`

Τα δυαδικά αρχεία τα οποία περιέχουν δεδομένα σε δυαδική μορφή μη αναγνώσιμη από το χρήστη μπορούν να χρησιμοποιηθούν για κάθε τύπο αντικειμένων ακόμα και εικόνας, ήχου κλπ.

Διαχείριση αρχείων κειμένου

Για να δημιουργηθεί ένα αντικείμενο διαχείρισης αρχείου κειμένουθα χρησιμοποιηθεί η συνάρτηση `open` η οποία συντάσσεται ως εξής:

```
<όνομα αναφοράς> = open(<διαδρομή και όνομα αρχείου>,
mode=<κατάσταση/mode>, encoding=<κωδικοποίηση>)
```

Κατάσταση mode	Λειτουργία
`mode='r'`	Άνοιγμα αρχείου για ανάγνωση κειμένου
`mode='w'`	Άνοιγμα αρχείου για εγγραφή, εάν δεν υπάρχει το αρχείο δημιουργείται, εάν υπάρχει ήδη, διαγράφονται πρώτα τα παλαιά περιεχόμενά του για να εγγραφούν τα νέα
`mode='a'`	Άνοιγμα νέου αρχείου για εγγραφή ή εάν υπάρχει ήδη, για προσάρτηση νέων περιεχομένων στο τέλος του
`mode='x'`	Άνοιγμα νέου αρχείου για εγγραφή, εάν υπάρχει ήδη επιστρέφεται εξαίρεση `FileExistsError`
`mode='r+'`	Άνοιγμα αρχείου για ανάγνωση και εγγραφή
`mode='w+'`	Άνοιγμα αρχείου για ανάγνωση και εγγραφή, τα προηγούμενα περιεχόμενα διαγράφονται

Κωδικοποιήσεις παράμετρος encoding	Λειτουργία

Εάν παραληφθεί	Χρησιμοποιείται η κωδικοποίηση του ΛΣ.
	Η κωδικοποίηση συστήματος μπορεί να εμφανιστεί φορτώνοντας το module `locale` και καλώντας τη μέθοδο `.getencoding()` ή τη μέθοδο `.getlocale()`
	`import locale`
	`locale.getencoding()`
	`locale.getlocale()`

`encoding='cp1253'`	Χρησιμοποιείται από τη Microsoft και συνήθως είναι η default τιμή σε ΛΣ windows
`encoding='utf-8'`	Κωδικοποίηση μεταβλητού μήκους από 1 έως 4 bytes. Μπορούν να κωδικοποιηθούν οι χαρακτήρες όλων των υπαρκτών γλωσσών. Είναι η προκαθορισμένη (default)τιμή για ΛΣ Linux
`encoding='iso 8859-7'`	Διεθνές πρότυπο κωδικοποίησης με 8-bits σχεδιάστηκε για να καλύπτει την ελληνική γλώσσα.

Μέθοδοι ανάγνωσης περιεχομένων αρχείων

`file.read(size)`: Επιστρέφει ως συμβολοσειρά τα περιεχόμενα μεγέθους size σε bytes του αρχείου

`data = f.read()` : Επιστρέφει σε συμβολοσειρά data όλα τα περιεχόμενα του αρχείου από τη θέση του δείκτη και μετά.

`data = f.read(20)` : Επιστρέφει 20 χαρακτήρες από τη θέση του δείκτη και μετά στη συμβολοσειρά με όνομα data.

`file.seek(position)` :Μετακίνησε το δείκτη στη θέση position.

`file.readlines()` : Η μέθοδος με μία κλήση της επιστρέφει μία λίστα με στοιχεία τις γραμμές του κειμένου

`file.readline()`: Η μέθοδος επιστρέφει με κάθε κλήση της μία γραμμή από το αρχείο κειμένου μετακινώντας ταυτόχρονα και το δείκτη ανάγνωσης του αρχείου

Κωδικοποιήσεις

cp1253 : ή αλλιώς windows-1253 κωδικοποίηση γνωστή και ως greek ANSI η οποία χρησιμοποιεί 8-bits και χρησιμοποιείται από τη Microsoft και συνήθως είναι η default ρύθμιση σε ΛΣ windows

iso 8859-7 : διεθνές πρότυπο κωδικοποίησης με 8-bits σχεδιάστηκε για να καλύπτει την ελληνική γλώσσα συμβατό με το πρότυπο ΕΛΟΤ – 928 και πλην λίγων χαρακτήρων (πχ Ά) κοινή κωδικοποίηση με το cp1253

utf-8 : Κωδικοποίηση μεταβλητού μήκους από 1 έως 4 bytes. Μπορούν να κωδικοποιηθούν οι χαρακτήρες όλων των υπαρκτών γλωσσών. Για τους χαρακτήρες του κώδικα ASCII χρησιμοποιείται μόνο ένα byte οπότε ως υπερσύνολό του είναι συμβατός με αυτόν και οι 128 πρώτοι χαρακτήρες έχουν την ίδια κωδικοποίηση (πχ οι λατινικοί χαρακτήρες). Είναι η default ρύθμιση για ΛΣ linux.

Άρθρωμα λογισμικού os

Το module `os` παρέχει δυνατότητες ώστε μέσα από το σενάριο σε γλώσσα Python να δίνονται εντολές που διαχειρίζονται αρχεία και καταλόγους του ΛΣ. Σε προηγούμενη παράγραφο αναφέρθηκε ότι αφού φορτωθεί το άρθρωμα `os` (`import os`) η εντολή: `os.getcwd()` επιστρέφει τον τρέχοντα κατάλογο εργασίας.

Εντολή	Λειτουργία	Παράδειγμα

`os.getcwd()`	Εμφάνιση τρέχοντος καταλόγου εργασίας	`>>> os.getcwd()`
		`'c:\\python311'`
		`'/home/teacher'`

`os.chdir(path)`	Αλλαγή τρέχοντος καταλόγου στο συγκεκριμένο path	`>>> os.chdir('c:\\python27')` `>>> os.chdir('/home/teacher/Έγγραφα')`

`os.chdir('../')`	Μετάβαση στον προηγούμενο κατάλογο	`>>> os.getcwd()`
		`'/home/teacher/Έγγραφα'`
		`>>> os.chdir('../ ')`
		`>>> os.getcwd()`
		`'/home/teacher'`

`os.mkdir(path)`	Δημιουργία τελικού καταλόγου (my_files) στο συγκεκριμένο path	`>>> os.chdir('c:\\python311\my_files)` `>>> os.chdir('/home/teacher/Έγγραφα/my_files')`
`os.mkdir(folder)`	Δημιουργία καταλόγου (my_files) στον τρέχοντα κατάλογο	`>>> os.chdir('my_files')`
`os.rename(name1, name2)`	Μετονομασία αρχείου name1 σε name2	`>>> os.rename('old_name.txt', 'new_name.txt')`
`os.remove(file)`	Διαγραφή του αρχείου (old_data.txt)	`>>> os.remove('old_data.txt')`
`os.listdir(path)`	Επιστρέφει τα περιεχόμενα του τελικού καταλόγου σε μία λίστα	`>>> os.listdir('c:\\python311\\os_module')` `['data1.txt', 'data2.txt', 'data_ansi.txt']`

8.9 Ερωτήσεις Κατανόησης

1. Οι έμπειροι προγραμματιστές δεν κάνουν ποτέ λάθη στην κωδικοποίηση;

   ΝΑΙ ΟΧΙ

2. Τα συντακτικά λάθη προκαλούν τη βίαιη διακοπή του προγράμματος;

   ΝΑΙ ΟΧΙ

3. Τα Λάθη χρόνου εκτέλεσης διορθώνονται από το διερμηνέα ή τον μεταγλωττιστή;

   ΝΑΙ ΟΧΙ

4. Τα λογικά λάθη είναι τα πιο δύσκολα στην ανεύρεσή τους;

   ΝΑΙ ΟΧΙ

5. Πολλές δοκιμαστικές εκτελέσεις με κατάλληλα επιλεγμένα έγκυρα δεδομένα, όπου συγκρίνονται τα αναμενόμενα με τα παραγόμενα αποτελέσματα του προγράμματος εξασφαλίζουν 100% την απουσία λογικών λαθών;

   ΝΑΙ ΟΧΙ

6. Ο αμυντικός προγραμματισμός χρησιμοποιείται μόνο στα προγράμματα που αφορούν στην εθνική ασφάλεια μίας χώρας;

   ΝΑΙ ΟΧΙ

7. Οι εντολές παγίδευσης λαθών προστατεύουν από λάθη χρόνου εκτέλεσης;

   ΝΑΙ ΟΧΙ

8. Το περιεχόμενων των δυαδικών αρχείων είναι αναγνώσιμο από ένα text editor;

   ΝΑΙ ΟΧΙ

9. Τα ΛΣ windows & Linux έχουν την ίδια default κωδικοποίηση στα αρχεία κειμένου;

   ΝΑΙ ΟΧΙ

10. Η μέθοδος `.close()` όταν τελειώνουμε τις εργασίες σε ένα αρχείο κειμένου είναι προαιρετική;

    ΝΑΙ ΟΧΙ

11. Η μέθοδος `.readline()` επιστρέφει κάθε φορά που καλείται μία γραμμή από το αρχείο μετακινώντας το δείκτη του αρχείου στο τέλος της γραμμής;

    ΝΑΙ ΟΧΙ

12. Η μέθοδος `.seek(0)` μετακινεί το δείκτη στην αρχή του αρχείου;

    ΝΑΙ ΟΧΙ

13. Η μέθοδος `.read()` επιστρέφει σε συμβολοσειρά όλα τα περιεχόμενα του αρχείου από τη θέση του δείκτη και μετά;

ΝΑΙ ΟΧΙ

14. Η μέθοδος `.readlines()`με μία κλήση της επιστρέφει μία πλειάδα με στοιχεία τις γραμμές του κειμένου.;

ΝΑΙ ΟΧΙ

15. Η μέθοδος `.read(1)` επιστρέφει σε συμβολοσειρά όλα τα περιεχόμενα της τρέχουσας γραμμής που βρίσκεται δηλαδή ο δείκτης του αρχείου;

ΝΑΙ ΟΧΙ

8.10 Ερωτήσεις Ανάπτυξης

1. Ποιες οι κατηγορίες σφαλμάτων σε μία γλώσσα προγραμματισμού;
2. Τι γνωρίζετε για τα συντακτικά λάθη σε μία γλώσσα προγραμματισμού;
3. Τι γνωρίζετε για τα λάθη χρόνου εκτέλεσης σε μία γλώσσα προγραμματισμού;
4. Τι γνωρίζετε για τα λογικά σε μία γλώσσα προγραμματισμού;
5. Πως μπορούμε να εντοπίσουμε τα λογικά λάθη σε ένα πρόγραμμα;
6. Τι σημαίνει ο όρος debugging και ποιες διαδικασίες περιλαμβάνει;
7. Τι είναι ο αμυντικός προγραμματισμός;
8. Τι σημαίνει ο όρος παγίδευση λαθών και με ποιες εντολές υλοποιείται στη γλώσσα Python;
9. Ποιες οι διαφορές των αρχείων κειμένου και των δυαδικών αρχείων;
10. Τι ονομάζουμε current working directory στην Python και πως το βρίσκουμε;
11. Τι γνωρίζετε για την κωδικοποίηση αρχείων κειμένου `cp1253`;
12. Τι γνωρίζετε για την κωδικοποίηση αρχείων κειμένου `iso 8859-7`;
13. Τι γνωρίζετε για την κωδικοποίηση αρχείων κειμένου `utf-8`;
14. Ποιες είναι οι default κωδικοποιήσεις σε ΛΣ windows & Linux;

8.11 Ασκήσεις

1. Να υλοποιήσετε συνάρτηση με χαρακτηριστικά αμυντικού προγραμματισμού η οποία να δέχεται και να επιστρέφει μία βαθμολογία [0..100] πανελλαδικών εξετάσεων, εμφανίζοντας ανάλογα μηνύματα σε περίπτωση λανθασμένης εισόδου.
2. Χρησιμοποιώντας την προηγούμενη συνάρτηση, να υλοποιήσετε συνάρτηση η οποία να δέχεται ως είσοδο μία βαθμολογία [0..100] πανελλαδικών εξετάσεων και να επιστρέφει ολογράφως τη βαθμολογία αυτή.
3. Να υλοποιήσετε συνάρτηση με χαρακτηριστικά αμυντικού προγραμματισμού η οποία να δέχεται ως είσοδο μία συμβολοσειρά που να αναπαριστά μία ημερομηνία της μορφής dd/mm/yy και να επιστρέφει ως πλειάδα τον αριθμό της ημέρας, μήνα, έτους, εμφανίζοντας ανάλογα μηνύματα σε περίπτωση λανθασμένης εισόδου.
4. Να υλοποιήσετε ένα πρόγραμμα το οποίο θα διαβάζει και θα αποθηκεύει σε ένα αρχείο κειμένου το όνομα και τους 5 βαθμούς κάθε μαθητή μίας τάξης (η είσοδος σταματά όταν ως όνομα δοθεί το κενό) ώστε κάθε γραμμή στο αρχείο να έχει τη μορφή: `όνομα,β1,β2,β3,β4,β5,ΜΟ`
5. Να υλοποιήσετε ένα πρόγραμμα το οποίο θα διαβάζει το προηγούμενο αρχείο και θα υπολογίζει και θα εμφανίζει το γενικό ΜΟ της τάξης;
6. Να επεκτείνετε τον κώδικα των ασκήσεων 4 και 5 με χαρακτηριστικά αμυντικού προγραμματισμού, εμφανίζοντας ανάλογα μηνύματα σε περίπτωση λανθασμένης εισόδου.
7. Να υλοποιήσετε μία αριθμομηχανή των 4 βασικών πράξεων η οποία να λειτουργεί με μενού επιλογών για τις 4 πράξεις. Επίσης από το μενού επιλογών να διαθέτει επιλογή για εμφάνιση του ιστορικού των πράξεων που έγιναν χρησιμοποιώντας για την υλοποίηση αυτή ένα αρχείο κειμένου. Επίσης στο μενού επιλογών να υπάρχει η δυνατότητα και η δυνατότητα διαγραφής του ιστορικού των πράξεων. πχ εμφάνισης ιστορικού πράξεων:
    a. 2 + 4 = 6
    b. 8 – 5 = 3
    c. 4 * 8 = 32
    d. 12 / 4 = 3
8. Να υλοποιήσετε ένα πρόγραμμα στο οποίο θα δίνετε τη διαδρομή (path) στο δίσκο και θα δημιουργεί ένα αρχείο κειμένου όπου στην 1η γραμμή θα υπάρχει η πλήρης διαδρομή και σε κάθε επόμενη γραμμή το όνομα κάθε αρχείου που βρέθηκε στον τελικό κατάλογο της διαδρομής αυτής.
9. Να υλοποιήσετε ένα πρόγραμμα στο οποίο θα δίνετε τη διαδρομή (path) στο δίσκο και θα δημιουργεί ένα αρχείο κειμένου όπου στην 1η γραμμή θα υπάρχει η διαδρομή του 1ου στη σειρά καταλόγου και σε κάθε επόμενη γραμμή το όνομα κάθε αρχείου που βρέθηκε στον 1ο κατάλογο της διαδρομής αυτής, στη συνέχεια μία κενή γραμμή και η διαδρομή του 2ου στη σειρά καταλόγου και σε κάθε επόμενη γραμμή το όνομα κάθε αρχείου που βρέθηκε στον 2ο κατάλογο της διαδρομής αυτής, αυτό θα συνεχίζεται μέχρι τον τελικό κατάλογο της αρχικής διαδρομής.

Κεφάλαιο 9

9. ΒΙΒΛΙΟΓΡΑΦΙΑ - ΙΣΤΟΓΡΑΦΙΑ

9.1 Βιβλιογραφία

- Υπολογισμοί και προγραμματισμός με την Python, John V. Guttag
- Προγραμματισμός Υπολογιστών ΕΠΑΛ, Αράπογλου Α, et al.
- Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον, Βακάλη Α. et al.
- Εισαγωγή στον Αντικειμενοστραφή Προγραμματισμό με Python, Μαγκούτης Κ, et al.
- Οδηγός Python Μέσω Παραδειγμάτων, Λεβεντέας Δ., TasPython
- Data Structures & Algorithms in Python, Goodrich et al.
- An Introduction to Programming Using Python, Schneider D.
- Python Crash Course, Mathes E.
- Fundamentals of Python Data Structures, Lambert K.
- Introduction to Programming Using Python, Liang D.
- Think Python, Downey A.

9.2 Ιστογραφία

- https://www.python.org/
- https://www.geeksforgeeks.org/python-programming-language/
- Python Tutorials – Real Python
- https://www.programiz.com/python-programming
- Python Tutorial (w3schools.com)
- κύριε δεν έχω internet - Αναστάσιος Χατζηπαπαδόπουλος (sch.gr)
- Learn Python - Free Interactive Python Tutorial
- Python Tutorial (w3schools.com)
- Python Tutorial (tutorialspoint.com)