

**Υπουργείο Παιδείας, Έρευνας και
Θρησκευμάτων
Ινστιτούτο Εκπαιδευτικής Πολιτικής**

**Αράπογλου Α., Βραχνός Ε., Λέκκα Δ., Κανίδης Ε., Μακρυγιάννης Π.,
Μπελεσιώτης Β., Τζήμας Δ., Παπαδάκης Σπ.**

**Προγραμματισμός Υπολογιστών
Γ' Τάξη ΕΠΑ.Λ.**

Διδακτικό υλικό Μαθητή

**Γ' και Δ' τάξη ημερησίων και εσπερινών ΕΠΑ.Λ., του Τομέα
Πληροφορικής**

**Ινστιτούτο Τεχνολογίας Υπολογιστών & Εκδόσεων
«Διόφαντος»**

**Υπουργείο Παιδείας, Έρευνας και
Θρησκευμάτων
Ινστιτούτο Εκπαιδευτικής Πολιτικής**

**Αράπογλου Α., Βραχνός Ε., Λέκκα Δ., Κανίδης Ε., Μακρυγιάννης Π.,
Μπελεσιώτης Β., Τζήμας Δ., Παπαδάκης Σπ.**

Προγραμματισμός Υπολογιστών

Διδακτικό υλικό μαθητή

**Γ' και Δ' τάξη ημερησίων και εσπερινών ΕΠΑ.Λ., του Τομέα
Πληροφορικής**

**Ινστιτούτο Τεχνολογίας Υπολογιστών & Εκδόσεων
«Διόφαντος»**

Ινστιτούτο Εκπαιδευτικής Πολιτικής

Πρόεδρος: **Κουζέλης Γεράσιμος**, Καθηγητής ΕΚΠΑ

Γραφείο έρευνας, σχεδιασμού και εφαρμογών Β΄

Προϊστάμενος: **Μάραντος Παύλος**

Επιστημονικά Υπεύθυνος: **Τσαπέλας Θεοδόσιος**

Συγγραφική ομάδα

Αράπογλου Αριστείδης, Εκπαιδευτικός Πληροφορικής

Βραχνός Ευριπίδης, Εκπαιδευτικός Πληροφορικής

Κανίδης Ευάγγελος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής

Λέκκα Δήμητρα, Εκπαιδευτικός Πληροφορικής

Μακρυγιάννης Παναγιώτης, Εκπαιδευτικός Πληροφορικής

Μπελεσιώτης Βασίλειος, Σχολικός Σύμβουλος Πληροφορικής

Τζήμας Δημήτριος, Εκπαιδευτικός Πληροφορικής

Παπαδάκης Σπυρίδων, Σχολικός Σύμβουλος Πληροφορικής

Επιμέλεια συντονισμού ομάδας

Κανίδης Ευάγγελος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής

Μπελεσιώτης Βασίλειος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής

Φιλολογική επιμέλεια

Ευφροσύνη Δεληγιάννη, Σχολική Σύμβουλος, ΠΕ02

Επιτροπή κρίσης

Βογιατζής Ιωάννης, Επίκουρος Καθηγητής Α.Τ.Ε.Ι. Αθηνών

Εφόπουλος Βασίλειος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής

Κωτσάκης Σταύρος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής

Προεκτυπωτικές εργασίες

Διεύθυνση εκδόσεων Ι.Τ.Υ.Ε. «Διόφαντος»

Εισαγωγικό σημείωμα

Το παρόν διδακτικό υλικό σχετίζεται με το Α.Π.Σ. του μαθήματος Προγραμματισμός ΑΠΣ_Τομέα_Πληρ_ΕΠΑΛ (2015) και το σχετικό Ωρολόγιο Πρόγραμμα (ΩΠ_Μαθημάτων_ΕΠΑΛ, 2015). Είναι άρρηκτα συνδεδεμένο με το παραπάνω Α.Π.Σ., τον Οδηγό Εκπαιδευτικού και το διδακτικό υλικό του Τετραδίου Εργασιών Μαθητή. Δημιουργήθηκε δε αμισθί και μετά τις σχετικές αποφάσεις του ΙΕΠ (Συγγραφείς_Προγραμματισμός, 2016).

Βασίζεται στο Πρόγραμμα Σπουδών και στον Οδηγό Εκπαιδευτικού του Μαθήματος *Πληροφορική Γ' τάξης* Γενικού Λυκείου, ως αναφέρεται στο (ΠΣ_Γ_ΓΕΛ, 2015).

Αποτελεί συνέχεια του διδακτικού υλικού του μαθήματος της Β' τάξης *Αρχές Προγραμματισμού Υπολογιστών*. Συγκεκριμένα, αποτελείται από δύο μέρη. Στο Μέρος Ι, καλύπτονται οι ενότητες του Προγράμματος Σπουδών 1,2,3,4, οι οποίες έχουν εμπλουτιστεί με σκοπό την επανάληψη και όπου χρειάζεται την εμβάθυνση, σε έννοιες που αναφέρθηκαν στη Β' τάξη και στα αντίστοιχα κεφάλαια του βιβλίου *Αρχές Προγραμματισμού Υπολογιστών*. Στο Μέρος ΙΙ, αναπτύσσονται οι νέες θεματικές ενότητες βάσει του Προγράμματος Σπουδών. Το παρόν υλικό είναι άρρηκτα συνδεδεμένο με το Τετράδιο Εργασίας Μαθητή.

Έχει ως στόχο, μέσα από την υλοποίηση ποικίλων δραστηριοτήτων, να βοηθήσει στον εμπλουτισμό των γνώσεων και στην απόκτηση των απαραίτητων δεξιοτήτων, για την επίλυση υπολογιστικών προβλημάτων και την ανάπτυξη κατάλληλων προγραμμάτων στη γλώσσα προγραμματισμού Python.

Οι συγγραφείς ευελπιστούν με το υλικό αυτό στην ουσιαστική υποστήριξη του μαθήματος *Προγραμματισμός Υπολογιστών* για την επίτευξη των μαθησιακών του στόχων, με δεδομένες τις συνθήκες ανάπτυξής του.

Τέλος, το όλο υλικό συνοδεύεται από τις παρατηρήσεις των συγγραφέων σε σχέση με θέματα καθορισμού διδακτέας ύλης - έχει προβλεφθεί να μπορεί αυτή να είναι υποσύνολο της όλης ύλης του βιβλίου- και του τρόπου διδασκαλίας, που αναφέρονται τόσο εδώ όσο και στο βιβλίο καθητή.

1. Από το πρόβλημα στην ανάπτυξη αλγορίθμου.....13

- 1.1 Εισαγωγή στη διαχείριση της πολυπλοκότητας ενός προβλήματος.....13
- 1.2 Στάδια επίλυσης προβλήματος.....14
- 1.3 Ανάλυση προβλήματος σε απλούστερα υποπροβλήματα.....15
- 1.4 Γραφική απεικόνιση της δομής ενός προβλήματος...16
- 1.5 Αναπαράσταση αλγορίθμων.....17

2. Ανάπτυξη προγράμματος.....19

- 2.1 Κύκλος ανάπτυξης προγράμματος/λογισμικού.....19
 - 2.1.1 Μοντέλο του καταρράκτη19
 - 2.1.2 Μοντέλο σπείρας.....20
 - 2.1.3 Η λογική συγγραφής προγράμματος ανάλογα με το είδος προγραμματισμού21
 - 2.1.4 Προστακτικός προγραμματισμός21
 - 2.1.5 Δηλωτικός προγραμματισμός22
 - 2.1.6 Λοιπά πρότυπα και τεχνικές προγραμματισμού 23
 - 2.1.7 Ενδεικτικά περιβάλλοντα και γλώσσες προγραμματισμού24
- 2.2 Εισαγωγή στις βασικές έννοιες του αντικειμενοστρεφούς προγραμματισμού.....25
 - 2.2.1 Σύγκριση Διαδικαστικού και Αντικειμενοστρεφούς προγραμματισμού28
 - 2.2.2 Αντικειμενοστρεφής σχεδίαση.....28

3. Βασικά στοιχεία γλώσσας προγραμματισμού30

- 3.1 Μεταβλητές και τύποι δεδομένων.....31

3.1.1	Τύποι δεδομένων	31
3.2	Αριθμητικές και λογικές πράξεις και εκφράσεις	32
3.3	Βασικές (ενσωματωμένες) συναρτήσεις	41
3.4	Δομή προγράμματος και καλές πρακτικές.....	43
3.5	Τύποι και δομές δεδομένων στις γλώσσες προγραμματισμού	44
4.	Αλγοριθμικές δομές	47
4.1	Αλγοριθμικές δομές - Ροές εκτέλεσης προγράμματος	47
4.1.1	Δομή ακολουθίας.....	47
4.1.2	Δομή επιλογής if (AN)	48
4.1.3	Δομή επανάληψης (for και while).....	53
4.2	Συναρτήσεις	59
4.2.1	Δημιουργώντας δικές μας συναρτήσεις	59
4.2.2	Παράμετροι συναρτήσεων.....	60
5.	Κλασικοί Αλγόριθμοι II	66
5.1	Διαδική αναζήτηση.....	67
5.2	Ταξινόμηση Ευθείας ανταλλαγής.....	76
5.3	Ταξινόμηση με Εισαγωγή.....	81
5.4	Δραστηριότητες - Άλυτες.....	87
5.5	Ερωτήσεις - Ασκήσεις	90
6.	Διαχείριση Αρχείων	92
6.1	Εισαγωγή - δημιουργία, άνοιγμα, κλείσιμο αρχείων ..	92
6.2	Ανάγνωση και εγγραφή σε αρχείο	95
6.3	Πρόσθετες λειτουργίες σε αρχεία.....	99
6.4	Ερωτήσεις - Ασκήσεις	103
7.	Προηγμένα στοιχεία γλώσσας προγραμματισμού.....	106
7.1	Υποπρογράμματα και τρόποι κλήσης τους	106
7.1.1	Υποπρογράμματα	106

7.1.2	Συναρτήσεις στην Python.....	108
7.2	Μεταβλητές και παράμετροι.....	112
7.2.1	Παράμετροι συναρτήσεων	112
7.2.2	Εμβέλεια των μεταβλητών.....	114
7.3	Αρθρώματα (Modules).....	118
7.3.1	Εισαγωγή.....	118
7.3.2	Σύντομη περιγραφή της Πρότυπης βιβλιοθήκης (Standard Library)	120
7.3.3	Πακέτα (Packages).....	122
7.4	Δραστηριότητες.....	123
	Δραστηριότητα 1	123
	Να εκτελεστούν και να κατανοηθούν τα παρακάτω παραδείγματα:.....	123
7.5	Ερωτήσεις	125
8.	Δομές Δεδομένων II.....	128
8.1	Συμβολοσειρές (strings).....	128
8.2	Λίστες	131
8.3	Στοίβα.....	140
8.4	Ουρά	144
8.5	Πλειάδες.....	146
8.6	Λεξικά	147
8.7	Εισαγωγή στους γράφους και τα δέντρα	148
8.8	Δραστηριότητες.....	151
8.9	Ερωτήσεις	152
9.	Εφαρμογές σε γλώσσα προγραμματισμού με χρήση API	154
9.1	Επικοινωνία ανθρώπου-υπολογιστή και διεπαφή χρήστη	155
9.1.1	Παραδείγματα – Εφαρμογές	156
9.2	Γενικές αρχές σχεδίασης διεπαφής	157

9.3	Η βιβλιοθήκη Tkinter για ανάπτυξη γραφικών διεπαφών GUI στην Python	159
9.4	Δραστηριότητες	167
9.4.1	Εργαστηριακές ασκήσεις	167
9.5	Σύντομα Θέματα για συζήτηση στην τάξη	169
9.6	Ερωτήσεις - Ασκήσεις	170
9.6.1	Ερωτήσεις	170
10.	Βάσεις δεδομένων	172
10.1	Αναφορά στο Μοντέλο Δεδομένων	173
10.2	Εισαγωγή στη διαχείριση Βάσεων Δεδομένων με προγραμματισμό	175
10.2.1	Η γλώσσα SQL	175
10.2.2	Η βιβλιοθήκη SQLite της Python	176
10.3	Δημιουργία ή σύνδεση με μια Βάση Δεδομένων στην Python	177
10.4	Εισαγωγή, ενημέρωση και διαγραφή δεδομένων	178
10.4.1	Ενημέρωση δεδομένων	181
10.4.2	Διαγραφή πίνακα	183
10.5	Αναζήτηση και ταξινόμηση δεδομένων	183
10.6	Δραστηριότητες	186
10.7	Ερωτήσεις	187
10.8	Εργαστηριακές ασκήσεις	188
11.	Αντικειμενοστρεφής Προγραμματισμός	192
11.1	Αντικείμενα και Κλάσεις	192
11.2	Στιγμιότυπα (αυτόματη αρχικοποίηση αντικειμένων)	197
11.3	Ιδιότητες και Μέθοδοι	203
11.4	Χαρακτηριστικά: Κληρονομικότητα, Πολυμορφισμός	207
11.4.1	Κληρονομικότητα (Inheritance)	207

11.4.2	Πολυμορφισμός (polymorphism).....	212
11.4.3	Ενθυλάκωση και Απόκρυψη δεδομένων	212
11.5	Οδήγηση από Γεγονότα - Μηνύματα	216
11.6	Δραστηριότητες.....	221
11.7	Ερωτήσεις - Ασκήσεις.....	223
12.	Εισαγωγή στην Υπολογιστική Σκέψη .	226
12.1	Η έννοια της Υπολογιστικής Σκέψης.....	228
12.2	Χαρακτηριστικά της Υπολογιστικής Σκέψης	229
12.3	Υπολογιστική σκέψη και επίλυση προβλημάτων.....	231
12.4	Δραστηριότητες κεφαλαίου	232
	Δραστηριότητα 1. Διαχείριση προβλημάτων.....	232
	Δραστηριότητα 2. Γενίκευση	233
	Δραστηριότητα 3. Γενίκευση	233
12.5	Ερωτήσεις - Ασκήσεις.....	234

Μέρος I

Κεφάλαια

1. Από το πρόβλημα στην ανάπτυξη αλγορίθμου
2. Ανάπτυξη προγράμματος
3. Βασικά στοιχεία γλώσσας προγραμματισμού
4. Αλγοριθμικές δομές

1. Από το πρόβλημα στην ανάπτυξη αλγορίθμου

Εισαγωγή

Στο κεφάλαιο αυτό, αρχικά θα προσεγγίσουμε θέματα που αφορούν την αναγνώριση της δομής ενός προβλήματος και της πολυπλοκότητάς του. Στη συνέχεια, θα γνωρίσουμε τη διαδικασία της αφαίρεσης και την απλοποίηση ενός προβλήματος με την ανάλυσή του σε απλούστερα υποπροβλήματα. Τέλος, θα γνωρίσουμε τον τρόπο που περιγράφουμε αλγοριθμικά τη λύση του, εκφρασμένη με ψευδοκώδικα ή διάγραμμα ροής.

Λέξεις κλειδιά

Πρόβλημα, επίλυση προβλήματος, ανάλυση προβλήματος, αφαίρεση, πολυπλοκότητα, αλγόριθμος, αναπαράσταση αλγορίθμου.

Διδακτικές Ενότητες

Για να αναπτύξουμε προγράμματα που μας βοηθούν στην επίλυση των προβλημάτων, είναι αναγκαία η κατανόηση τριών βασικών εννοιών: της έννοιας του *προβλήματος*, της έννοιας του *αλγορίθμου* (algorithm) για τη περιγραφή της λύσης του και της *δομής δεδομένων* (data structure), για να μπορούν να χρησιμοποιηθούν τα δεδομένα από τον αλγόριθμο. Η *σχεδίαση αλγορίθμων* αποτελεί σημαντικό μέρος της διαδικασίας *επίλυσης ενός προβλήματος*. Συνοδεύεται με την ανάλυσή του προβλήματος σε απλούστερα, τη σύνθεση των λύσεων των επιμέρους προβλημάτων και κυρίως με την περιγραφή και μορφοποίηση της λύσης του. Αυτό, ώστε η λύση να μπορεί να αναπαρασταθεί σε μορφή κατανοητή από τον υπολογιστή.

1.1 Εισαγωγή στη διαχείριση της πολυπλοκότητας ενός προβλήματος

Ο άνθρωπος αντιμετωπίζει καθημερινά μικρά ή μεγάλα προβλήματα, πολλά από τα οποία προσπαθεί να τα επιλύσει κατά το δυνατόν ευκολότερα με τη βοήθεια των εργαλείων και την τεχνολογία της εποχής του.

Με τον όρο **πρόβλημα** προσδιορίζεται μια κατάσταση η οποία χρήζει αντιμετώπισης, απαιτεί λύση, η δε λύση της δεν είναι ούτε γνωστή, ούτε προφανής.

Τα προβλήματα μπορούμε να τα διακρίνουμε, με κριτήριο τη δυνατότητα επίλυσης, σε *επιλύσιμα*, όταν η λύση τους είναι γνωστή, *ανοιχτά*, όταν η λύση τους δεν έχει βρεθεί και παράλληλα δεν έχει αποδειχτεί ότι υπάρχει και *άλυτα*, όταν έχει αποδειχθεί ότι αυτή δεν υπάρχει.

Τα προβλήματα χαρακτηρίζονται επίσης, ως απλά ή σύνθετα. Εκείνα που η λύση είναι εύκολο να βρεθεί χαρακτηρίζονται ως *απλά*. Ως *σύνθετα* χαρακτηρίζονται τα προβλήματα που αποτελούνται από πολλά μέρη και στη λύση τους συμμετέχουν πολλοί παράγοντες, που συχνά αλληλεπιδρούν μεταξύ τους. Για παράδειγμα, προβλήματα όπως η ρύπανση του περιβάλλοντος, ή η ασφαλής πλοήγηση στο Διαδίκτυο χαρακτηρίζονται ως σύνθετα προβλήματα.

Τα προβλήματα που απαντώνται στην καθημερινότητα και στις επιστήμες ποικίλουν, με ορισμένα από αυτά να μπορούν να επιλυθούν με τη βοήθεια υπολογιστή. Αυτά τα προβλήματα σχηματίζουν μια ευρύτερη κατηγορία με την ονομασία, *υπολογιστικά προβλήματα*, προβλήματα με τα οποία θα ασχοληθούμε στη συνέχεια.

1.2 Στάδια επίλυσης προβλήματος

Η επίλυση ενός προβλήματος εξελίσσεται σε τρία στάδια: α) της *κατανόησης*, β) της *ανάλυσης* και γ) του *σχεδιασμού και της υλοποίησης* της λύσης.

Στάδιο Α. Κατανόηση προβλήματος

Στο πρώτο στάδιο, της κατανόησης του προβλήματος, γίνεται η διάκριση της δομής του προβλήματος, διαδικασία αναγκαία, ειδικά σε ένα σύνθετο πρόβλημα. Με τον όρο *δομή*, εννοούμε τα επιμέρους στοιχεία-τμήματα που αποτελούν το πρόβλημα, καθώς και τον τρόπο με τον οποίο αυτά συνδέονται και αλληλεπιδρούν. Για την κατανόηση της δομής, είναι απαραίτητη η ανάλυση του προβλήματος στα επιμέρους στοιχεία του.

Στάδιο Β. Ανάλυση προβλήματος

Είναι το στάδιο στο οποίο το αρχικό πρόβλημα, διασπάται σε άλλα επί μέρους απλούστερα προβλήματα, διαδικασία με την οποία, ελαττώνεται η δυσκολία αντιμετώπισης των προβλημάτων, όσο περισσότερο προχωράει βαθύτερα η ανάλυσή τους σε απλούστερα προβλήματα.

Στάδιο Γ. Σχεδιασμός και Υλοποίηση

Πρόκειται για το στάδιο που υλοποιείται η λύση του προβλήματος, μέσω της λύσης των επιμέρους προβλημάτων. Στο στάδιο αυτό αρχικά σχεδιάζονται οι διαφορετικές δυνατές λύσεις και τέλος επιλέγεται και υλοποιείται η πλέον κατάλληλη.

Για τη λύση ενός προβλήματος μπορεί να διατυπωθούν διαφορετικοί αλγόριθμοι, με αποτέλεσμα να τίθεται το θέμα της αξιολόγησής τους, δηλαδή ποιος από αυτούς είναι ο βέλτιστος. Η αξιολόγηση αλγορίθμου είναι συχνά ένα σύνθετο θέμα και η ερώτηση «υπάρχει αποδοτικότερος αλγόριθμος που λύνει το ίδιο πρόβλημα;» μελετάται από τη Θεωρητική Πληροφορική (Θεωρία Υπολογιστικής Πολυπλοκότητας). Η αποδοτικότητα αλγορίθμου αναφέρεται κυρίως στην ταχύτητα εκτέλεσής του, δηλαδή πόσο χρόνο χρειάζεται για να λύσει το πρόβλημα, αλλά και στο χώρο (ποσότητα) της κύριας μνήμης που χρησιμοποιεί. Έτσι, για παράδειγμα, αν για δύο αλγόριθμους Α και Β, που λύνουν το ίδιο πρόβλημα, συγκρινόμενοι με τα ίδια δεδομένα και συνθήκες (γλώσσα προγραμματισμού, υπολογιστικό σύστημα, σκληρός δίσκος κλπ.), διαπιστωθεί ότι ο αλγόριθμος Α δίνει λύση σε μικρότερο χρόνο από το χρόνο του Β, τότε ο Α χαρακτηρίζεται ως αποδοτικότερος του Β. Παρόμοια, αν ο αλγόριθμος Α χρησιμοποιεί λιγότερη μνήμη από τον Β χαρακτηρίζεται αποδοτικότερος του Β. Υποθέτουμε ότι και στις δύο περιπτώσεις ο άλλος παράγοντας αντίστοιχα, είναι ίδιος.

1.3 Ανάλυση προβλήματος σε απλούστερα υποπροβλήματα

Υπάρχουν διάφορες επιστημονικές μέθοδοι του τρόπου εργασίας για την ανάλυση της δομής ενός προβλήματος και την εύρεση των τμημάτων-υποπροβλημάτων που το αποτελούν. Οι μέθοδοι αυτές

είναι γνωστές ως: *αναλυτική μέθοδος* (από πάνω προς τα κάτω - Top Down), *συνθετική* (από κάτω προς τα πάνω - Bottom Up) και *μικτή μέθοδος*.

Στο κεφάλαιο αυτό θα ασχοληθούμε με την αναλυτική μέθοδο επίλυσης προβλήματος (Top Down problem solving), η οποία είναι και η ευρύτερα εφαρμοζόμενη μέθοδος. Η γενική αρχή της αναλυτικής μεθόδου ορίζει ότι, για να λύσουμε κάποιο σύνθετο πρόβλημα, πρέπει:

- Να καθορίσουμε τα υποπροβλήματα.
- Να επαναλάβουμε τη διαδικασία αυτή για κάθε ένα από τα υποπροβλήματα αυτά, σε όσο βάθος αυτό είναι αναγκαίο.
- Να προχωράμε στην άμεση επίλυσή τους, όταν φτάσουμε σε υποπροβλήματα που δεν απαιτούν επιπλέον διάσπαση.

Η λύση του προβλήματος επιτυγχάνεται με τη σύνθεση των λύσεων των επιμέρους προβλημάτων.

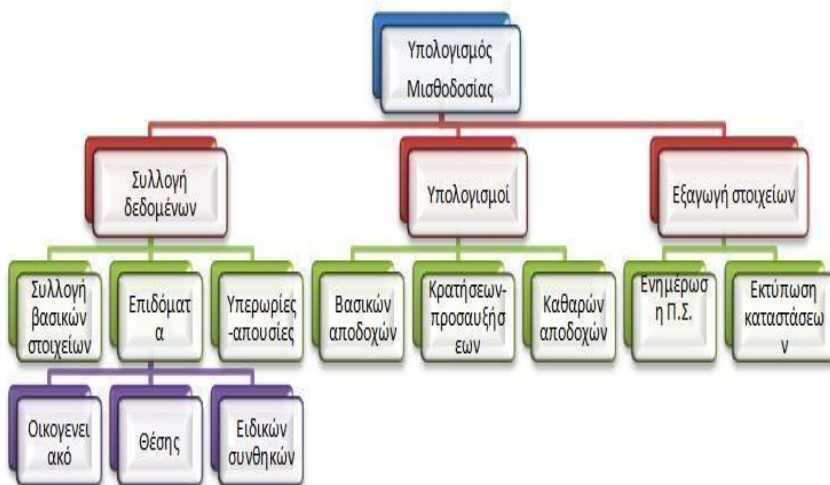
1.4 Γραφική απεικόνιση της δομής ενός προβλήματος

Η ανάλυση ενός προβλήματος σε άλλα απλούστερα, αναδύει παράλληλα και τη δομή του, για τη γραφική απεικόνιση της οποίας χρησιμοποιείται συχνά μια *διαγραμματική αναπαράσταση*, σύμφωνα με την οποία:

- Το αρχικό πρόβλημα αναπαρίσταται από ένα ορθογώνιο παραλληλόγραμμο.
- Κάθε ένα από τα απλούστερα προβλήματα στα οποία αναλύεται αναπαρίσταται, επίσης, από ένα ορθογώνιο παραλληλόγραμμο.
- Τα παραλληλόγραμμα που αντιστοιχούν στα απλούστερα προβλήματα σχηματίζονται ένα επίπεδο χαμηλότερα. Έτσι σε κάθε κατώτερο επίπεδο, δημιουργείται η γραφική αναπαράσταση των προβλημάτων στα οποία αναλύονται τα προβλήματα του αμέσως υψηλότερου επιπέδου.

Η διαγραμματική αναπαράσταση δίνει μια απτή απεικόνιση της δομής του προβλήματος και βοηθάει τόσο στην καλύτερη κατανόηση του ίδιου όσο και στη σχεδίαση της λύσης του.

Στην εικόνα 1.1 φαίνεται η διαγραμματική αναπαράσταση παραδείγματος μισθοδοσίας. Η ανάλυση του προβλήματος βρίσκεται στο Τετράδιο Εργασιών Μαθητή.



Εικόνα 1-1. Διαγραμματική αναπαράσταση προβλήματος "Υπολογισμός Μισθοδοσίας"

1.5 Αναπαράσταση αλγορίθμων

Στη βιβλιογραφία συναντώνται διάφοροι τρόποι αναπαράστασης ενός αλγορίθμου, όπως:

- Με *φυσική γλώσσα* (natural language), η οποία αποτελεί τον πιο απλό, ανεπεξέργαστο και αδόμητο τρόπο παρουσίασης ενός αλγορίθμου, όπου με απλά λόγια και ελεύθερες εκφράσεις περιγράφουμε τα βήματα του αλγορίθμου. Αυτός ο τρόπος έκφρασης κρύβει αυξημένη πιθανότητα λάθους, λόγω της ασάφειας στην περιγραφή.
- Με *διαγραμματικές τεχνικές* (diagramming techniques), που συνιστούν ένα γραφικό τρόπο παρουσίασης του αλγορίθμου. Η πιο γνωστή τεχνική είναι το *διάγραμμα ροής* (flow chart).

- Με *κωδικοποίηση* (coding), δηλαδή με ένα πρόγραμμα γραμμένο είτε σε *ψευδογλώσσα* είτε σε κάποια *γλώσσα προγραμματισμού* που, όταν εκτελεσθεί θα δώσει τα ίδια αποτελέσματα με τον αλγόριθμο. *Ψευδογλώσσα* ή *Ψευδοκώδικας* είναι μια υποθετική δομημένη γλώσσα με στοιχεία από υπαρκτές γλώσσες προγραμματισμού, με λίγες εντολές και απλοποιημένη σύνταξη, χρήσιμη ιδιαίτερα στα πρώτα στάδια εκμάθησης του προγραμματισμού.

Στο βιβλίο αυτό θα χρησιμοποιηθεί για την υλοποίηση των αλγορίθμων η γλώσσα προγραμματισμού Python, ενώ για την αναπαράστασή τους, το διάγραμμα ροής και η ψευδογλώσσα, όπως αυτά χρησιμοποιούνται και στο Βιβλίο *Εισαγωγή στις Αρχές της Επιστήμης των Υπολογιστών* της Β΄ τάξης ΕΠΑ.Λ.

2. Ανάπτυξη προγράμματος

Εισαγωγή

Στο κεφάλαιο αυτό αναπτύσσονται, σε ένα πρώτο επίπεδο, θέματα σχετικά με τον κύκλο ανάπτυξης ενός προγράμματος και τις μεθοδολογίες που χρησιμοποιούνται. Αναλύονται θέματα ειδών προγραμματισμού και αναφέρονται αρχικές έννοιες για τον αντικειμενοστρεφή προγραμματισμό, καθώς και τεχνικές σχεδίασης και περιγραφής συστημάτων.

Λέξεις κλειδιά

Κύκλος ανάπτυξης προγράμματος, μοντέλα ανάπτυξης και σχεδίασης λογισμικού, προγραμματιστικά υποδείγματα, αντικειμενοστρεφής / δομημένος προγραμματισμός.

Διδακτικές Ενότητες

2.1 Κύκλος ανάπτυξης προγράμματος/λογισμικού

Η διαδικασία ανάπτυξης λογισμικού εξελίσσεται σε διακριτές φάσεις ή στάδια. Θεωρείται υποσύνολο του κύκλου ζωής ενός συστήματος λογισμικού που ξεκινά από την ανάλυση απαιτήσεων και τελειώνει με την παύση λειτουργίας του.

Αρκετές φορές, οι θεωρητικές προσεγγίσεις για την ανάπτυξη λογισμικού έχουν δεχθεί κριτική για την αποτελεσματικότητά τους, καθώς συχνά, στη πράξη, παρουσιάζονται σημαντικές αποκλίσεις από τον αρχικό σχεδιασμό. Παρόλα αυτά είναι ιδιαίτερα χρήσιμο, η ανάπτυξη ενός λογισμικού να ακολουθεί συγκεκριμένα μοντέλα, τα οποία έχουν προταθεί ανάλογα με τα προτερήματα και τις αδυναμίες τους.

Μεταξύ των βασικών μεθοδολογιών (μοντέλων) που έχουν προταθεί και ακολουθούνται, θα αναφερθούμε στο μοντέλο του *καταρράκτη* (waterfall model) και της *σπειροειδούς προσέγγισης* (spiral model).

2.1.1 Μοντέλο του καταρράκτη

Πρόκειται για το μοντέλο που υποδιαιρεί τη διαδικασία ανάπτυξης ενός συστήματος λογισμικού στις ακόλουθες φάσεις:

- Καθορισμού απαιτήσεων.
- Ανάλυσης απαιτήσεων.
- Σχεδίασης.
- Υλοποίησης.
- Ολοκλήρωσης. Λειτουργίας και συντήρησης.

Οι φάσεις εφαρμόζονται διαδοχικά με τη σειρά (γραμμικά) με κύκλους ανατροφοδότησης.



Εικόνα 2-1. Διαγραμματική αναπαράσταση του μοντέλου του καταρράκτη

2.1.2 Μοντέλο σπείρας

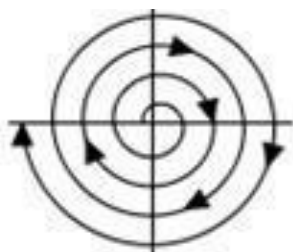
Στο μοντέλο της **σπείρας**, η ανάπτυξη ακολουθεί μια εξελικτική διαδικασία με την επαναληπτική εκτέλεση ενός κύκλου φάσεων, που σε καθεμιά δημιουργείται μια ενδιάμεση έκδοση του τελικού προϊόντος, η οποία βελτιώνεται κατά τον επόμενο κύκλο κ.ο.κ. Η διαδικασία αυτή συνεχίζεται μέχρι να παραχθεί μια έκδοση που να ικανοποιεί τις απαιτήσεις των χρηστών. Το πλεονέκτημα αυτού του μοντέλου είναι η γρήγορη ανάπτυξη ενός πρωτότυπου και η ανάλυση όλων των κινδύνων, πριν από κάθε φάση. Το μοντέλο αυτό χρησιμοποιείται περισσότερο σε έργα μεγάλης κλίμακας.

Κατά την ανάπτυξη ενός προγράμματος, εργαζόμαστε παρόμοια με τη διαδικασία ανάπτυξης ενός λογισμικού και ακολουθείται η

μεθοδολογία υλοποίησης του **κύκλου ανάπτυξης προγράμματος** (program development life cycle-PDLC).

Μια τέτοια μεθοδολογία ανάπτυξης προγράμματος μπορεί να αποτελείται από φάσεις, όπως:

- Ανάλυση του προβλήματος.
- Σχεδίαση.
- Συγγραφή κώδικα.
- Έλεγχος και εκσφαλμάτωση.
- Τεκμηρίωση.



Εικόνα 2-2. Διαγραμματική αναπαράσταση του μοντέλου της Σπείρας

2.1.3 Η λογική συγγραφής προγράμματος ανάλογα με το είδος προγραμματισμού

Από τη δεκαετία του 1960 και μέχρι σήμερα, έχουν αναπτυχθεί διάφορα είδη προγραμματισμού που τα υποστήριξαν πολλές γλώσσες προγραμματισμού. Μπορούμε να τα κατηγοριοποιήσουμε σε μεγάλες κατευθύνσεις, τα λεγόμενα **προγραμματιστικά υποδείγματα** (programming paradigms).

Τα βασικά προγραμματιστικά υποδείγματα είναι τα ακόλουθα:

- Ο Προστακτικός προγραμματισμός.
- Ο Δηλωτικός προγραμματισμός.

2.1.4 Προστακτικός προγραμματισμός

Ο **Προστακτικός προγραμματισμός** (Imperative programming) βασίζεται σε εντολές που υλοποιούν τα βήματα ενός αλγόριθμου, και βρίσκεται πιο κοντά στη λογική λειτουργίας του υπολογιστή. Γλώσσες που ακολούθησαν το είδος αυτό είναι οι κλασικές γλώσσες προγραμματισμού, όπως Cobol, Fortran, Pascal, C κ.ά.

Δομημένος και μη προγραμματισμός

Κατά την αρχική περίοδο του προγραμματισμού, η διακλάδωση και η αλλαγή της ροής εκτέλεσης ενός προγράμματος γίνονταν με τη χρήση της εντολής goto. Με την εντολή αυτή, η ροή εκτέλεσης μεταφερόταν σε οποιοδήποτε σημείο του προγράμματος, γεγονός που οδηγούσε σε μη δομημένα προγράμματα (μη δομημένος προγραμματισμός - unstructured programming). Στη συνέχεια δημιουργήθηκε το πρότυπο του **Δομημένου προγραμματισμού** (Structured programming), όπου η αλλαγή της ροής εκτέλεσης γίνεται με συγκεκριμένους κανόνες.

Ο **Διαδικαστικός προγραμματισμός** (Procedural programming) αποτελεί μια υποκατηγορία του Δομημένου προγραμματισμού, με το πρόγραμμα να αποτελείται από ομάδες εντολών, τις *διαδικασίες* (procedures). Η γλώσσα προγραμματισμού που αξιοποίησε, ήδη από το 1970, το είδος αυτό είναι η Pascal.

Ο **Αντικειμενοστρεφής προγραμματισμός** (Object-oriented programming) βασίζεται -σε αντίθεση με το Διαδικαστικό προγραμματισμό- σε αντικείμενα που αλληλεπιδρούν μεταξύ τους. Ο Αντικειμενοστρεφής προγραμματισμός αποτελεί ένα πρότυπο που ταιριάζει περισσότερο στη λογική οργάνωσης και λειτουργίας του πραγματικού κόσμου.

2.1.5 Δηλωτικός προγραμματισμός

Ο **Δηλωτικός προγραμματισμός** (Declarative programming) βασίζεται στην περιγραφή του σκοπού, ο οποίος ζητείται να επιτύχει το πρόγραμμα. Στο γενικό αυτό υπόδειγμα ανήκουν διάφορες υποκατηγορίες προγραμματισμού, όπως είναι ο Συναρτησιακός και ο Λογικός.

Ο **Συναρτησιακός προγραμματισμός** (functional programming) βασίζεται σε μαθηματικές συναρτήσεις, με γλώσσες όπως Lisp, Logo κ.ά.

Στο **Λογικό προγραμματισμό** (logic programming), ένα πρόγραμμα είναι ένα σύνολο από αξιώματα ή κανόνες οι οποίοι καθορίζουν σχέσεις ανάμεσα σε αντικείμενα. Υπολογισμός ενός λογικού προγράμματος είναι ένα συμπέρασμα που συνάγεται από τα αποτελέσματά του, όπως για παράδειγμα γίνεται στην Prolog.

Στο προγραμματιστικό πρότυπο του δηλωτικού προγραμματισμού μπορούμε να θεωρήσουμε ότι ανήκουν και άλλες γλώσσες που δεν υπάγονται στις δύο προηγούμενες κατηγορίες. Χαρακτηριστικές είναι η *HTML* (HyperText Markup Language), γλώσσα σήμανσης-χαρακτηρισμού υπερκειμένου και εν μέρει η *SQL* (Structured Query Language) γλώσσα για τη διαχείριση δεδομένων, σε ένα Σύστημα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων (RDBMS-Relational Database Management System).

2.1.6 Λοιπά πρότυπα και τεχνικές προγραμματισμού

Εκτός από τα παραπάνω υποδείγματα προγραμματισμού υπάρχουν και άλλα τα οποία, είτε δεν μπορούν να χαρακτηριστούν πλήρως ως προγραμματιστικά υποδείγματα, είτε αποτελούν *τεχνικές και μεθοδολογίες προγραμματισμού*, που θα αναφέρουμε στη συνέχεια.

Παράλληλος προγραμματισμός (parallel programming). Επιτρέπει ταυτόχρονη εκτέλεση διαδικασιών από διαφορετικούς επεξεργαστές, όπως στην γλώσσα προγραμματισμού Occam.

Προγραμματισμός οδηγούμενος από γεγονότα (event-driven programming). Αποτελεί περισσότερο *τεχνική αρχιτεκτονικής ενός προγράμματος* σχετικά με τη ροή του, παρά προγραμματιστικό υπόδειγμα. Η ροή του προγράμματος εξαρτάται από την ύπαρξη *γεγονότων* (events), όπως είναι, για παράδειγμα, ένα μήνυμα ενός αισθητήρα ή μια ενέργεια του χρήστη με το πάτημα του ποντικιού ή ενός πλήκτρου. Παραδείγματα γλωσσών που το ενσωματώνουν είναι η Python, Java κ.ά.

Οπτικός προγραμματισμός (Visual programming). Εκφράζει τη δυνατότητα γλωσσών ή περιβαλλόντων προγραμματισμού να παρέχουν τη δυνατότητα δημιουργίας του προγράμματος μέσω γραφικών αντικειμένων, αντί της πληκτρολόγησης του κώδικα ο οποίος αντιστοιχεί σε εντολές. Για παράδειγμα, στην κατηγορία αυτή ανήκουν περιβάλλοντα δημιουργίας σεναρίων όπως το Kodu (Kodu, 2016) και το Alice (Alice, 2016) ή το Scratch (Scratch, 2016).

Προγραμματισμός Δέσμης ενεργειών (Script programming). Είναι τύπος προγραμματισμού και όχι υπόδειγμα, δημιουργίας μικρών τμημάτων κώδικα και όχι ολοκληρωμένων προγραμμάτων.

Είναι υψηλού επιπέδου προγραμματισμός που διερμηνεύεται κατά την εκτέλεση από ένα άλλο πρόγραμμα, όπως ένας φυλλομετρητής.

Αρθρωτός ή Τμηματικός Προγραμματισμός (Modular programming). Σχετίζεται περισσότερο με *τεχνική σχεδίασης λογισμικού* παρά με πρότυπο. Χαρακτηρίζεται από τη διαίρεση του προβλήματος σε απλούστερα τμήματα, αυτά με τη σειρά τους σε επί μέρους μικρότερα κ.ο.κ. Παρέχει απλούστευση της επίλυσης ενός προβλήματος, ευκολία κωδικοποίησης και συντήρησης. Γενικά ως *τμήμα* θεωρούμε ένα σύνολο ενεργειών το οποίο εκτελεί μια καθορισμένη λειτουργία ενός προγράμματος και είναι κατά το δυνατόν ανεξάρτητο από τα άλλα τμήματα.

Ιεραρχικός σχεδιασμός

Η μέθοδος ανάλυσης ενός προβλήματος σε μικρότερα, είναι εκείνη με την οποία αντιμετωπίζουμε το πρόβλημα ως μια πολυεπίπεδη δομή. Έτσι, για τη σχεδίασή του, ξεκινάμε από το υψηλότερο επίπεδο και στη συνέχεια το αναλύουμε σε όλο και χαμηλότερα, έως ότου φθάσουμε στο κατώτερο επίπεδο ανάλυσης. Η τεχνική αυτή ονομάζεται **ιεραρχικός σχεδιασμός** (top down design).

2.1.7 Ενδεικτικά περιβάλλοντα και γλώσσες προγραμματισμού

Pascal. Η πλέον κλασική γλώσσα δομημένου προγραμματισμού. **Visual Basic** (Visual Basic, 2016). Περιβάλλον προγραμματισμού, που ακολουθεί μικτό πρότυπο υποδειγμάτων.

C++. Επέκταση της C. Αποτελεί γλώσσα Αντικειμενοστρεφούς προγραμματισμού, αν και μπορεί να χρησιμοποιηθεί και για διδασκαλία Διαδικαστικού προγραμματισμού.

Java (Java, 2016). Σύγχρονη γλώσσα Αντικειμενοστρεφούς προγραμματισμού.

Python (Python, 2016). Γλώσσα που ανήκει ουσιαστικά σε μικτά υποδείγματα προγραμματισμού, όπως αντικειμενοστρεφές, συναρτησιακό και διαδικαστικό.

2.2 Εισαγωγή στις βασικές έννοιες του αντικειμενοστρεφούς προγραμματισμού

2.2 Εισαγωγή στις βασικές έννοιες του αντικειμενοστρεφούς προγραμματισμού

Στην παράγραφο αυτή θα αναφερθούμε επαναληπτικά σε σχέση με την προηγούμενη τάξη, στον Αντικειμενοστρεφή προγραμματισμό, ενώ πληρέστερη ανάλυση θα γίνει στο κεφάλαιο 11.



Εικόνα 2-3. Είδη προγραμματισμού

Η μέχρι τώρα διδασκαλία του προγραμματισμού ακολουθεί το υπόδειγμα του Δομημένου προγραμματισμού. Ο Δομημένος προγραμματισμός έλυσε σημαντικά προβλήματα του προγραμματισμού, αλλά οδήγησε και στην ανάπτυξη πολύπλοκων προγραμμάτων. Η κατασκευή των προγραμμάτων αυτών και η συντήρησή τους απαιτούσε ειδικούς με βαθιές γνώσεις σε επιμέρους θέματα, όπως στη διαχείριση μνήμης, και στα λειτουργικά συστήματα. Η πολυπλοκότητα κατασκευής των προγραμμάτων δημιούργησε μια άλλη λογική προγραμματισμού που κινείται σε διαφορετική κατεύθυνση. Πρόκειται για τον Αντικειμενοστρεφή προγραμματισμό.

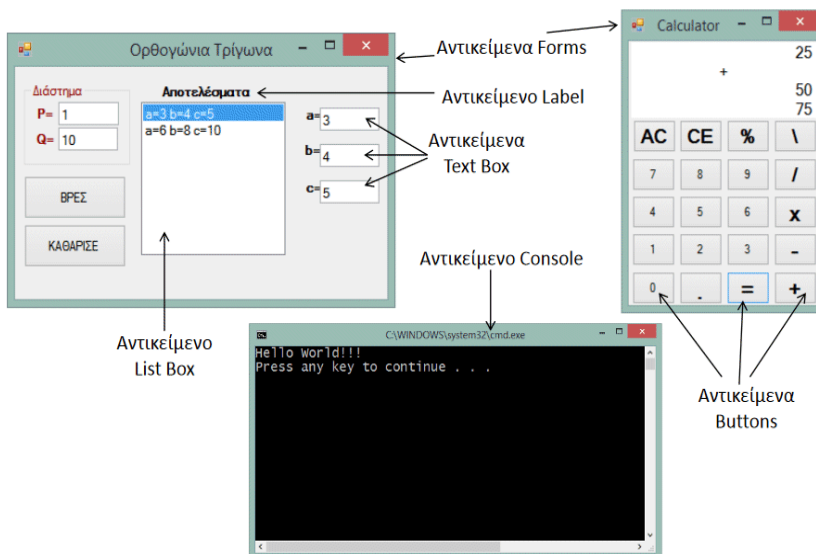
Βασική αρχή πάνω στην οποία δομήθηκε Αντικειμενοστρεφής προγραμματισμός είναι ότι ο πραγματικός κόσμος αποτελείται από αντικείμενα (objects). Τα αντικείμενα αυτά παίρνουν μορφή και η

συμπεριφορά τους εξαρτάται από τα ερεθίσματα που λαμβάνουν από το περιβάλλον στο οποίο βρίσκονται. Ως αντικείμενα μπορούμε να θεωρήσουμε σχεδόν τα πάντα, δηλαδή οτιδήποτε υπάρχει στον πραγματικό κόσμο, όπως π.χ. μαθητής, παγκάκι, πορτοκάλι, ραδιόφωνο, αυτοκίνητο κ.λ.π. (εικόνα 2-4).



Εικόνα 2-4. Φυσικά αντικείμενα

Τα αντικείμενα δε σημαίνει ότι απαραίτητα έχουν υλική υπόσταση. Μπορούν να θεωρηθούν ως αντικείμενα έννοιες, όπως μια συγκεκριμένη μαθηματική θεωρία ή προγραμματιστικά αντικείμενα, όπως αυτά που εμφανίζει η εικόνα 2-5. Για παράδειγμα, κάθε παράθυρο οθόνης είναι ένα αντικείμενο.



Εικόνα 2-5. Προγραμματιστικά αντικείμενα

Κάθε αντικείμενο είναι ξεχωριστό και έχει ένα σύνολο από χαρακτηριστικά. Για παράδειγμα: το παγκάκι είναι ξύλινο, η μπάλα είναι στρογγυλή, ο μαθητής φοιτά σε μια τάξη κλπ. Το σύνολο αυτών των χαρακτηριστικών αποτελούν τις *ιδιότητες* του αντικειμένου.

Ανάλογα με την εφαρμογή που αναπτύσσουμε, καθορίζουμε τόσο τα αντικείμενα όσο και τις ιδιότητες που μας ενδιαφέρουν. Ένα αντικείμενο συνήθως μπορεί να εκτελέσει κάποια λειτουργία. Για παράδειγμα: ο μαθητής μπορεί να πάρει μια εργασία, να προβιβαστεί στην επόμενη τάξη, η μπάλα να αναπηδήσει, ένα παράθυρο οθόνης να κλείσει κλπ. Τέτοια αποτελέσματα αποτελούν τη λεγόμενη *συμπεριφορά* του αντικειμένου.

Η συμπεριφορά του αντικειμένου ενεργοποιείται από ένα γεγονός. Για παράδειγμα: ρίχνουμε τη μπάλα στο πάτωμα ή κάνουμε κλικ σε ένα σημείο του παραθύρου.

Το πώς ακριβώς θα συμπεριφερθεί ένα προγραμματιστικό αντικείμενο εξαρτάται από ένα τμήμα κώδικα που ανήκει στο αντικείμενο και ονομάζεται *κώδικας-χειριστής*.

Όπως ήδη αναφέραμε ένα αντικείμενο είναι ξεχωριστό, έχει, δηλαδή, τη δική του ταυτότητα. Αυτό πρακτικά σημαίνει ότι, ακόμη και αν δύο αντικείμενα έχουν ακριβώς τις ίδιες ιδιότητες, εξακολουθούν να παραμένουν δύο ανεξάρτητα αντικείμενα. Για παράδειγμα: δύο μπάλες ποδοσφαίρου έχουν τα ίδια χαρακτηριστικά ανεξάρτητα σε ποιο παιχνίδι χρησιμοποιείται η κάθε μία. Για λόγους απλότητας και "οικονομίας" θα επιθυμούσαμε σε παρόμοια αντικείμενα να καταγράφονται τα χαρακτηριστικά τους μόνο μια φορά. Έτσι οδηγούμαστε στις *κλάσεις* που είναι, ουσιαστικά, "καλούπια" για τη δημιουργία παρόμοιων αντικειμένων. Με την ίδια κλάση, μπορούν να δημιουργηθούν πολλά διαφορετικά αντικείμενα. Κάθε παράθυρο είναι ένα αντικείμενο που ανήκει στην κλάση παραθύρων, αλλά και κάθε ένα από τα επιπλέον στοιχεία που περιέχει όπως ετικέτες, κουμπιά, μπάρες κύλισης κλπ, είναι επίσης αντικείμενα τα οποία ανήκουν σε μια πιο γενική κατηγορία ή κλάση.

Το αντικείμενο έχει τιμές στις ιδιότητες, ενώ κληρονομεί τις λειτουργίες της κλάσης από την οποία προέρχεται.

Όταν κάνουμε μια ενέργεια (γεγονός), όπως, για παράδειγμα αριστερό κλικ με το ποντίκι σε ένα από τα στοιχεία αυτά, ενημερώνονται όλα για την ενέργεια, αλλά αντιδρά μόνο το ενεργό. Έτσι για παράδειγμα, αν έχουμε πολλά όμοια παράθυρα του φυλλομετρητή το ένα πίσω από το άλλο και πατήσουμε το κουμπί κλεισίματος (γεγονός), θα κλείσει μόνο το ενεργό παράθυρο (συμπεριφορά).

Η όλη παραπάνω λειτουργία των παραθύρων στηρίζεται σε μια εφαρμογή που λέγεται *γραφική διεπαφή χρήστη* (graphical user interface-GUI) και η οποία αποτελεί ένα χαρακτηριστικό παράδειγμα Αντικειμενοστρεφούς προγραμματισμού.

2.2.1 Σύγκριση Διαδικαστικού και Αντικειμενοστρεφούς προγραμματισμού

Οι γλώσσες Αντικειμενοστρεφούς προγραμματισμού επιτρέπουν ένα υψηλότερο επίπεδο *αφαίρεσης*, όρος που θίγεται στο κεφάλαιο 12, από αυτές του Διαδικαστικού προγραμματισμού, για την επίλυση των προβλημάτων της πραγματικής ζωής. Στη *δομημένη σχεδίαση* σκεφτόμαστε με τη λογική της δομής και λειτουργίας του υπολογιστή (διαδοχή, απόφαση, αποθήκευση - μνήμη κλπ), ενώ στην *αντικειμενοστρεφή σχεδίαση* σκεπτόμαστε με κέντρο το πρόβλημα, χρησιμοποιώντας τα αντικείμενα του λογισμικού για την αναπαράσταση αφηρημένων οντοτήτων του προβλήματος προς την επίλυσή του.

2.2.2 Αντικειμενοστρεφής σχεδίαση

Μεταξύ των μεθοδολογιών ανάλυσης και σχεδίασης πληροφοριακών συστημάτων υπάρχουν δύο που μας ενδιαφέρουν περισσότερο στο κεφάλαιο αυτό.

Η πρώτη, σχετίζεται με το Δομημένο και η δεύτερη με τον Αντικειμενοστρεφή προγραμματισμό. Έτσι, όπως ο Δομημένος προγραμματισμός οδήγησε στην *δομημένη μεθοδολογία σχεδίασης* (structured design), ο Αντικειμενοστρεφής προγραμματισμός οδήγησε στην *αντικειμενοστρεφή μεθοδολογία*

σχεδίασης πληροφοριακών συστημάτων (object-oriented analysis and design).

Ενοποιημένη Γλώσσα Μοντελοποίησης (Unified Modeling Language-UML)

Για τη γραφική απεικόνιση, τον προσδιορισμό, την τεκμηρίωση κ.ά. των στοιχείων και των φάσεων ενός συστήματος λογισμικού χρησιμοποιείται ευρέως η γλώσσα μοντελοποίησης UML. Αποτελείται από ένα σύνολο προσυμφωνημένων όρων, συμβόλων και διαγραμμάτων και κυριαρχεί στη γραφική απεικόνιση της σχεδίασης ενός αντικειμενοστρεφούς συστήματος.

Στην έκδοση 2 διαθέτει πολλούς τύπους διαγραμμάτων, κυρίως δύο κατηγοριών: τους τύπους που σχετίζονται με την αναπαράσταση της δομής (structure) του συστήματος και τους τύπους που σχετίζονται με την αναπαράσταση της συμπεριφοράς (behavior) των δομικών στοιχείων του.

ΕΣΩΤΕΡΙΚΗ ΕΓΚΕΚΡ. ΕΚΔΟΣΗ 1.0

3. Βασικά στοιχεία γλώσσας προγραμματισμού

Εισαγωγή

Στο κεφάλαιο αυτό αναπτύσσονται τα βασικά χαρακτηριστικά του ολοκληρωμένου περιβάλλοντος ανάπτυξης της γλώσσας προγραμματισμού Python. Επίσης αναπτύσσονται οι βασικοί τύποι μεταβλητών, οι βασικές εντολές, οι δομές επιλογής και επανάληψης και οι συναρτήσεις.

Λέξεις κλειδιά

Προγραμματιστικό περιβάλλον ανάπτυξης, λογικές και αριθμητικές εκφράσεις, τελεστές, δομές και τύποι δεδομένων, μεταβλητή, εντολές, συναρτήσεις, διερμηνευτής.

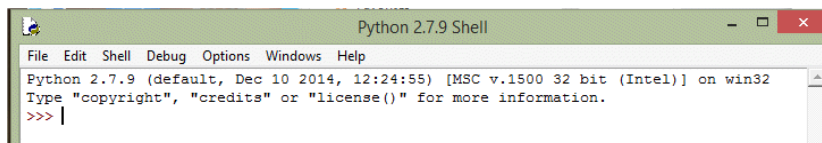
Διδακτικές Ενότητες

Γνωριμία με το ολοκληρωμένο περιβάλλον ανάπτυξης της γλώσσας προγραμματισμού Python

Η γλώσσα προγραμματισμού Python είναι μια σύγχρονη γλώσσα προγραμματισμού που υποστηρίζει τόσο διάφορα προγραμματιστικά υποδείγματα -όπως δομημένο, συναρτησιακό, αντικειμενοστρεφές- όσο και τεχνικές προγραμματισμού. Επιπρόσθετα περιέχει πλούσιες βιβλιοθήκες από έτοιμο κώδικα προγραμματισμού και υποστηρίζει τον *αρθρωτό προγραμματισμό*. Τις βιβλιοθήκες αυτές (μονάδες λογισμικού), αλλά και άλλες επιπρόσθετες εξωτερικές βιβλιοθήκες με έτοιμο κώδικα, μπορούμε να τις χρησιμοποιούμε στα προγράμμά μας ακολουθώντας ένα σύνολο απλών κανόνων και εντολών. Ο τρόπος αυτός μας επιτρέπει τη σχετικά εύκολη συγγραφή πιο σύνθετων προγραμμάτων. Μονάδες λογισμικού - βιβλιοθήκες μπορεί να δημιουργήσει και ένας προγραμματιστής, ώστε να τις χρησιμοποιεί κατάλληλα σε διαφορετικά προγράμμά του, αποφεύγοντας να γράφει, κάθε φορά, κώδικα από την αρχή.

Το **Ολοκληρωμένο Περιβάλλον Ανάπτυξης Προγραμμάτων IDLE** (Integrated DeveLopment Environment) της Python, είναι ένα *Ελεύθερο Λογισμικό/ Λογισμικό Ανοικτού Κώδικα* (ΕΛ/ΛΑΚ), που, εύκολα, μπορούμε να το εγκαταστήσουμε στον υπολογιστή μας. Αρκεί να κατεβάσουμε από το Διαδίκτυο -και στη συνέχεια να

εκτελέσουμε- το κατάλληλο αρχείο, ανάλογα με το λειτουργικό σύστημα του υπολογιστή μας, εφ' όσον είναι διαθέσιμο για διάφορα λειτουργικά συστήματα, όπως MS Windows, Linux / Ubuntu, MAC OS X.



Εικόνα 3-1. Το περιβάλλον IDLE

3.1 Μεταβλητές και τύποι δεδομένων

3.1.1 Τύποι δεδομένων

Ένα πρόγραμμα συνήθως επεξεργάζεται *δεδομένα* τα οποία μπορεί να είναι αποθηκευμένα στην κύρια μνήμη του υπολογιστή, σε αποθηκευτικό μέσο ή η εισαγωγή τους να γίνεται από κάποια μονάδα. Οι *τύποι δεδομένων* προσδιορίζουν τον τρόπο παράστασης των δεδομένων εσωτερικά στον υπολογιστή, καθώς και το είδος της επεξεργασίας τους από αυτόν. Στην Python δε δηλώνουμε ποιο τύπο δεδομένων χρησιμοποιούμε.

Οι χαρακτηριστικοί τύποι δεδομένων στην Python είναι ο αριθμητικός, ο λογικός (boolean) και οι συμβολοσειρές ή αλφαριθμητικά (strings).

Οι **αριθμοί** στην Python είναι κυρίως τριών τύπων:

- ακέραιοι (Integer)
- αριθμοί κινητής υποδιαστολής (floating point)
- μιγαδικοί αριθμοί (complex numbers), τύπος που δε θα μας απασχολήσει στη συνέχεια.

Παραδείγματα

Ο 19, αποτελεί παράδειγμα ακέραιου.

Οι 256.14 και 28.2E-5, όπου το σύμβολο E δηλώνει δύναμη του 10, είναι παραδείγματα αριθμών κινητής υποδιαστολής (ή *floats* για συντομία). Σε αυτή την περίπτωση, το 28.2E-5 σημαίνει $28.2 \cdot 10^{-5}$.

10^{-5} . Παρατηρούμε ότι το δεκαδικό τμήμα διαχωρίζεται με το χαρακτήρα τελεία “.” και όχι το κόμμα “,”.

Ο **λογικός τύπος** (boolean) που δέχεται μόνο δύο τιμές, την τιμή *True* (Αληθής) και την τιμή *False* (Ψευδής) και έχει σκοπό την καταγραφή του αποτελέσματος ενός ελέγχου.

Οι **συμβολοσειρές** είναι μια ακολουθία από χαρακτήρες. Μια συμβολοσειρά μπορεί να αποτελείται από περισσότερες από μία λέξεις. Οι λέξεις μπορούν να είναι σε κάθε γλώσσα που υποστηρίζεται από το πρότυπο Unicode, άρα σε ελληνική, αγγλική κ.ο.κ. Μπορούμε να ορίσουμε μια συμβολοσειρά με μονά εισαγωγικά ή με διπλά, αλλά όχι ανάμικτα. Με ότι ξεκινάμε, θα πρέπει πάλι να κλείνουμε.

Παράδειγμα: “221051445” ή ‘Καλημέρα’.

Για να ελέγξουμε τον **τύπο δεδομένων** χρησιμοποιούμε την εντολή **type ()**.

Παράδειγμα

```
>>> type (1)
<type 'int'>
>>> type (3.14)
<type 'float'>
>>> type (False)
<type 'bool'>
>>> type ('Αρετή')
<type 'str'>
>>>
```

3.2 Αριθμητικές και λογικές πράξεις και εκφράσεις

Χρησιμοποιώντας τιμές κάθε τύπου δεδομένων, μπορούμε να κάνουμε διάφορες πράξεις, χρησιμοποιώντας τους αντίστοιχους τελεστές. Οι **τελεστές** (operators) είναι σύμβολα ή λέξεις για τη δημιουργία αριθμητικών και λογικών εκφράσεων. Οι βασικότεροι τελεστές στη γλώσσα Python είναι:

Αριθμητικοί τελεστές: Είναι τα σύμβολα που χρησιμοποιούμε για να κάνουμε μαθηματικές πράξεις. Στη γλώσσα Python χρησιμοποιούμε τους παρακάτω βασικούς αριθμητικούς τελεστές:

Πρόσθεση	+
Αφαίρεση	-
Πολλαπλασιασμός	*
Διαίρεση	/
Ύψωση σε δύναμη	**
Το υπόλοιπο της ακέραιας διαίρεσης	%

Σε κάθε έκφραση στην οποία υπάρχουν αριθμητικοί τελεστές ακολουθείται μια προσδιορισμένη ιεραρχία πράξεων, που είναι:

1. Ύψωση σε δύναμη.
2. Πολλαπλασιασμός, διαίρεση, υπόλοιπο ακέραιας διαίρεσης.
3. Πρόσθεση, αφαίρεση.

Αν θέλουμε να αλλάξουμε την ιεραρχία των πράξεων, μπορούμε να χρησιμοποιήσουμε παρενθέσεις. Για παράδειγμα, στην έκφραση $(2+3)*5$ θα εκτελεστεί πρώτα η πρόσθεση μέσα στην παρένθεση και μετά, το αποτέλεσμα θα πολλαπλασιαστεί επί 5, σε αντίθεση με την έκφραση $2+3*5$ στην οποία, πρώτα θα γίνει ο πολλαπλασιασμός και μετά η πρόσθεση.

Σχεσιακοί (ή συγκριτικοί) τελεστές: χρησιμοποιούνται για τη σύγκριση δύο τιμών ή μεταβλητών, με το αποτέλεσμα μιας σύγκρισης να είναι είτε True (Αληθής) είτε False (Ψευδής). Στη γλώσσα Python χρησιμοποιούνται οι παρακάτω βασικοί σχεσιακοί τελεστές:

Μικρότερο από	<
Μικρότερο ή ίσο από	<=
Μεγαλύτερο από	>
Μεγαλύτερο ή ίσο από	>=
Ίσο με	==
Διάφορο από	!=

Τελεστές λογικών πράξεων: Στις λογικές πράξεις και εκφράσεις χρησιμοποιούνται οι λογικοί τελεστές not (ΟΧΙ), and (ΚΑΙ), or (Ή) με τις ακόλουθες λογικές λειτουργίες:

- not (ΟΧΙ): πράξη άρνησης
- and (ΚΑΙ): πράξη σύζευξης
- or (Η): πράξη διάζευξης.

Το αποτέλεσμα μιας λογικής πράξης είναι True (Αληθής) ή False (Ψευδής) σύμφωνα με τον παρακάτω πίνακα:

P	Q	P and Q	P or Q	Not P
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Παραδείγματα

Πίνακας 3-1. Αριθμητικές και λογικές εκφράσεις

Αριθμητικές πράξεις	Αποτελέσματα
>>> 2+8*2	18
>>> 2**2+4/2-3*4	-6
>>> 45 / 10	4 # Στην Python, η διαίρεση ακεραίων αριθμών μας επιστρέφει το ακέραιο πηλίκο
>>> 45 % 10	5 # μας επιστρέφει το υπόλοιπο της ακέραιας διαίρεσης 45 / 10
>>> 45.0 /10	4.5 # Η διαίρεση αριθμών κινητής υποδιαστολής, μας επιστρέφει το πηλίκο, ως αριθμό κινητής υποδιαστολής

Λογικές εκφράσεις με συγκριτικούς τελεστές	Αποτελέσματα
>>> 23 ==23	True
>>> 34 != 45	True
>>> 56<=12	False
Λογικές εκφράσεις με τελεστές λογικών πράξεων	Αποτελέσματα
>>> (12<11) and (23>10)	False
>>> (12<11) or (23>10)	True
>>> not(56<=12)	True

Μεταβλητές

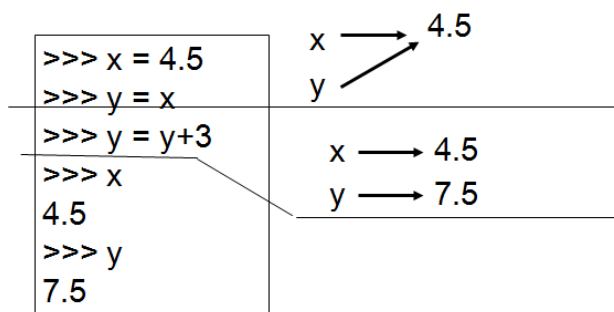
Η γλώσσα Python παρέχει εντυπωσιακές εναλλακτικές δυνατότητες για τη διαχείριση μεταβλητών που διευκολύνουν τον προγραμματιστή. Για τη χρησιμοποίηση μιας μεταβλητής δεν απαιτείται η δήλωσή της, ενώ μπορεί να εκχωρήσουμε διαφορετικούς τύπους τιμών σε μια μεταβλητή κατά τη διάρκεια ενός προγράμματος, όπως ακέραιες τιμές, κινητής υποδιαστολής και συμβολοσειρές.

Για να χρησιμοποιήσουμε μια μεταβλητή απαιτείται να της δώσουμε ένα όνομα και στη συνέχεια να της εκχωρήσουμε κάποια τιμή. Στη γλώσσα Python υπάρχουν ορισμένοι κανόνες που πρέπει να ακολουθούμε σχετικά με το όνομα μιας μεταβλητής. Έτσι, για παράδειγμα, δεν επιτρέπεται να ξεκινά το όνομα μιας μεταβλητής με αριθμό και το όνομα που θα δώσουμε δεν πρέπει να είναι όμοιο με κάποιο όνομα ενσωματωμένης συνάρτησης ή

εντολής. Συνηθίζουμε να δίνουμε ένα όνομα σχετικό με το είδος της μεταβλητής με λατινικούς χαρακτήρες, που μπορεί να συνοδεύεται από κάποιον αριθμό, όπως για παράδειγμα: `onoma1`, `onoma_2`, `timi`, `mesi_timi`, `embado` κ.ά.

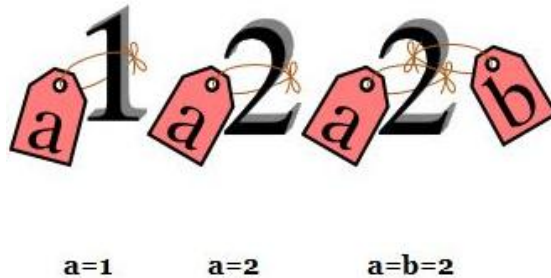
Για να *εκχωρήσουμε* μια τιμή σε μια μεταβλητή, χρησιμοποιούμε τον τελεστή ίσον "=", όπως για παράδειγμα `a = 125`.

Στο αριστερό μέρος δίνουμε το όνομα της μεταβλητής, στη συνέχεια χρησιμοποιούμε τον τελεστή εκχώρησης "=" και στο δεξιό μέρος βάζουμε την τιμή, μια άλλη μεταβλητή ή μια έκφραση που έχει ως αποτέλεσμα μια τιμή.



Προσοχή! ο τελεστής "=" δεν έχει τη σημασία που έχει το σύμβολο της ισότητας, όπως το χρησιμοποιούμε στα μαθηματικά. Στις περισσότερες γλώσσες προγραμματισμού, ο τελεστής εκχώρησης τιμής "=" σημαίνει ότι εκχωρούμε στη μεταβλητή μια τιμή ενός τύπου δεδομένου και ταυτόχρονα την εισάγουμε στη θέση μνήμης που αντιστοιχεί στη μεταβλητή με αυτό το όνομα. Για παράδειγμα, η εντολή `a = 125` σημαίνει ότι η μεταβλητή με όνομα `a` αντιστοιχεί σε μια θέση μνήμης του υπολογιστή και η ακέραια τιμή 125 εισάγεται στη θέση αυτή. Αν στη συνέχεια θέσουμε `a = 250`, στη θέση μνήμης που αναφέρεται η μεταβλητή `a` εισάγεται η νέα ακέραια τιμή 250 και η παλαιά τιμή 125 παύει πλέον να υπάρχει, αφού έχει αντικατασταθεί από τη 250.

Όλα τα δεδομένα σε ένα πρόγραμμα Python αναπαρίστανται με αντικείμενα ή με σχέσεις μεταξύ των αντικειμένων, με κάθε αντικείμενο να έχει μια ταυτότητα (identity), έναν τύπο και μία τιμή. Για παράδειγμα, το 12 είναι ένα αντικείμενο με τιμή 12, τύπου `int` (ακέραιος).



Οι μεταβλητές στην Python λειτουργούν σαν ετικέτες με κάποιο όνομα, που χρησιμεύουν για να αναφερόμαστε (ή αλλιώς για να δείχνουμε) σε κάποια αντικείμενα. Ο τύπος της μεταβλητής είναι ο εκάστοτε τύπος του αντικειμένου στην οποία αναφέρεται. Στο παράδειγμα `a = 125`, δημιουργείται η μεταβλητή με όνομα `a` και αναφέρεται στο αντικείμενο, με τιμή 125, με ακέραιο τύπο δεδομένων. Το σύμβολο `"="` δημιουργεί ένα είδος δεσίματος μεταξύ του αντικειμένου 125 και της μεταβλητής με όνομα `a`. Ο τύπος της μεταβλητής είναι και αυτός ακέραιος, αφού αναφέρεται σε αντικείμενο με ακέραιο τύπο δεδομένων.

Όταν θέσουμε στη συνέχεια `a = 250.0`, η μεταβλητή `a` παύει πλέον να δείχνει το αντικείμενο με τιμή 125 και δημιουργείται ένα νέο "δέσιμο", ώστε να αναφέρεται στο αντικείμενο με τιμή 250.0 (κινητής υποδιαστολής τύπος δεδομένων). Ο τύπος δεδομένων της μεταβλητής `a` τώρα έχει αλλάξει και είναι κινητής υποδιαστολής, όμοιος με τον τύπο του νέου αντικειμένου στο οποίο αναφέρεται.

Η Python παρακολουθεί όλες τις τιμές και τις διαγράφει όταν πάψουν να υπάρχουν μεταβλητές που να αναφέρονται σε αυτές. Η διαδικασία αυτή ονομάζεται *συλλογή σκουπιδιών* (garbage collection).

Στο παράδειγμα που ακολουθεί η μεταβλητή `a` παίρνει ως τιμή διαδοχικά ένα ακέραιο, ένα πραγματικό, μια λογική τιμή και τέλος μια συμβολοσειρά. Κάθε φορά ελέγχεται ο τύπος δεδομένων με την εντολή `type`.

```
>>> a = 125
>>> type (a)
<type 'int'>
>>> a = 250.7
>>> type (a)
<type 'float'>
>>> a = True
>>> type (a)
<type 'bool'>
>>> a = 'Καλημέρα'
>>> type (a)
<type 'str'>
```

Αν θέλουμε να εμφανίσουμε το περιεχόμενο της μεταβλητής, τότε μπορούμε να χρησιμοποιήσουμε την εντολή **print** μαζί με το όνομα της μεταβλητής. Για παράδειγμα: `message= 'Καλημέρα'`, η εντολή `print message` θα εμφανίσει στην οθόνη τη λέξη Καλημέρα. Στο όνομα της μεταβλητής `message`, που ακολουθεί μετά την `print`, δεν πρέπει να χρησιμοποιηθούν εισαγωγικά (μονά ή διπλά), διότι τότε θα εμφανιστεί στην οθόνη η συμβολοσειρά `message`. Παρόμοια, αν θέσουμε `a = 1234`, η `print a` θα εμφανίσει την τιμή 1234 στην οθόνη.

Βασικές εντολές στη γλώσσα Python

Στις προηγούμενες παραγράφους παρουσιάστηκαν μερικές από τις βασικές εντολές (statements) στην Python, όπως:

`print`: χρησιμοποιείται για την εμφάνιση τιμών στην οθόνη του υπολογιστή, με ποικίλες μορφές, όπως:

- `print` όνομα_μεταβλητής, π.χ. `print name`
- `print` αριθμός, π.χ. `print 1045.34`
- `print` 'συμβολοσειρά', π.χ. `print 'θαλασσινός αέρας'`
- `print` όνομα_μεταβλητής (τελεστής) αριθμός

```
>>> name = 'kostas'
>>> print name
Kostas
>>> print 1045.34
1045.34
>>> message = "Τώρα "
>>> print message*3
Τώρα Τώρα Τώρα
```

Εκχώρηση τιμής σε μια μεταβλητή: χρησιμοποιείται το σύμβολο "=" για την εκχώρηση μιας τιμής σε μια μεταβλητή.

```
όνομα_μεταβλητής = τιμή μεταβλητής
tree = 'Πεύκο'
x = 135
x = x **2+ 34
```

Εισαγωγή τιμής σε μια μεταβλητή από το πληκτρολόγιο. Το πρόγραμμα αναμένει από το χρήστη να πληκτρολογήσει μια τιμή. Η τιμή που εισάγεται αποδίδεται σε μια μεταβλητή. Προς τούτο χρησιμοποιούνται δύο εντολές: η **input()** και η **raw_input()**.

Σύνταξη: Όνομα μεταβλητής= input('μήνυμα').

```
>>> number = input(' Δώσε έναν αριθμό:')
Δώσε έναν αριθμό: 34
>>> print number
34
```

Αν θέλουμε να εισάγουμε ένα αλφαριθμητικό (ή αριθμό με τη μορφή αλφαριθμητικού), χρησιμοποιούμε τη **raw_input()**:

Σύνταξη: Όνομα μεταβλητής = raw_input ()

```
>>>name = raw_input('Δώσε το όνομά σου : ')
Δώσε το όνομά σου : Kostas
>>> print name
Kostas
>>> tree = raw_input('Δώσε το όνομα ενός δέντρου με άνθη
που βλέπεις πηγαίνοντας στο σχολείο: ')
Δώσε το όνομα ενός δέντρου με άνθη που βλέπεις
πηγαίνοντας στο σχολείο: Αμυγδαλιά
>>> print 'Το δέντρο ',tree,'είναι όμορφο ανθισμένο'
Το δέντρο Αμυγδαλιά είναι όμορφο ανθισμένο
```

Γενικά, ότι εισάγεται με τη raw_input θεωρείται αυτόματα αλφαριθμητικό, ενώ η input προσπαθεί να το υπολογίσει. Για παράδειγμα, αν δώσουμε στην input το όνομα μιας μεταβλητής, θα επιστρέψει το περιεχόμενο της μεταβλητής.

Εισαγωγή σχολίων

Τα σχόλια σε ένα πρόγραμμα διευκολύνουν την κατανόησή του. Στην Python, τα σχόλια εισάγονται θέτοντας μπροστά από αυτά το σύμβολο # . Τα σχόλια μπορούν να αρχίζουν και μετά από εντολές στη μέση μιας γραμμής. Ό,τι βρίσκεται δεξιά από το #, αγνοείται από το διερμηνευτή.

```
#Πρόγραμμα πολλαπλασιασμός δύο αριθμών
>>> x=input('Δώσε τον πρώτο αριθμό: ')
Δώσε το πρώτο αριθμό: 34
>>> y=input('Δώσε το δεύτερο αριθμό: ')
Δώσε το δεύτερο αριθμό: 2
>>> ginomeno = x*y #πολλαπλασιάζει τις τιμές των x και y και
το αποτέλεσμα το εκχωρεί στη μεταβλητή ginomeno
>>> print ginomeno
68
```

3.3 Βασικές (ενσωματωμένες) συναρτήσεις

Η Python παρέχει μια ποικιλία ενσωματωμένων συναρτήσεων για τη μετατροπή τιμών δεδομένων από έναν τύπο σε έναν άλλο, όπως οι: `int()`, `float()` και `str()`.

Δραστηριότητα εμπέδωσης

Στο περιβάλλον της γλώσσας Python, επαληθεύστε τις παρακάτω συναρτήσεις:

- Η **`float()`** μετατρέπει ακραίους και συμβολοσειρές σε δεκαδικούς αριθμούς.
- Η **`int()`** δέχεται οποιαδήποτε αριθμητική τιμή και τη μετατρέπει σε ακέραιο κόβοντας τα δεκαδικά ψηφία, αν υπάρχουν.
- Η **`str()`** δέχεται οποιαδήποτε τιμή και την μετατρέπει σε συμβολοσειρά.
- Η **`abs()`** επιστρέφει την απόλυτη τιμή ενός αριθμού.
- Η **`pow(a,b)`** επιστρέφει τη δύναμη του `a` υψωμένη στο `b`.
- Η **`divmod(x,y)`** επιστρέφει το ακέραιο πηλίκο και το ακέραιο υπόλοιπο της διαίρεσης `x/y`.

```
>>> float(10)
10.0
>>> int(5.678)
5
>>> str(5.678)
5.678
>>> abs(-45)
45
>>> divmod(10,3)
(3, 1)
>>> pow(2,3)
8
```

Αν θέλουμε να διαβάσουμε από το πληκτρολόγιο έναν ακέραιο αριθμό με τη συνάρτηση `input()`, πρέπει να χρησιμοποιούμε και τη συνάρτηση `int()`

```
a=int(input('Δώσε έναν αριθμό :'))
```

Παράδειγμα

```
>>> a=int(input('Δώσε έναν ακέραιο αριθμό: '))  
Δώσε έναν ακέραιο αριθμό: 2345.10    #ο χρήστης δίνει την τιμή  
2345.10  
>>> print a  
2345    #εμφανίζεται η ακέραια τιμή του αριθμού αποκόπτοντας  
τα δεκαδικά ψηφία
```

Δραστηριότητα

Συμβουλευτείτε τη «βοήθεια» του προγραμματιστικού περιβάλλοντος και πειραματιστείτε με τις ενσωματωμένες συναρτήσεις που παρέχει η γλώσσα Python.

Εξωτερικές βιβλιοθήκες

Εκτός από τις ενσωματωμένες συναρτήσεις οι οποίες περιλαμβάνονται στη γλώσσα Python, μπορεί κανείς να βρει πληθώρα εξωτερικών βιβλιοθηκών (αρθρώματα ή modules) στους Διαδικτυακούς τόπους υποστήριξης της γλώσσας. Μια βιβλιοθήκη είναι ένα αρχείο το οποίο περιέχει μια συλλογή από σχετικές συναρτήσεις.

Ενδεικτικά, μπορούν να αναφερθούν: η μαθηματική βιβλιοθήκη και οι βιβλιοθήκες γραφικών. Οι βιβλιοθήκες αυτές για να χρησιμοποιηθούν θα πρέπει να εισαχθούν στο πρόγραμμά μας. Η εισαγωγή αυτή γίνεται με την εντολή **import**. Για παράδειγμα, προκειμένου να χρησιμοποιήσουμε μαθηματικές συναρτήσεις σε ένα πρόγραμμα, θα πρέπει μία από τις αρχικές εντολές του προγράμματός μας να είναι: `import math`.

Για να έχουμε πρόσβαση σε μια από τις συναρτήσεις, θα πρέπει να δηλώσουμε το όνομα της μονάδας και το όνομα της συνάρτησης, χωρισμένα με μια τελεία, μορφή που ονομάζεται *συμβολισμός με τελεία* (dot notation). Ας δούμε ένα παράδειγμα συνάρτησης για την τετραγωνική ρίζα, `sqrt()`.

```
>>> import math
>>> riza = math.sqrt(2)
>>> print riza
1.41421356237
>>> math.sqrt(3)
1.7320508075688772
>>> x = math.pi
>>> print x
3.14159265359
```

Σε επόμενο κεφάλαιο θα δούμε διάφορα παραδείγματα από έτοιμες βιβλιοθήκες λογισμικού για την υλοποίηση ελκυστικών προγραμμάτων με χρήση γραφικών, καθώς και τη δημιουργία δικών μας.

3.4 Δομή προγράμματος και καλές πρακτικές

Μερικές καλές πρακτικές και συντακτικές συμβάσεις που πρέπει να τηρούμε κατά τη συγγραφή ενός προγράμματος, είναι οι παρακάτω:

- Να δίνουμε ένα χαρακτηριστικό τίτλο στο πρόγραμμα με τη μορφή σχολίων, τα οποία ξεκινάνε με το σύμβολο `#`. Αυτό, παρότι δεν είναι αναγκαίο, είναι ιδιαίτερα χρήσιμο.
- Να προσέχουμε τα κενά διαστήματα πριν την κάθε εντολή, καθώς, όπως θα δούμε στο επόμενο κεφάλαιο, η Python βασίζεται σε αυτά, για να ορίσει ομάδες εντολών.
- Να επιλέγουμε τους κατάλληλους τελεστές.
- Να χρησιμοποιούμε τα ίδια εισαγωγικά (μονά εισαγωγικά με μονά, διπλά εισαγωγικά με διπλά) για μια συμβολοσειρά.
- Να προσθέτουμε, όπου κρίνουμε χρήσιμο, επεξηγηματικά σχόλια μέσα στον κώδικα. Θυμίζουμε ότι τα σχόλια περιγράφουν τη λειτουργία ενός προγράμματος ή γενικότερα, ενός τμήματος κώδικα και γράφονται, για να βοηθήσουν τους ανθρώπους -και όχι τον υπολογιστή- στην κατανόηση και συντήρηση ενός προγράμματος. Όταν ένα

πρόγραμμα μεταφράζεται, τα σχόλια αγνοούνται. Τα σχόλια που περιγράφουν τη λειτουργία ενός προγράμματος, το ρόλο των μεταβλητών και τη λειτουργία πολύπλοκων τμημάτων κώδικα, αποτελούν παράδειγμα καλού προγραμματιστικού στυλ.

- Να δίνουμε ονόματα μεταβλητών που έχουν σχέση με τη χρήση τους.

3.5 Τύποι και δομές δεδομένων στις γλώσσες προγραμματισμού

Στην παράγραφο αυτή θα παρουσιάσουμε τους *Τύπους* και τις *Δομές Δεδομένων* στη γενική τους μορφή, όπως ορίζονται στις περισσότερες Γλώσσες Προγραμματισμού. Γίνεται σε συνέχεια της εξειδίκευσής τους στην γλώσσα προγραμματισμού Python που προηγήθηκε.

Δραστηριότητα

Μελετήστε, συμπληρώστε ή επεκτείνετε όπου πρέπει, τα παρακάτω σχετικά με όρους των προηγούμενων κεφαλαίων.

Τύποι και Δομές Δεδομένων

Εννοιολογικοί προσδιορισμοί

Ένας *Τύπος Δεδομένων* (Data Type) είναι ένα σύνολο τιμών δεδομένων και λειτουργιών επί αυτών των τιμών. Οι τύποι δεδομένων είτε είναι προκαθορισμένοι από τις γλώσσες προγραμματισμού και καλούνται *Πρωτογενείς Τύποι Δεδομένων* (Primitive Data Types) είτε δημιουργούνται από τον προγραμματιστή, οπότε καλούνται *Μη Πρωτογενείς τύποι δεδομένων* (Non-Primitive Data Types). Διακρίνονται δε, ανάλογα με τη σύσταση των μερών που τους αποτελούν, σε: *Απλούς* τύπους και *Σύνθετους* τύπους δεδομένων.

Απλοί τύποι δεδομένων

Στους απλούς τύπους δεδομένων, οι τιμές των δεδομένων είναι στοιχεία μη περαιτέρω-χωριζόμενα (άτομα). Δηλαδή κάθε δεδομένο έχει μία και μοναδική τιμή.

Σύνθετοι

Σύνθετος τύπος δεδομένων (Composite Data Type) είναι εκείνος, που αποτελείται από Πρωτογενείς ή / και άλλους σύνθετους τύπους, όπου μια μεταβλητή μπορεί να πάρει ως τιμή μια ενότητα τιμών. Οι σύνθετοι τύποι καλούνται και **Δομές Δεδομένων**. Παραδείγματα σύνθετων τύπων είναι η Εγγραφή, το Σύνολο, ο Πίνακας.

Συνήθεις τύποι σε γλώσσες προγραμματισμού**Απλοί**

- **Ακέραιος** (Integer), με τιμές τους ακέραιους αριθμούς μέσα σε ένα κάτω και ένα άνω όριο και πράξεις: +, -, *, /, mod, div, := καθώς και οι συγκρίσεις >, =, <
- **Πραγματικός** (real).
- **Χαρακτήρας** (Character), με τιμές από το σύνολο των χαρακτήρων που διαθέτει ο υπολογιστής.
- **Λογικός** (Boolean), για την αναπαράσταση Λογικών δεδομένων, με τιμές True (σωστό), False (λάθος) και επιτρεπτές πράξεις για τις τιμές αυτές τις: and, or, not.
- **Αλφαριθμητικός-string**.

Σύνθετοι Τύποι Δεδομένων

- **Πίνακας** (Array). Συνήθως τα στοιχεία του καθορίζονται με τη βοήθεια δεικτών. Μπορεί να είναι πολλών διαστάσεων.
- **Εγγραφή** (Record/tuple/struct).
- **Λίστα** (List).
- **Σύνολο** (Set).
- **Σωρός** (Heap).
- **Στοιβά** (Stack).
- **Ουρά** (Queue).
- **Δένδρο** (Tree).
- **Γράφος** (Graph).

Κατηγοριοποίηση**Επίπεδα αφαίρεσης σε Τύπους Δεδομένων**

Ένας τύπος δεδομένων αρχικά και στο στάδιο σχεδίασης, υπάρχει μόνον διανοητικά, ορίζοντας έτσι έναν *αφηρημένο τύπο δεδομένων*.

ΑΤΔ (Abstract Data Type). Στο στάδιο αυτό έχει οριστεί με βάση τις λειτουργίες που επιτελεί, κρατώντας κρυφή την εκτέλεση. Οι περισσότερες αντικειμενοστρεφείς γλώσσες προγραμματισμού παρέχουν τη δυνατότητα να καθορίζονται από το χρήστη τύποι αφηρημένων στοιχείων. Δεν μας απασχολεί, στο επίπεδο αυτό, ούτε ο τρόπος με τον οποίο θα αναπαρασταθούν τα δεδομένα ούτε ο τρόπος που θα υλοποιηθούν οι λειτουργίες σε κώδικα (code implementation). Μας ενδιαφέρει δηλαδή, το τι θα υπολογιστεί και όχι το πώς. Με την παραπάνω λογική οι Σύνθετοι Τύποι Δεδομένων μπορούν να χαρακτηριστούν ως ΑΤΔ. Έτσι, για παράδειγμα, μια γέφυρα είναι δυνατόν είτε να την βλέπουμε ως μια οντότητα (υψηλό επίπεδο αφάιρεσης) είτε ως πολλά τμήματα -δοκοί, στερεωτικά κ.ά., όπως θα την έβλεπε ένας Μηχανικός.

ΕΣΩΤΕΡΙΚΗ ΕΓΚΕΚΡ. ΕΚΔΟΣΗ 1.0

4. Αλγοριθμικές δομές

Εισαγωγή

Στο κεφάλαιο αυτό γίνεται η εμβάθυνση σε βασικές έννοιες, αλγοριθμικές δομές και τεχνικές προγραμματισμού, καθώς και στην αξιοποίησή τους για την επίλυση προβλημάτων με προγραμματισμό, σε γλώσσα Python.

Λέξεις κλειδιά

Αλγοριθμικές δομές, δομή ακολουθίας, δομή επιλογής, δομή επανάληψης, συναρτήσεις.

Διδακτικές Ενότητες

4.1 Αλγοριθμικές δομές - Ροές εκτέλεσης προγράμματος

4.1.1 Δομή ακολουθίας

Πρόκειται για μια σειρά από εντολές που εκτελούνται η μία μετά την άλλη με τη σειρά. Η δομή ακολουθίας χρησιμοποιείται πρακτικά για την επίλυση απλών προβλημάτων, όπου είναι δεδομένη η σειρά εκτέλεσης ενός συνόλου ενεργειών.

Χρησιμοποιώντας, αποκλειστικά, τη δομή ακολουθίας, μπορούμε να λύσουμε περιορισμένα προβλήματα στα οποία:

- η σειρά των βημάτων είναι καθορισμένη
- όλα τα βήματα εκτελούνται πάντοτε
- δεν υπάρχουν εξαιρέσεις.

Παράδειγμα

Να γραφεί πρόγραμμα που να υλοποιεί τον αλγόριθμο: εμβαδόν ορθογωνίου παραλληλογράμμου σε γλώσσα Python.

The image shows two windows. The top window is a text editor titled 'εμβαδον.py - /home/administrator/εμβαδον.py'. It contains the following Python code:

```
# Εμβαδόν ορθογωνίου Παραλληλογράμμου
BASH = input('Δώσε το μήκος της βάσης :')
YPSOS = input('Δώσε το μήκος του ύψους :')
EMBADO = BASH * YPSOS
print 'Το Εμβαδόν του ορθογωνίου παραλληλογράμμου είναι: ', EMBADO
```

The bottom window is a 'Python Shell' titled 'Python Shell'. It shows the execution of the script:

```
Python 2.7.3 (default, Dec 18 2014, 19:03:52)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>>
Δώσε το μήκος της βάσης :20
Δώσε το μήκος του ύψους :15
Το Εμβαδόν του ορθογωνίου παραλληλογράμμου είναι: 300
>>> |
```

4.1.2 Δομή επιλογής if (AN)

Η δομή επιλογής if (AN) χρησιμοποιείται, όταν θέλουμε να εκτελεστεί μια ακολουθία εντολών, μόνον, εφόσον πληρείται μία συγκεκριμένη συνθήκη. Για να επιλύσουμε πιο σύνθετα προβλήματα, πρέπει να είμαστε σε θέση να δημιουργούμε στον αλγόριθμο λογικά μονοπάτια, εξετάζοντας απλά λογικά ερωτήματα για την εκτέλεση ή όχι μιας ομάδας εντολών.

Απλή δομή επιλογής

Αν η συνθήκη είναι αληθής, τότε το σύνολο των εντολών που περιέχονται στην δομή if, θα εκτελεστούν. Αλλιώς, η ροή του προγράμματος θα προσπεράσει τη δομή if και θα συνεχίσει από την εντολή που βρίσκεται αμέσως μετά το τέλος της if.

if <συνθήκη ελέγχου>:

εντολές που θα εκτελεστούν αν ισχύει (αληθής-True) η συνθήκη ελέγχου

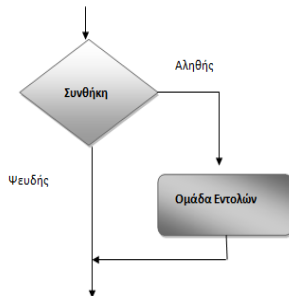
Παράδειγμα

```
#πρόγραμμα εμφάνισης προειδοποίησης αρνητικού ποσού.
```

```
a = input('Δώσε ένα ποσό')
```

```
if a < 0:
```

```
    print "ΠΡΟΣΟΧΗ το ποσό είναι αρνητικό"
```



Εικόνα 4-1. Διάγραμμα ροής για τη δομή επιλογής **if...else** (AN...ΑΛΛΙΩΣ)

Σύνθετη δομή επιλογής

Αν, ανάλογα με την αποτίμηση μιας συνθήκης, θέλουμε να εκτελεστούν διαφορετικές ακολουθίες εντολών, τότε μπορούμε να χρησιμοποιήσουμε τη δομή επιλογής **if...else** (AN...ΑΛΛΙΩΣ).

Αν ισχύει η συνθήκη (έχει τιμή TRUE), θα εκτελεστεί η A ομάδα εντολών της if, αλλιώς (αν δεν ισχύει-έχει τιμή FALSE), θα εκτελεστεί η B ομάδα εντολών της else.

Η εντολή ελέγχου **if else** (AN_ΑΛΛΙΩΣ) συντάσσεται:

```
if <συνθήκη ελέγχου>:
```

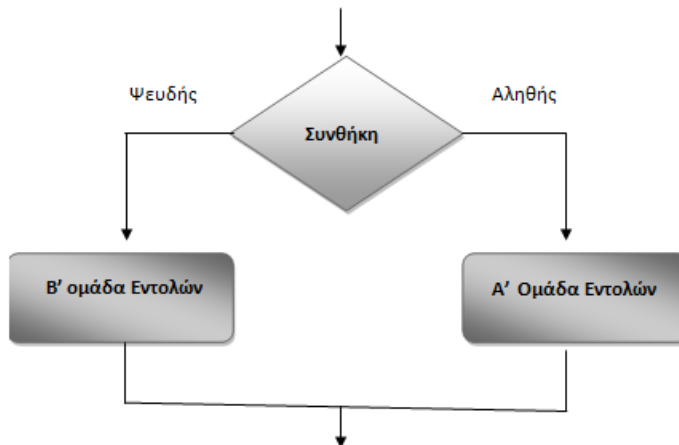
```
    #εντολές που θα εκτελεσθούν, αν η συνθήκη ελέγχου είναι αληθής
```

```
else:
```

```
    #εντολές που θα εκτελεσθούν αν η συνθήκη ελέγχου είναι ψευδής
```

Σημείωση: Οι ομάδες εντολών που θα εκτελεστούν, αν ισχύει μια συνθήκη, ορίζονται ως ένα μπλοκ με τη χρήση εσοχής (κενά). Οι διαδοχικές εντολές που έχουν την ίδια εσοχή ανήκουν στο ίδιο

μπλοκ. Οι εσοχές μπαίνουν αυτόματα, αν πατήσουμε Enter μετά το σύμβολο της άνω και κάτω τελείας “:”. Δεν πρέπει να διαγράψουμε αυτά τα κενά διαστήματα.



Δραστηριότητα: Δομή επιλογής ΑΝ_ΑΛΛΙΩΣ

Να γραφεί αλγόριθμος και πρόγραμμα σε Python που να διαβάζει τις ηλικίες δύο ατόμων και να τις καταχωρεί στις μεταβλητές A και B αντίστοιχα. Στη συνέχεια, να συγκρίνει τις ηλικίες των δύο ατόμων και, αν η ηλικία του A είναι μεγαλύτερη του B, να εμφανίζει στην οθόνη το μήνυμα: "Η ηλικία A είναι μεγαλύτερη από τη B". Διαφορετικά, να εμφανίζει στην οθόνη το μήνυμα "Η ηλικία A είναι μικρότερη ή ίση ($\leq$) της ηλικίας B".

Πρόγραμμα σε Python

```
# Σύγκριση ενός αριθμού A ως προς έναν αριθμό B.  
a = input("Δώσε την ηλικία A του πρώτου ατόμου: ")  
b = input("Δώσε την ηλικία B του δεύτερου ατόμου:")  
if a > b:  
    print "Η ηλικία A είναι μεγαλύτερη από τη B"  
else:  
    print "Η ηλικία A είναι μικρότερη ή ίση ( $\leq$ ) της ηλικίας B"
```

Πολλαπλή Επιλογή: Η Python προσφέρει τη δυνατότητα για σύνταξη σύνθετων δομών επιλογής με τη χρήση της εντολής **elif**. Η σύνταξη είναι η εξής:

```
if <συνθήκη>:  
    <εντολές>  
elif <συνθήκη2>:  
    <εντολές_2>  
else:  
    <εντολές_3>
```

Δραστηριότητα: Πολλαπλή επιλογή

Να γραφεί πρόγραμμα που να διαβάζει τον κωδικό συναγερμού (υποθέτουμε ότι είναι αριθμοί από 1 έως 50 που αντιστοιχούν σε δυνατά σημεία ελέγχου ενός σπιτιού) και να τυπώνει ανάλογο μήνυμα σχετικά με τη ζώνη που τον προκάλεσε. Υποθέτουμε ότι στη ζώνη 1 ανήκουν οι κωδικοί 1 έως και 10, στη Ζώνη 2 οι κωδικοί 11 έως και 20 και στη ζώνη 3 οι μεγαλύτεροι του 20.

```
#Πρόγραμμα εντοπισμού ζώνης συναγερμού  
alarm = input("Δώσε έναν κωδικό συναγερμού: ")  
if alarm > 20:  
    z=3  
elif alarm>10:  
    z=2  
else:  
    z=1  
print "Πρόβλημα από τη ζώνη: ",z  
print "τέλος"
```

Εμφωλευμένες δομές επιλογής: Οι πολλαπλές επιλογές μπορούν να υλοποιηθούν και με εμφωλευμένες δομές Αν-αλλιώς, όπως στη δραστηριότητα που ακολουθεί.

Δραστηριότητα: Δομή επιλογής - εμφωλευμένες δομές επιλογής

A) Μελετήστε το παρακάτω πρόγραμμα.

- Τι θα τυπώσει το παρακάτω πρόγραμμα για τιμές 1, 7, 9, 16;
- Τι πιστεύετε ότι κάνει το πρόγραμμα;

B) Να τροποποιηθεί κατάλληλα το παρακάτω πρόγραμμα, ώστε να περιέχει και την περίπτωση που ο χρήστης δώσει, κατά λάθος, ως βαθμό έναν αρνητικό αριθμό.

```
# Πρόγραμμα ελέγχου υπερϊώδους ακτινοβολίας
# Εισαγωγή τιμής δείκτη ακτινοβολίας από το χρήστη
# Δεδομένα: Ο Δείκτης UV αποτελεί διεθνώς ένα μέγεθος
έκφρασης της επικινδυνότητας της ηλιακής υπερϊώδους
ακτινοβολίας που φθάνει στο έδαφος
# Όρια κινδύνου: 0-5.9 ελάχιστος ή μικρός, 6-10.9 Μεγάλος-
Πολύ μεγάλος, >=11 <=15 Ακραία κατάσταση
uv = input(' Παρακαλώ δώσε έναν αριθμό έκθεσης δείκτη UV: ')
if uv<=15:
# Εμφωλευμένο if
    if uv>= 11:
        print ' Ακραία κατάσταση κινδύνου '
    else:
        if uv>= 6:
            print ' Μεγάλος ή Πολύ μεγάλος κίνδυνος '
            # για 6<= uv < 11
        else:
            if uv>= 0:
```

```
print 'Ελάχιστος ή μικρός κίνδυνος'
# για un 0 <= un < 6

else:
# για un >15
print 'Ο αριθμός που έδωσες υπερβαίνει το 15. Μη αποδεκτή
τιμή!'
```

4.1.3 Δομή επανάληψης (for και while)

Συχνά σε ένα πρόγραμμα, μια ομάδα εντολών είναι αναγκαίο να εκτελείται περισσότερες από μία φορές. Υπάρχουν δύο τύποι επαναλήψεων:

- Οι προκαθορισμένοι, όπου το πλήθος των επαναλήψεων είναι δεδομένο, πριν αρχίσουν οι επαναλήψεις. Για παράδειγμα: ο υπολογισμός του μέσου όρου βαθμολογίας των μαθητών ενός τμήματος 22 μαθητών.
- Οι μη προκαθορισμένοι, όπου το πλήθος των επαναλήψεων καθορίζεται κατά τη διάρκεια της εκτέλεσης των εντολών του σώματος της επανάληψης. Για παράδειγμα: ο υπολογισμός των μορίων όσων υποβάλλουν αίτηση σε ένα διαγωνισμό του Δημοσίου.

Στην γλώσσα προγραμματισμού Python, χρησιμοποιούμε την εντολή **for** για να εκτελεστεί ένα τμήμα του κώδικα για έναν καθορισμένο αριθμό επαναλήψεων, ενώ την εντολή **while** για να εκτελείται υπό συνθήκη και μάλιστα, όσο αυτή είναι αληθής.

Στην εντολή **for** χρησιμοποιείται η συνάρτηση **range()** για τον καθορισμό των επαναλήψεων.

```
for onoma_metavlitis in range (αρχή, μέχρι, βήμα):
    Εντολή_1
    Εντολή_2
    .....
    Εντολή_v
```

Δραστηριότητα: Χρήση της εντολής for

Να γραφεί αλγόριθμος και το αντίστοιχο πρόγραμμα σε γλώσσα Python που να υπολογίζει το άθροισμα των περιπτών αριθμών από το 1 έως και το 100.

Πρόγραμμα σε Python

```
# Πρόγραμμα αθροίζω περιπτούς
athroisma = 0
for i in range(1,100,2):
    athroisma = athroisma + i
print ' Το αποτέλεσμα είναι ', athroisma
```

Προσεγγίζοντας τη συνάρτηση range

Η range() είναι μια ενσωματωμένη συνάρτηση της γλώσσας Python, η οποία, ανάμεσα σε άλλα, χρησιμοποιείται για την υπόδειξη του αριθμού των επαναλήψεων που θα εκτελεστούν σε ένα βρόχο. Η δομή της είναι της μορφής range(αρχή, μέχρι, βήμα), όπου αρχή, μέχρι, βήμα ακεραίοι αριθμοί. Οι ενδείξεις της αρχής και του βήματος δεν είναι υποχρεωτικές και, αν δεν αναφέρονται, θα αρχίσει από 0 και θα συνεχίσει με βήμα 1. Αντίθετα η ένδειξη μέχρι πρέπει πάντα να αναφέρεται.

- range(10), παράγει τη λίστα: [0,1,2,3,4,5,6,7,8,9].
- range(1, 8), παράγει τη λίστα: [1,2,3,4,5,6,7]
- range(8, -1, -1), παράγει τη λίστα [8, 7, 6, 5, 4, 3, 2, 1, 0]

Δομή Επανάληψης με while βρόχο (ή Όσο <συνθήκη> επανάλαβε)

Η δομή while χρησιμοποιείται για μη προκαθορισμένο αριθμό επαναλήψεων. Σε κάθε επανάληψη (και στην αρχική) πραγματοποιείται ο έλεγχος της συνθήκης, πριν από την εκτέλεση των εντολών του βρόχου, πράγμα που σημαίνει ότι υπάρχει περίπτωση να μην εκτελεστούν οι εντολές του βρόχου.

Αρχική τιμή μεταβλητής

while ονομα_μεταβλητής <συνθήκη>:

Εντολή_1

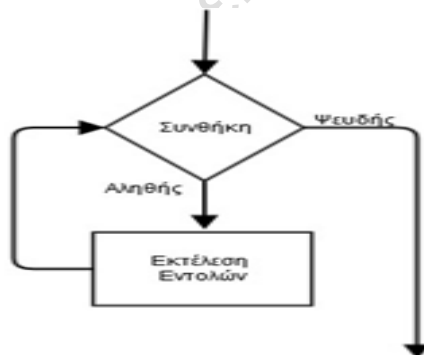
Εντολή_2

....

Εντολή_κ

Σημείωση1: Θα πρέπει μέσα στο μπλοκ εντολών να υπάρχει κατάλληλη εντολή, ώστε να εξασφαλίζεται ότι κάποια στιγμή η συνθήκη θα γίνει ψευδής και θα διακοπεί ο βρόχος. Διαφορετικά ο βρόχος δε θα τερματίζει.

Σημείωση 2: πριν το βρόχο while θα πρέπει αρχικά να δώσουμε μία τιμή στη μεταβλητή που ελέγχει τη συνθήκη του βρόχου, ώστε ανάλογα να εκτελεστεί ή όχι ο βρόχος.



Παράδειγμα

Πρόγραμμα επανάληψη while

x = 40

αρχική τιμή στη μεταβλητή x

while x < 50 :

έλεγχος της επανάληψης

 x = x + 1

αύξηση του μετρητή

print x

print x

Μια συνήθης εφαρμογή του while βρόχου είναι να ελέγχει την εγκυρότητα των δεδομένων εισόδου από το χρήστη.

Δραστηριότητα

Συμπλήρωσε κατάλληλα την παρακάτω συνθήκη, έτσι ώστε να εξασφαλίζεται ότι ο χρήστης θα εισάγει τελικά την τιμή 0-20 για το βαθμό στο μάθημα Εφαρμογές Πληροφορικής.

```
#Έλεγχος εισαγωγής δεδομένων
choice = input(' Δώστε το βαθμό που πήρατε στο μάθημα Εφαρμογές
Πληροφορικής')
while .....:
    choice = input('Παρακαλώ δώστε έγκυρη τιμή ')
```

Δραστηριότητα: Δομή επανάληψης

Βρείτε τι κάνει το παρακάτω πρόγραμμα και δώστε με μία πρόταση ένα χαρακτηριστικό τίτλο.

Σημείωση: Η εντολή "import random" εισάγει μια βιβλιοθήκη συναρτήσεων για την παραγωγή τυχαίων αριθμών. Η συνάρτηση random επιστρέφει έναν τυχαίο δεκαδικό ανάμεσα στο 0.0 και στο 1.0 (συμπεριλαμβανομένου του 0.0, αλλά όχι του 1.0). Η συνάρτηση randint παίρνει ως παραμέτρους ένα κάτω και ένα άνω όριο και επιστρέφει έναν ακέραιο μεταξύ αυτών των ορίων, συμπεριλαμβανομένων και αυτών των δύο.

```
import random
thenum = random.randint(1,10)
print "Ψήφισα έναν από τους 10 υποψηφίους για πρόεδρο
15μελούς"
print "Μπορείς να μαντέψεις τον αύξοντα αριθμό αυτού που
ψήφισα"
guess = 0
while guess != thenum:
```

```
guess = input("Δώσε αριθμό: ")
if guess>thenum:
    print "Έδωσες μεγαλύτερο αριθμό"
elif guess<thenum:
    print "Έδωσες μικρότερο αριθμό"
else:
    print "Τον βρήκες! "
```

Δραστηριότητα: Εμφωλευμένη δομή επανάληψης

Σε έναν αθλητικό μαθητικό αγώνα στίβου, στο αγώνισμα του μήκους, συμμετέχουν στους προκριματικούς 20 μαθητές από όλα τα σχολεία της Περιφέρειας. Στον τελικό περνούν όσοι μαθητές σημειώσουν επίδοση μεγαλύτερη ή ίση από 4.5 μέτρα. Κάθε αθλητής έχει 3 προσπάθειες. Αν σημειώσει επίδοση ίση ή μεγαλύτερη από το όριο πρόκρισης, σταματάει τις προσπάθειες. Να γραφεί αλγόριθμος και στη συνέχεια αντίστοιχο πρόγραμμα σε Python, που να διαβάζει τις επιδόσεις των αλμάτων κάθε αθλητή και να υπολογίζει την καλύτερη επίδοσή του. Να ελέγχει δίνοντας ανάλογο μήνυμα στην οθόνη αν ο αθλητής προκρίθηκε ή όχι στον τελικό και τελικά να εμφανίζει στην οθόνη, πόσοι αθλητές προκρίθηκαν και ποια ήταν η καλύτερη επίδοση που σημειώθηκε.

Αλγόριθμος πρόκριση στο στίβο και εύρεση καλύτερης επίδοσης

```
max_alma ← 0
count_athletes ← 0
Για i=1 μέχρι 20
    max_epidosi ← 0
    prospatheia ← 1
    Όσο prospatheia ≤ 3 και max_epidosi < 4.5 επανάλαβε
        Εμφάνισε ("αγωνίζεται ο", i, "ος αθλητής στην", prospatheia, "η
```

```
προσπάθεια. Δώσε την επίδοση του αθλητή:")
    Διάβασε epidosi
    Αν max_epidosi < epidosi τότε
        max_epidosi ← epidosi
    Αν epidosi ≥ 4.5 τότε
        Εμφάνισε 'Ο', i, 'ος αθλητής προκρίθηκε με άλμα
στα', epidosi, 'μ.'
        count_athletes = count_athletes + 1
    Τέλος_αν
    Τέλος_αν
    prospatheia = prospatheia + 1
Τέλος_Επανάληψης
Αν epidosi < 4.5 τότε
Εμφάνισε ('Ο', i, 'ος αθλητής δεν προκρίθηκε. Το καλύτερο άλμα του
ήταν:', max_epidosi)
    Αν max_alma < max_epidosi τότε
        max_alma = max_epidosi
Τέλος_Επανάληψης
Εμφάνισε ('Τελικά προκρίθηκαν', count_athletes, 'αθλητές')
Εμφάνισε ('Η καλύτερη επίδοση που σημειώθηκε
ήταν', max_alma, 'm.')
Τέλος πρόκριση στο στίβο και εύρεση καλύτερης επίδοσης
```

```
#Πρόγραμμα πρόκριση στο στίβο και εύρεση καλύτερης επίδοσης
max_alma=0
count_athletes=0
for i in range(1,21):
    max_epidosi=0
    prospatheia=1
    while prospatheia<=3 and max_epidosi<4.5:
        print 'αγωνίζεται ο',i,'ος αθλητής στην',prospatheia,'η προσπάθεια'
        epidosi=input('Δώσε την επίδοση του αθλητή: ')
        if max_epidosi<epidosi:
            max_epidosi=epidosi
            if epidosi>=4.5:
                print 'Ο',i,'ος αθλητής προκρίθηκε με άλμα στα',epidosi,'μέτρα'
                count_athletes=count_athletes+1
            prospatheia=prospatheia+1
    if max_epidosi < 4.5:
        print 'ο',i,'ος αθλητής δεν προκρίθηκε. Το καλύτερο άλμα του ήταν:',max_epidosi
    if max_alma<max_epidosi:
        max_alma=max_epidosi
print 'Τελικά προκρίθηκαν',count_athletes,'αθλητές'
print 'Η καλύτερη επίδοση που σημειώθηκε ήταν',max_alma,'μέτρα'
```

4.2 Συναρτήσεις

Οι συναρτήσεις είναι επαναχρησιμοποιήσιμα μέρη προγραμμάτων. Μας επιτρέπουν να δίνουμε ένα όνομα σε ένα σύνολο εντολών και να το εκτελούμε καλώντας το όνομα αυτό, από οπουδήποτε στο πρόγραμμα και όσες φορές θέλουμε, διαδικασία που ονομάζεται *κλήση* (calling) της συνάρτησης.

4.2.1 Δημιουργώντας δικές μας συναρτήσεις

Για να ορίσουμε μια δική μας συνάρτηση χρησιμοποιούμε τη χαρακτηριστική λέξη **def**, ακολουθεί ένα όνομα που ταυτοποιεί την εκάστοτε συνάρτηση και ένα ζευγάρι παρενθέσεων που μπορούν να περικλείουν ονόματα μεταβλητών, ενώ η γραμμή τελειώνει με άνω και κάτω τελεία (:).

Ας δούμε ένα απλό παράδειγμα:

```
def hellow ():  
    print('Γεια σου κόσμε!')  
# Τέλος της συνάρτησης  
hellow() # κλήση της συνάρτησης  
hellow() # κι άλλη μία κλήση της συνάρτησης
```

Αν θέλουμε μπορούμε να φτιάξουμε άλλη μία συνάρτηση, την `epanalave_hellow()`, που να καλεί την `hellow` δύο φορές. Σκεφτείτε τι θα εμφανιστεί στην οθόνη, όταν εκτελεστούν τα παρακάτω. Δημιουργήστε μια ακόμα συνάρτηση που να καλεί και να χρησιμοποιεί κατάλληλα την `epanalave_hellow()`, ώστε να εμφανίζει τα λόγια τέσσερις φορές.

```
def epanelave_hellow():  
    hellow()  
    hellow()  
#τέλος συνάρτησης  
epanalave_hellow()  
def epanelave_4fores():  
    epanelave_hellow()  
    epanelave_hellow()  
#τέλος συνάρτησης  
epanalave_4fores()
```

4.2.2 Παράμετροι συναρτήσεων

Όπως αναφέρθηκε, μια συνάρτηση μπορεί να δεχθεί παραμέτρους, οι οποίες χρησιμοποιούνται για να παρέχουμε διάφορες τιμές σε αυτήν, έτσι ώστε να παράγει κάποιο αποτέλεσμα ή να εκτελεί κάποιες ενέργειες χρησιμοποιώντας τις τιμές αυτές. Οι παράμετροι μοιάζουν με τις μεταβλητές, με τη διαφορά ότι οι τιμές αυτών των μεταβλητών ορίζονται, όταν καλούμε τη συνάρτηση.

Οι παράμετροι καθορίζονται μέσα στο ζευγάρι των παρενθέσεων στον ορισμό της συνάρτησης και διαχωρίζονται με κόμμα. Όταν καλούμε τη συνάρτηση, δίνουμε και τις τιμές με τον ίδιο τρόπο.

Σημείωση για την ορολογία που χρησιμοποιείται: οι ονομασίες που δίνουμε στον ορισμό της συνάρτησης ονομάζονται *παράμετροι*, ενώ οι τιμές που δίνουμε, όταν καλούμε τη συνάρτηση, ονομάζονται *ορίσματα*.

Κάποιες από τις ενσωματωμένες συναρτήσεις που έχουμε ήδη συναντήσει, δεν απαιτούν ορίσματα, όπως, για παράδειγμα, όταν καλούμε τη `math.pi`, που χρησιμοποιείται για τη μαθηματική σταθερά π 3.14..., όπου δεν έχουμε κάποιο όρισμα. Σε άλλες όμως, συναρτήσεις απαιτούνται ένα ή και περισσότερα ορίσματα, όπως στη `math.pow`, η οποία υπολογίζει την ύψωση σε δύναμη ακεραίων, που απαιτούνται δύο, ένα για τη βάση και ένα για τον εκθέτη.

```
def ginomeno (a,b):  
    x = a * b  
    return x  
print (ginomeno(5,10))
```

Περισσότερα για τις συναρτήσεις και την εμβέλεια των παραμέτρων θα έχουμε την ευκαιρία να μελετήσουμε στο κεφάλαιο

Μέρος II

ΕΣΩΤΕΡΙΚΗ ΕΓΚΕΛΕΞΗ ΕΚΔΟΣΗ v201

5

Κλασικοί Αλγόριθμοι II

ΕΣΩΤΕΡΙΚΗ ΕΓΚΕΚΡΙΤΗ ΕΚΔΟΣΗ v201

5. Κλασικοί Αλγόριθμοι II

Εισαγωγή

Στο κεφάλαιο αυτό αναπτύσσονται ορισμένοι από τους βασικούς αλγορίθμους της Πληροφορικής Επιστήμης. Η έννοια του αλγορίθμου αποτελεί τον ακρογωνιαίο λίθο της επιστήμης της Πληροφορικής, μια και το βασικό πρόβλημα το οποίο εξετάζει είναι ο σχεδιασμός αλγορίθμων για την επίλυση προβλημάτων. Αυτό όμως δε σημαίνει ότι, αν επινοήσουμε έναν οποιονδήποτε αλγόριθμο που να λύνει το πρόβλημα που θέλουμε, αυτό είναι αρκετό. Το βασικό, αλλά και αιώνιο ερώτημα της Πληροφορικής, είναι “Μπορούμε καλύτερα;” Μπορούμε να σχεδιάσουμε έναν αλγόριθμο ο οποίος να κάνει καλύτερη διαχείριση των υπολογιστικών πόρων του συστήματος (μνήμης και επεξεργαστή); Σήμερα το βασικό πρόβλημα δεν είναι τόσο η χρήση της κύριας μνήμης, αλλά ο χρόνος εκτέλεσης του αλγορίθμου. Ο στόχος μας λοιπόν, δεν είναι να λύσουμε απλώς ένα πρόβλημα, αλλά να το λύσουμε με το βέλτιστο τρόπο. Πολλές φορές δεν υπάρχει ένας αλγόριθμος που να είναι βέλτιστος για όλες τις περιπτώσεις δεδομένων, αλλά εξαρτάται από την οργάνωση των δεδομένων. Για παράδειγμα η *σειριακή αναζήτηση* σαρώνει όλα τα στοιχεία μιας ταξινομημένης λίστας χωρίς να εκμεταλλεύεται τη διάταξη των στοιχείων. Ο σχεδιασμός ενός νέου αλγορίθμου που θα εκμεταλλεύεται την ιδιαίτερη δομή των δεδομένων, για να βρίσκει το ζητούμενο στοιχείο πιο γρήγορα, αποτελεί ένα από τα αντικείμενα αυτής της ενότητας.

Επειδή υπάρχουν διάφορες κατηγορίες δεδομένων, οι επιστήμονες της Πληροφορικής σχεδιάζουν διάφορους αλγορίθμους ανάλογα με την ιδιαίτερη δομή των δεδομένων. Γι' αυτόν το λόγο δεν έχουμε μόνο έναν αλγόριθμο ταξινόμησης, αλλά αρκετούς, κάποιους από τους οποίους θα μελετήσουμε σε αυτή την ενότητα.

Διδακτικοί στόχοι

Μετά τη μελέτη του κεφαλαίου, θα μπορούμε να:

- αναλύουμε και να εφαρμόζουμε κατάλληλα κλασσικούς αλγορίθμους για προβλήματα ταξινόμησης και αναζήτησης
- υλοποιούμε σε μια γλώσσα προγραμματισμού κλασσικούς αλγορίθμους ταξινόμησης και αναζήτησης
- συγκρίνουμε και να επιλέγουμε τον κατάλληλο αλγόριθμο ανάλογα με το είδος του προβλήματος.

Λέξεις κλειδιά

Αλγόριθμοι, ταξινόμηση, αναζήτηση.

Διδακτικές Ενότητες

5.1 Δυαδική αναζήτηση

Στην προηγούμενη τάξη λύσαμε το πρόβλημα της αναζήτησης, με τον αλγόριθμο της *Σειριακής αναζήτησης*. Ο αλγόριθμος αυτός στην χειρότερη περίπτωση, όπου το στοιχείο δεν υπάρχει στο σύνολο δεδομένων ή είναι το τελευταίο στη σειρά, θα ελέγξει όλα τα στοιχεία του συνόλου στο οποίο ψάχνουμε. Υπάρχει τρόπος να βρούμε το ζητούμενο στοιχείο πιο γρήγορα; Μια ιδέα είναι να εκμεταλλευτούμε τη δομή του συνόλου των δεδομένων, αλλά θα πρέπει να ξέρουμε αν τα δεδομένα είναι τυχαία τοποθετημένα ή με βάση κάποια λογική. Για παράδειγμα: τα ονόματα σε μια λίστα μπορεί να είναι σε αλφαβητική σειρά ή οι απόφοιτοι μιας σχολής σε φθίνουσα σειρά με βάση τον βαθμό πτυχίου.

Ας παίξουμε το εξής παιχνίδι:

Σκεφτείτε έναν αριθμό από το 1 έως το 1000 και απαντήστε στην ερώτηση:

Ο αριθμός που σκεφτήκατε είναι μεγαλύτερος ή μικρότερος από το 500;

Ας υποθέσουμε ότι ο αριθμός που σκεφτήκαμε είναι μικρότερος του 500, άρα η απάντηση στην ερώτηση είναι μικρότερος. Αφού ο αριθμός ξέρουμε ότι είναι μικρότερος του 500, τότε θα βρίσκεται σίγουρα μεταξύ του 1 και του 500. Έτσι, ενώ στην αρχή είχαμε να

ψάξουμε μεταξύ 1000 αριθμών τώρα με μία ερώτηση καταφέραμε να περιορίσουμε το χώρο αναζήτησης του προβλήματος στο μισό. Με μια δεύτερη ερώτηση για τη σχέση του αριθμού μας με το 250 και ανεξάρτητα από την απάντηση, περιορίζουμε πάλι το πρόβλημα στο μισό του μισού, δηλαδή στο $\frac{1}{4}$ του αρχικού προβλήματος.

Μπορείτε να συνεχίσετε το παιχνίδι; Πόσα βήματα θα χρειαστούν για να βρούμε τον αριθμό;

Η παραπάνω μέθοδος είναι γνωστή ως **Διαδική αναζήτηση** και εκμεταλλεύεται τη διάταξη των στοιχείων του συνόλου, διαμερίζοντας κάθε φορά το σύνολο σε δυο ίσα μέρη, και εφαρμόζοντας πάλι την ίδια μέθοδο στο υποσύνολο στο οποίο ανήκει ο ζητούμενος αριθμός.

Ας δούμε πώς μπορούμε να αποτυπώσουμε την παραπάνω ιδέα σε πρόγραμμα, υλοποιώντας τον αλγόριθμο αυτόν σε Python.

Δραστηριότητα : Μάντεψε τον αριθμό

Βήμα 1: Παραγωγή τυχαίων αριθμών

Πολλές φορές θέλουμε να παράγουμε αριθμούς με τυχαίο τρόπο. Η βιβλιοθήκη *random* περιέχει μια ποικιλία συναρτήσεων γι' αυτόν τον σκοπό. Δύο από αυτές που θα χρησιμοποιούμε κατά κόρον, είναι η **randint** και η **randrange**, που επιστρέφουν τυχαίους ακέραιους αριθμούς εντός κάποιων ορίων. Τα όρια στην **randrange** ακολουθούν την ίδια λογική με την **range** της Python, ενώ η **randint** συμπεριλαμβάνει και το δεξιό άκρο.

```
import random
number = random.randint(1, 10)
# επιστρέφει έναν τυχαίο αριθμό στο [ 1, 10]
number = random.randrange(1,10)
# επιστρέφει έναν τυχαίο αριθμό στο [ 1, 9]
number = random.randrange(10)
# επιστρέφει έναν τυχαίο αριθμό στο [ 0, 9]
```

Βήμα 2: Ψάχνοντας στα τυφλά

Το παρακάτω πρόγραμμα επιλέγει έναν αριθμό στην τύχη στο διάστημα [1,100] και στη συνέχεια δίνει στο χρήστη άπειρες προσπάθειες μέχρι να μαντέψει τον αριθμό. Ο χρήστης μαντεύει στην τύχη, χωρίς κάποια συγκεκριμένη στρατηγική, με το πρόγραμμα να χρησιμοποιεί έναν μετρητή, τον `guesses`, για να μετράει τις προσπάθειες του χρήστη.

```
import random
secret_number = random.randint(1, 100)

guesses = 0
found = False

while not found :
    guess = input("Μάντεψε τον αριθμό : ")
    guesses = guesses + 1
    if guess == secret_number :
        print "Μπράβο το βρήκες με ", guesses, " προσπάθειες"
        found = True
    else :
        print "Δυστυχώς δεν το βρήκες, Ξαναπροσπάθησε"
```

Βήμα 3: Καταστρώνοντας τη στρατηγική

Θα τροποποιήσουμε το πρόγραμμα έτσι, ώστε να δίνει μια βοήθεια στον παίκτη. Θα ελέγχει αν ο αριθμός που μαντέψε είναι μικρότερος ή μεγαλύτερος από τον μυστικό αριθμό και θα εμφανίζει κατάλληλο μήνυμα. Για να γίνει αυτό, θα χρειαστεί να προσθέσουμε δυο ακόμα περιπτώσεις στη δομή επιλογής, όπως φαίνεται παρακάτω:

```
if guess == secret_number :  
    print "Μπράβο το βρήκες με ", guesses, " προσπάθειες"  
    found = True  
else :  
    if guess < secret_number:  
        print "Ο αριθμός σου είναι μικρότερος από τον ζητούμενο"  
    else :  
        print " Ο αριθμός σου είναι μεγαλύτερος από τον ζητούμενο "
```

Έτσι, αν ο αριθμός που ψάχνει ο παίκτης είναι μικρότερος από αυτόν που έδωσε, τότε ξέρει ότι πρέπει να ψάξει από εκεί και κάτω, με αποτέλεσμα το μέγεθος του προβλήματος να μειωθεί στο μισό. Επίσης, θα επιτρέπει στον παίκτη μέχρι 10 προσπάθειες και οι αριθμοί θα είναι στο διάστημα [1, 1000]. Παρατήρηση: αρκούν άραγε 10 προσπάθειες για 1000 αριθμούς; Αν ναι, τότε για 1.000.000 αριθμούς πόσες προσπάθειες πιστεύετε ότι χρειάζονται;

```
import random  
secret_number = random.randint(1, 1000)  
  
guesses = 0  
found = False  
  
while not found and guesses < 10 :  
    guess = input("Μάντεψε τον αριθμό : ")  
    guesses = guesses + 1  
    if guess == secret_number :  
        found = True  
    else :  
        if guess < secret_number :
```

```

    print "Ο αριθμός σου είναι μικρότερος"
else :
    print "Ο αριθμός σου είναι μεγαλύτερος"
if found == True :
    print "Μπράβο το βρήκες με ", guesses, " προσπάθειες"
else :
    print "Δυστυχώς δεν το βρήκες"

```

Βήμα 4: Σκέφτεται ο υπολογιστής;

Θα είχε ενδιαφέρον η εναλλαγή ρόλων μεταξύ υπολογιστή και παίκτη, όπου ο υπολογιστής θα ρωτάει σε κάθε βήμα και ο χρήστης θα απαντάει ανάλογα. Κάθε φορά που θα λαμβάνουμε μια απάντηση από το χρήστη πρέπει να περιορίζουμε το σύνολο των δεδομένων στο οποίο γίνεται η αναζήτηση. Γι' αυτό θα χρειαστούμε δύο μεταβλητές οι οποίες θα καθορίζουν το διάστημα στο οποίο βρίσκεται ο ζητούμενος αριθμός και μια τρίτη για το μέσο του διαστήματος. Ακολουθεί ένα παράδειγμα του παιχνιδιού όπου ο μυστικός αριθμός είναι το 11 και το διάστημα είναι το [1,16]. Αρχικά το πρόγραμμα ρωτάει το χρήστη αν ο αριθμός είναι μικρότερος, μεγαλύτερος ή ίσος με το μέσον του διαστήματος που είναι το 8.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----



$$15 > 8$$

Ο χρήστης απαντά ότι ο αριθμός είναι μεγαλύτερος από το 8, άρα το διάστημα στο οποίο θα πρέπει να ψάξουμε τώρα είναι από το μέσον και πέρα, δηλαδή από το 9 και πάνω.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----



$$11 < 12$$

Συνεχίζουμε την ίδια διαδικασία υπολογίζοντας πάντα το μέσο του διαστήματος που έχει απομείνει και αναπροσαρμόζοντας κατάλληλα τα άκρα του. Τώρα ο αριθμός που ψάχνουμε είναι μικρότερος από το 12, άρα βρίσκεται αριστερά του και χρειάζεται να μεταφέρουμε αριστερά το δεξί άκρο.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----



$$11 > 10$$

Όμοια τώρα πρέπει το αριστερό άκρο να έρθει δεξιά.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----



Όπως διαπιστώσατε, η εύρεση του μυστικού αριθμού δε χρειάστηκε 16 συγκρίσεις αλλά μόλις 4. Ο αλγόριθμος που μόλις περιγράψαμε είναι γνωστός ως **αλγόριθμος της δυαδικής αναζήτησης**, γιατί σε κάθε βήμα χωρίζει το χώρο αναζήτησης στο μισό. Στη συγκεκριμένη περίπτωση, που ο χώρος αναζήτησης ήταν το διάστημα $[1,16]$, χρειάστηκαν 4 συγκρίσεις, αφού χρειάζονται τέσσερις διαδοχικές διαιρέσεις με το 2, για να καταλήξουμε από 16 στοιχεία σε 1. Μπορείτε να υπολογίσετε πόσες συγκρίσεις θα θέλαμε, αν είχαμε $1024 = 2^{10}$ στοιχεία;

Τώρα ο άνθρωπος σκέφτεται έναν αριθμό από 1 έως 1000

`N = 1000`

`print "Σκέψου έναν αριθμό από το 1 έως το ", N`

`guesses = 0`

`found = False`

`first = 1`

`last = N`

```
while not found and guesses < 10 :  
    mid = ( first + last ) // 2  
    answer = input("Είναι ο αριθμός ο  " + mid + " ? (N/O)" )  
    guesses = guesses + 1  
    if answer == "N" :  
        found = True  
    else :  
        answer = input("Είναι μικρότερος του " + mid + " ? (N/O)" )  
        if answer == "N" :  
            first = mid + 1  
        else :  
            last = mid - 1  
    if found == True :  
        print "Κέρδισα!!! Το βρήκα με ", guesses, " προσπάθειες"  
    else :  
        print "Κέρδισες"
```

Στο παραπάνω πρόγραμμα στις μεταβλητές `first` και `last` καταχωρούμε τις θέσεις των δυο άκρων του διαστήματος και στην `mid` το μέσο του διαστήματος.

Δυαδική αναζήτηση σε λίστα

Προηγουμένως χρησιμοποιήσαμε τον αλγόριθμο της δυαδικής αναζήτησης για να βρούμε έναν μυστικό αριθμό μέσα σε κάποια όρια. Ο αλγόριθμος βασίζεται στο γεγονός ότι οι αριθμοί είναι διατεταγμένοι κατά αύξουσα σειρά. Για να εφαρμοστεί ο ίδιος αλγόριθμος και σε άλλα δεδομένα, όπως για παράδειγμα σε ονόματα, θα πρέπει αυτά να είναι διατεταγμένα επίσης σε κάποιου είδους σειρά, όπως σε αλφαβητική. Για παράδειγμα με χρήση της δυαδικής αναζήτησης, μπορούμε να βρούμε ένα όνομα στον τηλεφωνικό κατάλογο, όπου όλα τα ονόματα είναι σε αλφαβητική σειρά. Έτσι μπορούμε να εφαρμόσουμε τη δυαδική αναζήτηση

στα στοιχεία μιας λίστας τα οποία βρίσκονται σε κάποια λογική διάταξη, είτε αυτά είναι αριθμοί είτε αλφαριθμητικά.

Παρακάτω φαίνεται η εκτέλεση του αλγόριθμου δυαδικής αναζήτησης σε μια λίστα 16 αριθμών, για την αναζήτηση του αριθμού 60. Η λογική εδώ είναι ακριβώς ίδια με το παιχνίδι “Βρες τον αριθμό”.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	17	23	28	30	35	40	45	50	60	63	68	70	88	96



$$60 > 40$$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	17	23	28	30	35	40	45	50	60	63	68	70	88	96



$$60 < 63$$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	17	23	28	30	35	40	45	50	60	63	68	70	88	96



$$60 > 50$$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	10	17	23	28	30	35	40	45	50	60	63	68	70	88	96



Παρακάτω δίνεται ο αλγόριθμος της δυαδικής αναζήτησης σε Python με τη μορφή μιας συνάρτησης, η οποία δέχεται ως παράμετρο μια λίστα array και το ζητούμενο στοιχείο key:

Αλγόριθμος Δυαδικής αναζήτησης

```
def binarySearch( array, key ) :  
    first = 0  
    last = len(array) - 1  
    found = False  
    while first <= last and not found :  
        mid = ( first + last ) // 2  
        if array[ mid ] == key :  
            found = True  
        elif array[ mid ] < key :  
            first = mid + 1  
        else :  
            last = mid - 1  
    return found
```

Παρατηρήστε ότι η παραπάνω συνάρτηση ισχύει για όλους τους τύπους δεδομένων για τους οποίους ισχύουν οι συγκριτικοί τελεστές `==`, `<`. Δηλαδή, μπορεί να κληθεί για ακέραιους ή πραγματικούς αριθμούς, αλφαριθμητικά ή ακόμα και σύνθετους τύπους για τους οποίους έχουμε ορίσει τους συγκεκριμένους τελεστές. Αυτό το χαρακτηριστικό είναι ένα από τα βασικά πλεονεκτήματα της Python.

Συνοψίζοντας, μπορούμε να πούμε ότι η δυαδική αναζήτηση:

- εκμεταλλεύεται τη διάταξη των στοιχείων ενός συνόλου δεδομένων για τη γρήγορη εύρεση ενός στοιχείου
- χρησιμοποιείται μόνο σε ταξινομημένες συλλογές δεδομένων
- βρίσκει το ζητούμενο πολύ πιο γρήγορα από ότι η σειριακή αναζήτηση.

5.2 Ταξινόμηση Ευθείας ανταλλαγής

Στην προηγούμενη τάξη παρουσιάστηκε ο αλγόριθμος **ταξινόμησης με επιλογή** (selection sort). Ένα από τα χαρακτηριστικά αυτού του αλγορίθμου είναι ότι εκτελεί πάντα τον ίδιο αριθμό συγκρίσεων για συλλογές δεδομένων με το ίδιο μέγεθος, ακόμα και για αυτές που είναι ήδη ταξινομημένες. Επειδή η ταξινόμηση είναι μια λειτουργία που χρησιμοποιείται πολύ συχνά στην Πληροφορική, έχουν αναπτυχθεί διάφοροι αλγόριθμοι ταξινόμησης, κάμποιοι από τους οποίους έχουν πολύ καλή απόδοση για ορισμένα σύνολα δεδομένων, όπως για παράδειγμα αυτά που είναι σχεδόν ταξινομημένα. Ένας αλγόριθμος ο οποίος σταματάει, όταν διαπιστώσει ότι τα δεδομένα είναι ήδη ταξινομημένα, είναι ο αλγόριθμος της **ευθείας ανταλλαγής**.

Εισαγωγική Δραστηριότητα

Σε μια πρόβα για την παρέλαση της 28ης Οκτωβρίου, τέσσερις μαθητές του τμήματος Β4 έχουν παραταχθεί σε αλφαβητική σειρά. Ενώ περιμένουν να ξεκινήσει η πρόβα, ο Αλέξανδρος φτάνει καθυστερημένος και ξεκινώντας από το τέλος της σειράς προσπαθεί να βρει τη θέση του, συγκρίνοντας κάθε φορά το όνομά του με αυτό του συμμαθητή ή της συμμαθήτριάς του που βρίσκεται μπροστά του στη σειρά.

Θέλουμε να σχεδιάσουμε έναν αλγόριθμο ο οποίος θα προωθεί τον Αλέξανδρο με διαδοχικές εναλλαγές προς τα εμπρός, ώστε να, βρεθεί μπροστά από τον Δημήτρη, στην αρχή της λίστας (θέση 0).

0	Δημήτρης	Δημήτρης	Δημήτρης	Δημήτρης	Αλέξανδρος
1	Ευγενία	Ευγενία	Ευγενία	Αλέξανδρος	Δημήτρης
2	Ναταλία	Ναταλία	Αλέξανδρος	Ευγενία	Ευγενία
3	Ρένια	Αλέξανδρος	Ναταλία	Ναταλία	Ναταλία
4	Αλέξανδρος	Ρένια	Ρένια	Ρένια	Ρένια

Εικόνα 5-1

Αρχικά, ας θεωρήσουμε ότι καταχωρούμε τους παραπάνω μαθητές σε μια λίστα `students` με την παρακάτω εντολή:

```
students = ["Δημήτρης", "Ευγενία", "Ναταλία", "Ρένια", "Αλέξανδρος"]
```

Αρχικά, ο Αλέξανδρος θα συγκριθεί αλφαβητικά με τη Ρένια και αφού διαπιστωθεί ότι προηγείται αλφαβητικά, θα αλλάξει θέση με τη Ρένια. Η σύγκριση δυο αλφαριθμητικών γίνεται με βάση την λεξικογραφική διάταξη ή όπως λέμε την αλφαβητική σειρά μεταξύ τους. Για παράδειγμα το αλφαριθμητικό "ΔΗΜΗΤΡΗΣ" έπεται του "ΔΗΜΗΤΡΑ" αφού "Α" < "Η".

Αφού διαπιστωθεί ότι τα στοιχεία της λίστας πρέπει να αλλάξουν αμοιβαία θέσεις, μπορεί να χρησιμοποιηθεί μια προσωρινή μεταβλητή `temp`.

```
if students[4] < students[3] :  
    temp      = students[4]  
    students[4] = students[3]  
    students[3] = temp
```

ή απευθείας η εντολή αντιμετάθεσης μεταβλητών της Python, με χρήση πλειάδων:

```
if students[4] < students[3] :  
    students[4] , students[3] = students[3] , students[4]
```

Στη συνέχεια, ο Αλέξανδρος θα συγκριθεί με τη Ναταλία και αφού ισχύει "Αλέξανδρος" < "Ναταλία", θα αλλάξει θέσεις και με τη Ναταλία. Ομοίως θα συγκριθεί και θα αλλάξει διαδοχικά θέσεις με την Ευγενία και τον Δημήτρη, ανεβαίνοντας στην θέση 0.

```
if students[4] < students[3] :  
    students[4] , students[3] = students[3] , students[4]  
if students[3] < students[2] :  
    students[3] , students[2] = students[2] , students[3]  
if students[2] < students[1] :  
    students[2] , students[1] = students[1] , students[2]  
if students[1] < students[0] :  
    students[1] , students[0] = students[0] , students[1]
```

Οι παραπάνω εντολές μπορούν να γραφούν με πιο συνοπτικό τρόπο, με τη χρήση εντολής επανάληψης:

```
for j in range(4, 0, -1):
    if students[j] < students[j-1] :
        students[j] , students[j-1] = students[j-1] , students[j]
```

Υπενθύμιση: `range(4, 0, -1) = [ 4, 3, 2, 1 ]`

Αν εφαρμόσουμε τον ίδιο αλγόριθμο σε μια λίστα με N αριθμούς, τότε παίρνει τη μορφή:

```
N = len( array )
for j in range(N-1, 0, -1):
    if array[ j ] < array[ j-1 ] :
        array[ j ] , array[ j-1 ] = array[ j-1 ] , array[ j ]
```

το αποτέλεσμα θα είναι η προώθηση του μικρότερου αριθμού στην πρώτη θέση (0) της λίστας, όπως φαίνεται παρακάτω για τη λίστα `array = [ 21, 13, 8, 5, 3, 2 ]`, αν ξεκινήσουμε να συγκρίνουμε από το τέλος της λίστας.

	j=5	j=4	j=3	j=2	j=1	
0	21	21	21	21	21	2
1	13	13	13	13	2	21
2	8	8	8	2	13	13
3	5	5	2	8	8	8
4	3	2	5	5	5	5
5	2	3	3	3	3	3

Εικόνα 5-2

Τι θα συμβεί, αν εκτελέσουμε άλλη μια φορά τις ίδιες εντολές στη λίστα που προέκυψε; Αυτή τη φορά θα ανέβει στη δεύτερη θέση ο αμέσως μικρότερος αριθμός, δηλαδή το 3. Κάθε φορά που θα

εκτελείται το συγκεκριμένο τμήμα κώδικα, το αμέσως μικρότερο στοιχείο της λίστας θα ανεβαίνει στη σωστή θέση. Άρα, για να ταξινομηθεί πλήρως η λίστα, πρέπει να εκτελέσουμε το παραπάνω τμήμα κώδικα τόσες φορές όσες είναι τα στοιχεία της, δηλαδή N .

Έτσι καταλήγουμε στον παρακάτω αλγόριθμο

```
N = len( array )
for i in range(1 , N, 1):
    for j in range(N-1 , 1, -1):
        if array[j] < array[j-1] :
            array [ j ] , array [ j-1 ] = array[ j-1 ] , array[j]
```

Το περιεχόμενο της λίστας για κάθε επανάληψη του αλγορίθμου φαίνεται παρακάτω:

	i=1	i=2	i=3	i=4	i=5
0	2	2	2	2	2
1	21	3	3	3	3
2	13	21	5	5	5
3	8	13	21	8	8
4	5	8	13	21	13
5	3	5	8	13	21

Εικόνα 5-3

Με πράσινο είναι τα στοιχεία που ήδη έχουν ανέβει στις τελικές θέσεις τους. Κατά την εκτέλεση του αλγορίθμου, η λίστα χωρίζεται σε δύο τμήματα: το ταξινομημένο (πράσινο) και αυτό που δεν έχει ταξινομηθεί ακόμα. Είναι φανερό λοιπόν ότι, κάθε φορά, το αμέσως μικρότερο στοιχείο θα ανέβει μέχρι το σημείο που αρχίζει το πράσινο χρώμα και όχι παραπάνω. Άρα δεν υπάρχει λόγος στην εσωτερική επανάληψη να συγκρίνουμε κάθε φορά μέχρι και το πρώτο στοιχείο, αλλά μέχρι το στοιχείο στη θέση $i-1$. Επίσης φαίνεται ότι δεν χρειάζονται N ($=6$) επαναλήψεις, αλλά $N-1$ ($=5$), όταν τα $N-1$ στοιχεία τοποθετηθούν στις θέσεις τους, με το

τελευταίο στοιχείο (21) να βρίσκεται στη μόνη θέση που περισσεύει, δηλαδή την τελευταία.

Αλγόριθμος ταξινόμησης ευθείας ανταλλαγής (Bubble Sort)

```
N = len( array )  
for i in range(1 , N, 1):  
    for j in range(N-1 , i-1 , -1):  
        if array[ j ] < array[ j-1 ] :  
            array [ j ] , array [ j-1 ] = array[ j-1 ] , array[ j ]
```

Ο αλγόριθμος αυτός ονομάζεται **αλγόριθμος ταξινόμησης ευθείας ανταλλαγής** (straight exchange sort), και είναι ευρύτερα γνωστός ως αλγόριθμος **ταξινόμησης φυσαλίδας** (bubble sort). Αυτό διότι σε κάθε πέρασμα, το αμέσως μικρότερο στοιχείο ανεβαίνει, όπως μια φυσαλίδα, στην επιφάνεια του νερού. Ο αλγόριθμος βασίζεται στη σύγκριση και αντιμετάθεση ζευγών που δεν ακολουθούν τη διάταξη της ταξινόμησης.

Ο αλγόριθμος ευθείας ανταλλαγής, αν και θεωρείται από τους πιο αργούς αλγορίθμους ταξινόμησης, έχει ένα πολύ σημαντικό πλεονέκτημα: ότι μπορεί να τροποποιηθεί, ώστε να τερματίσει μόλις διαπιστώσει ότι η λίστα έχει ταξινομηθεί. Έτσι αποφεύγονται πολλές περιπτώσεις συγκρίσεις στην περίπτωση λιστών που είναι κατά ένα σημαντικό μέρος ήδη ταξινομημένες. Για παράδειγμα, αν δοθούν οι αριθμοί : 3, 5, 8, 13, 21, 34, 55, 2, αρκεί μόνο ένα πέρασμα, ώστε να έρθει το 2 στην πρώτη θέση, αφού τα υπόλοιπα στοιχεία είναι ήδη στη σωστή σειρά. Σε αυτή την περίπτωση προσθέτουμε μια λογική μεταβλητή, η οποία θα παραμένει ψευδής (false), όσο η λίστα δεν είναι ταξινομημένη, ενώ αν γίνει αληθής ο αλγόριθμος τερματίζει.

Στην εικόνα 5.4 φαίνεται το περιεχόμενο μιας λίστας επτά (7) αριθμών αναλυτικά για κάθε βήμα του αλγορίθμου ταξινόμησης ευθείας ανταλλαγής. Κάθε στήλη αντικατοπτρίζει τη λίστα των αριθμών για κάθε τιμή των μετρητών i , j των επαναλήψεων. Με πράσινο χρώμα είναι τα στοιχεία που είναι ήδη ταξινομημένα και με πορτοκαλί αυτά που συγκρίνονται στο συγκεκριμένο βήμα.

1 ^ο βήμα για i = 1							
	j=6	j=5	j=4	j=3	j=2	j=1	
0	60	60	60	60	60	60	20
1	38	38	38	38	38	20	60
2	98	98	98	98	20	38	38
3	54	54	54	20	98	98	98
4	32	32	20	54	54	54	54
5	90	20	32	32	32	32	32
6	20	90	90	90	90	90	90

2 ^ο βήμα για i = 2						
	j=6	j=5	j=4	j=3	j=2	
0	20	20	20	20	20	20
1	60	60	60	60	60	32
2	38	38	38	38	32	60
3	98	98	98	32	38	38
4	54	54	32	98	98	98
5	32	32	54	54	54	54
6	90	90	90	90	90	90

3 ^ο βήμα για i = 3				
	j=6	j=5	j=4	j=3
0	20	20	20	20
1	32	32	32	32
2	60	60	60	38
3	38	38	38	60
4	98	98	54	54
5	54	54	98	98
6	90	90	90	90

4 ^ο βήμα για i = 4			
	j=6	j=5	j=4
0	20	20	20
1	32	32	60
2	38	38	38
3	60	60	54
4	54	54	60
5	98	90	90
6	90	98	98

5 ^ο βήμα για i = 5		
	j=6	j=5
0	20	20
1	32	32
2	38	38
3	54	54
4	60	60
5	90	90
6	98	98

6 ^ο βήμα για i = 6	
	j=6
0	20
1	32
2	38
3	54
4	60
5	90
6	98

Εικόνα 5-4

Παρατηρούμε ότι:

- Στο 6ο βήμα, τα 6 πρώτα στοιχεία είναι στις τελικές θέσεις τους οπότε το 7ο (98) που μένει, βρίσκεται υποχρεωτικά στη θέση που περισσεύει, που είναι η τελευταία. Για αυτό το λόγο ενώ έχουμε 7 στοιχεία, η εξωτερική επανάληψη είναι από 1 μέχρι N-1 και όχι από 0 μέχρι N-1. Έτσι δεν χρειάζονται επτά (7) αλλά έξι περάσματα.
- Μετά το 4ο βήμα, οι αριθμοί είναι ταξινομημένοι, άρα δεν χρειάζονται άλλες συγκρίσεις. Τα δυο τελευταία βήματα για αυτή την περίπτωση είναι περιττά. Πώς θα μπορούσαμε να βελτιώσουμε τον αλγόριθμο ευθείας ανταλλαγής, ώστε να σταματάει μόλις διαπιστώσει ότι δε χρειάζονται άλλες συγκρίσεις; Δείτε τη δραστηριότητα “βελτιωμένη φυσαλίδα” στο τέλος του κεφαλαίου.

5.3 Ταξινόμηση με Εισαγωγή

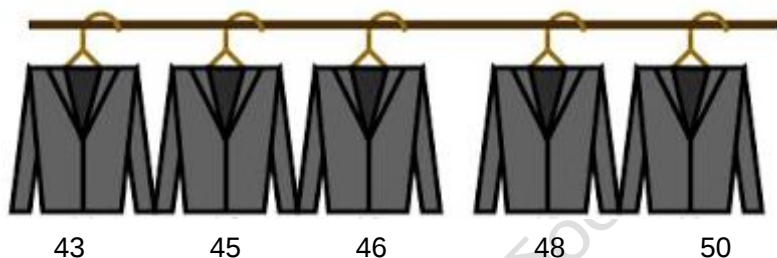
Σε μια βιοτεχνία ρούχων τα κοστούμια είναι τοποθετημένα σε αύξουσα σειρά με βάση το μέγεθός τους. Κάθε φορά που έρχεται ένα νέο κοστούμι ο ράφτης πρέπει να το τοποθετήσει στη σωστή θέση, χωρίς να χαλάσει τη σειρά των κοστούμιών και με προσοχή,

για να μην τα τσαλακώσει. Για να τοποθετήσει ο ράφτης το κοστούμι με μέγεθος 47 στο παρακάτω σχήμα, θα πρέπει να κάνει τρεις κινήσεις:

Βήμα 1: Να βρει σε ποιο σημείο πρέπει να μπει το νέο κοστούμι.

Βήμα 2: Να κάνει χώρο γι' αυτό μετακινώντας τα κοστούμια με μεγαλύτερο μέγεθος μια θέση δεξιά.

Βήμα 3: Να τοποθετήσει το κοστούμι στη θέση του.



Εικόνα 5-5

Πολλές φορές, χρειάζεται να έχουμε μια λίστα με στοιχεία η οποία ανανεώνεται συνεχώς και στην οποία τα στοιχεία πρέπει να βρίσκονται σε κάποια διάταξη. Για κάθε νέο στοιχείο που έρχεται στη λίστα, θα πρέπει να γίνουν δυο πράγματα, ώστε να διατηρήσουμε την επιθυμητή διάταξη των στοιχείων της λίστας:

Βήμα 1: Βρίσκουμε τη θέση στην οποία πρέπει να εισαχθεί ο αριθμός.

Βήμα 2: Εισάγουμε τον αριθμό μεταφέροντας όλους τους μεγαλύτερους αριθμούς από αυτόν μια θέση δεξιά (shift).

Για παράδειγμα: ας υποθέσουμε ότι έχουμε την παρακάτω λίστα 16 αριθμών οι οποίοι είναι σε αύξουσα σειρά και θέλουμε να εισάγουμε έναν νέο αριθμό, το 38, έτσι ώστε οι αριθμοί να παραμείνουν ταξινομημένοι. Αυτό μπορεί να γίνει σε δυο βήματα:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5	10	17	23	28	30	35	40	45	50	60	63	68	70	88	96	

38

Πρώτα γίνεται αναζήτηση στη λίστα, ώστε να βρεθεί η θέση στην οποία θα εισαχθεί το 38. Η θέση αυτή είναι η 7. Για να εισαχθεί το 38 στη θέση αυτή, θα πρέπει πρώτα, όλα τα στοιχεία από το 40 και μετά να μετακινηθούν μια θέση δεξιά, για να κάνουν χώρο στο 38.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5	10	17	23	28	30	35		40	45	50	60	63	68	70	88	96

↑
38

Στην συνέχεια το 38 τοποθετείται στην θέση 7, έτσι ώστε η νέα λίστα να παραμείνει ταξινομημένη.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5	10	17	23	28	30	35	38	40	45	50	60	63	68	70	88	96

Μπορούμε να χρησιμοποιήσουμε την παραπάνω ιδέα για να ταξινομήσουμε μια λίστα αριθμών. Κάθε φορά θα εισάγουμε και ένα νέο αριθμό στην ήδη ταξινομημένη λίστα, όπως κάναμε με το 38.

Ο αλγόριθμος ταξινόμησης που προκύπτει λέγεται **αλγόριθμος ταξινόμησης με εισαγωγή** (insertion sort) και είναι γνωστός και ως **ταξινόμηση παρεμβολής**, αφού ο νέος αριθμός παρεμβάλλεται στο ταξινομημένο τμήμα της λίστας.

Στην ταξινόμηση με εισαγωγή μπορούμε να χωρίσουμε τα στοιχεία σε 3 ομάδες:

- Αυτά που έχουν ήδη ταξινομηθεί και είναι σε πλήθος ίδια με τον αριθμό των βημάτων του αλγορίθμου μέχρι εκείνη τη στιγμή (πορτοκαλί χρώμα).
- Το στοιχείο που θα εισαχθεί (πράσινο χρώμα).
- Τα στοιχεία που δεν έχουν ταξινομηθεί ακόμα.

		i=1	i=2	i=3	i=4	i=5
0	13	8	8	3	3	2
1	8	13	13	8	8	3
2	21	21	21	13	10	8
3	3	3	3	21	13	10
4	10	10	10	10	21	13
5	2	2	2	2	2	21

Εικόνα 5-6

Σε αντίθεση με τον αλγόριθμο της ευθείας ανταλλαγής, τα στοιχεία με πράσινο είναι ταξινομημένα μεταξύ τους, αλλά δε βρίσκονται εξαρχής στις τελικές θέσεις. Μπορεί να εμφανιστεί ένα μικρότερο στοιχείο και να τοποθετηθεί στην πρώτη θέση σπρώχνοντάς τα όλα μια θέση προς τα κάτω, όπως, για παράδειγμα, συμβαίνει με τον αριθμό 2 στο τελευταίο βήμα του αλγορίθμου.

Αλγόριθμος Ταξινόμησης με Εισαγωγή (Insertion Sort)

```
def insertionSort( array ) :
    for i in range(1, len( array ) ) :
        value = array[ i ]
        pos = i
        # αναζήτηση
        while pos > 0 and array[ pos-1 ] > value :
            pos = pos - 1
        # μετακίνηση των στοιχείων μια θέση δεξιά
        for j in range(pos, i) :
            array[ j+1 ] = array[ j ]
        # το στοιχείο τοποθετείται στη θέση pos.
        array[ pos ] = value
```

Ο αλγόριθμος μπορεί να γραφτεί πιο συνοπτικά, όπως φαίνεται παρακάτω:

Αλγόριθμος Ταξινόμησης με Εισαγωγή (Insertion Sort) II

```
def insertionSort( array ) :  
    for i in range(1, len( array ) ) :  
        value = array[ i ]  
        j = i  
        while j > 0 and array[ j - 1 ] > value :  
            array[ j ] = array[ j-1 ]  
            j = j - 1  
        array[ j ] = value
```

Η λειτουργία του αλγορίθμου έχει ως εξής:

Σαρώνουμε το ταξινομημένο τμήμα της λίστας, από δεξιά προς τα αριστερά, μέχρι να συναντήσουμε το πρώτο στοιχείο που είναι μικρότερο αυτού που θέλουμε να εισάγουμε ($value=array[i]$). Ενώ διατρέχουμε τη λίστα, παράλληλα μετακινούμε και τα στοιχεία μια θέση προς τα δεξιά ($array[j] = array[j-1]$). Δηλαδή η μετακίνηση και η αναζήτηση γίνονται παράλληλα σε αυτήν την περίπτωση.

Επίσης, μόλις η μεταβλητή pos γίνει 0 η έκφραση $pos > 0$ είναι ψευδής, άρα όλη η συνθήκη είναι ψευδής, αφού χρησιμοποιείται ο λογικός τελεστής *και*, ανεξάρτητα από την τιμή της δεύτερης λογικής έκφρασης. Σε αυτήν την περίπτωση, η έκφραση $A[j-1] > value$ δε θα αποτιμηθεί.

Παρακάτω φαίνεται το περιεχόμενο μιας λίστας 7 αριθμών, αναλυτικά για κάθε βήμα του αλγορίθμου ταξινόμησης με εισαγωγή. Κάθε στήλη αντικατοπτρίζει το περιεχόμενο της λίστας για μια μετακίνηση του στοιχείου προς εισαγωγή. Με πράσινο χρώμα, είναι τα στοιχεία που είναι ήδη ταξινομημένα και με πορτοκαλί, αυτά που συγκρίνονται στο συγκεκριμένο βήμα.

1° βήμα για $i = 1$		2° βήμα για $i = 2$		3° βήμα για $i = 3$		4° βήμα για $i = 4$	
j=1		j=2		j=3 j=2 j=1		j=4 j=3 j=2	
0	60 38	0	38 38	0	38 38 38 38	0	38 38 38 38
1	38 60	1	60 60	1	60 60 54 54	1	60 60 54 54
2	98 98	2	98 98	2	98 54 60 60	2	98 54 60 60
3	54 54	3	54 54	3	54 98 98 98	3	54 98 98 98
4	32 32	4	32 32	4	32 32 32 32	4	32 32 32 32
5	90 90	5	90 90	5	90 90 90 90	5	90 90 90 90
6	20 20	6	20 20	6	20 20 20 20	6	20 20 20 20

4° βήμα για $i = 4$		5° βήμα για $i = 5$		6° βήμα για $i = 6$	
j=4 j=3 j=2 j=1		j=5 j=4		j=6 j=5 j=4 j=3 j=2 j=1	
0	38 38 38 38 32	0	32 32 32	0	32 32 32 32 32 20
1	54 54 54 32 38	1	38 38 38	1	38 38 38 38 20 32
2	60 60 32 54 54	2	54 54 54	2	54 54 54 54 20 38 38
3	98 32 60 60 60	3	60 60 60	3	60 60 60 20 54 54 54
4	32 98 98 98 98	4	98 90 90	4	90 90 20 60 60 60 60
5	90 90 90 90 90	5	90 98 98	5	98 20 90 90 90 90 90
6	20 20 20 20 20	6	20 20 20	6	20 98 98 98 98 98 98

Εικόνα 5-7

Παρατηρούμε ότι:

- Το πλήθος των συγκρίσεων και αντιμεταθέσεων σε κάθε βήμα δεν είναι σταθερό ή αυξανόμενο, αλλά εξαρτάται από τις θέσεις των στοιχείων.
- Ο αλγόριθμος ταξινόμησης με εισαγωγή είναι κατάλληλος για λίστες που είναι σχεδόν ταξινομημένες, σε αντίθεση με τον αλγόριθμο της επιλογής ή με τον αλγόριθμο ευθείας ανταλλαγής.
- Ο αλγόριθμος ταξινόμησης με εισαγωγή μπορεί να εφαρμοστεί σε λίστες οι οποίες δεν έχουν σταθερό μέγεθος και υλοποιούν συνεχή ροή δεδομένων σε πρακτικές εφαρμογές, όπως είναι π.χ. τα δίκτυα αισθητήρων (sensor networks). Το είδος αυτό των αλγορίθμων που επεξεργάζονται συνεχείς ροές δεδομένων ονομάζονται **άμεσοι αλγόριθμοι** (online algorithms).

Δραστηριότητες - Ερωτήσεις - Ασκήσεις κεφαλαίου**5.4 Δραστηριότητες - Άλυτες****Δραστηριότητα 1**

Να αντιστοιχήσετε τους παρακάτω αλγόριθμους με τις κατάλληλες λειτουργίες:

Αλγόριθμος	Λειτουργία
Ταξινόμηση με επιλογή	A. Αντιμετάθεση ζευγών
Ταξινόμηση ευθείας ανταλλαγής	B. Τοποθέτηση στοιχείου σε ταξινομημένη λίστα
Ταξινόμηση με εισαγωγή	Γ. Εύρεση ελαχίστου

Δραστηριότητα 2

Να τροποποιήσετε τον αλγόριθμο της ταξινόμησης με επιλογή, ώστε να ταξινομεί μια λίστα ακεραίων σε φθίνουσα σειρά. Υπάρχει τρόπος να το πετύχετε, χωρίς να κάνετε καμία απολύτως αλλαγή στον κύριο αλγόριθμο που δίνεται σε αυτή την ενότητα; Σε τι οφείλεται αυτό;

Δραστηριότητα 3

Να γράψετε μια συνάρτηση σε Python, η οποία θα δέχεται μια λίστα, θα ελέγχει αν τα στοιχεία της είναι σε αύξουσα σειρά και θα επιστρέφει αντίστοιχα `true` ή `false`. Προτείνεται να χρησιμοποιήσετε μια λογική μεταβλητή.

Δραστηριότητα 4 (βελτιωμένη φυσαλίδα)

Να δώσετε τη βελτιωμένη έκδοση του αλγορίθμου ταξινόμησης ευθείας ανταλλαγής η οποία τερματίζει, όταν διαπιστώσει ότι η λίστα είναι ταξινομημένη, ώστε να αποφεύγονται περιττές συγκρίσεις.

Υπόδειξη: Χρησιμοποιήστε μια λογική μεταβλητή η οποία θα αλλάζει τιμή, αν υπάρχουν τουλάχιστον δύο στοιχεία τα οποία δεν βρίσκονται στην επιθυμητή σειρά, καθώς η “φυσαλίδα ανεβαίνει στην επιφάνεια”.

Δραστηριότητα 5

Να γράψετε μια συνάρτηση σε Python η οποία θα δέχεται μια λίστα με λογικές τιμές True/False και θα διαχωρίζει τις τιμές αυτές, τοποθετώντας τα True πριν από τα False.

Δραστηριότητα 6

Να γράψετε ένα πρόγραμμα σε Python το οποίο θα δέχεται μια λίστα με λογικές τιμές True/False και στη συνέχεια θα καλεί την συνάρτηση του προηγούμενου ερωτήματος, ώστε να τοποθετηθούν τα True πριν από τα False. Στη συνέχεια θα τοποθετεί τις τιμές αυτές εναλλάξ, δηλαδή True, False, True, False, κλπ, εκτελώντας τις λιγότερες δυνατές συγκρίσεις.

Δραστηριότητα 7

Ας υποθέσουμε ότι σας δίνεται μια λίστα στην Python η οποία περιέχει λογικές τιμές True/False εναλλάξ. Επίσης, το πλήθος των True είναι ίσο με το πλήθος των False. Να γράψετε αλγόριθμο, σε Python, ο οποίος δεδομένης της παραπάνω δομής της λίστας, θα τοποθετεί τα True πριν από τα False εκτελώντας, τις λιγότερες δυνατές μετακινήσεις. Δεν επιτρέπεται να κάνετε καμία σύγκριση ούτε να χρησιμοποιήσετε τη δομή if. Θεωρήστε ότι το πρώτο στοιχείο της λίστας έχει την τιμή True.

Δραστηριότητα 8

Το πρόβλημα της ολλανδικής σημαίας αναφέρεται στην αναδιάταξη μιας λίστας γραμμάτων, η οποία περιέχει μόνο τους χαρακτήρες R, W, B. (Red, White, Blue), έτσι ώστε όλα τα R να βρίσκονται πριν από τα W και όλα τα W να βρίσκονται πριν από B. Να τροποποιήσετε έναν από τους αλγορίθμους ταξινόμησης που παρουσιάστηκαν σε αυτήν την ενότητα, ώστε να επιλύει αυτό το πρόβλημα.

Δραστηριότητα 9

Να γράψετε μια συνάρτηση σε Python η οποία διαβάζει αριθμούς από το χρήστη τους οποίους τοποθετεί σε μια λίστα σε φθίνουσα σειρά, την οποία και επιστρέφει. Κάθε φορά που διαβάζει έναν νέο αριθμό τον τοποθετεί στη σωστή θέση στην ήδη ταξινομημένη λίστα, ώστε να διατηρείται η φθίνουσα διάταξη των στοιχείων της λίστας. Ποιον αλγόριθμο ταξινόμησης σας θυμίζει η παραπάνω

λειτουργία; Σε τι διαφέρει η συνάρτηση που θα αναπτύξετε από τον αλγόριθμο αυτόν;

Δραστηριότητα 10

Να γράψετε ένα πρόγραμμα το οποίο θα διαβάζει από το χρήστη δυο λίστες αριθμών A και B και θα ταξινομεί σε αύξουσα σειρά τους αριθμούς της λίστας A. Στη συνέχεια θα εμφανίζει πόσοι από τους αριθμούς της λίστας B εμφανίζονται στην λίστα A.

Υπόδειξη: Να θεωρήσετε ότι οι αριθμοί της λίστας B είναι όλοι διαφορετικοί μεταξύ τους. Επίσης, να εκμεταλλευτείτε το γεγονός ότι η λίστα A είναι ταξινομημένη σε αύξουσα σειρά.

Δραστηριότητα 11

Δίνονται οι παρακάτω λίστες αριθμών.

55	34	21	13	8	5	3	2	1	1
----	----	----	----	---	---	---	---	---	---

4	6	8	16	28	32	64	2	1	1
---	---	---	----	----	----	----	---	---	---

28	28	28	28	28	28	28	28	28	28
----	----	----	----	----	----	----	----	----	----

28	28	28	28	28	6	6	6	6	6
----	----	----	----	----	---	---	---	---	---

Να απαντήσετε στα παρακάτω ερωτήματα:

1. Ποιος αλγόριθμος είναι κατάλληλος για την ταξινόμηση κάθε λίστας σε αύξουσα σειρά και γιατί;
2. Ποιος θα κάνει τις λιγότερες συγκρίσεις και ποιος τις λιγότερες αντιμεταθέσεις στοιχείων;
3. Ποιον ή ποιους αλγόριθμους ταξινόμησης θα επιλέγατε αν θέλατε να ταξινομήσετε μόνο τα 4 μικρότερα στοιχεία κάθε λίστας, χωρίς να ταξινομήσετε όλη τη λίστα;

5.5 Ερωτήσεις - Ασκήσεις

1. Να περιγράψετε τη λειτουργία του αλγορίθμου της δυαδικής αναζήτησης, δίνοντας και ένα παράδειγμα εκτέλεσής του.
2. Να περιγράψετε τη λειτουργία του αλγορίθμου της ταξινόμησης ευθείας ανταλλαγής και να δώσετε ένα παράδειγμα εκτέλεσής του.
3. Να περιγράψετε τη λειτουργία του αλγορίθμου της ταξινόμησης με εισαγωγή και να δώσετε ένα παράδειγμα εκτέλεσής του.
4. Αν έχουμε συνεχή ροή δεδομένων, αλλά θέλουμε τα δεδομένα μας να είναι μονίμως ταξινομημένα, ποιον αλγόριθμο θα χρησιμοποιήσουμε και γιατί;
5. Ποια είναι η απαραίτητη προϋπόθεση για να χρησιμοποιήσουμε τη δυαδική αναζήτηση;

Σύνοψη

Μια μέθοδος για την αναζήτηση ενός στοιχείου σε μια λίστα είναι η σειριακή αναζήτηση. Στην περίπτωση που η λίστα είναι ταξινομημένη, χρησιμοποιείται η δυαδική αναζήτηση η οποία εκμεταλλεύεται τη διάταξη των στοιχείων, για να βρει το ζητούμενο στοιχείο πιο γρήγορα. Για να χρησιμοποιηθεί η δυαδική αναζήτηση σε μη ταξινομημένα δεδομένα πρέπει προφανώς πρώτα αυτά να ταξινομηθούν.

Σε αυτό το κεφάλαιο παρουσιάστηκαν δυο αλγόριθμοι ταξινόμησης. Ο αλγόριθμος της ευθείας ανταλλαγής που βασίζεται στη σύγκριση γειτονικών ζευγών της λίστας και ο αλγόριθμος ταξινόμησης, που με εισαγωγή ταξινομεί μια λίστα βαθμιαία, εισάγοντας κάθε φορά το επόμενο στοιχείο, έτσι ώστε η λίστα να παραμένει ταξινομημένη.

6

Διαχείριση αρχείων

6. Διαχείριση Αρχείων

Εισαγωγή

Στο κεφάλαιο αυτό θα αναπτυχθούν οι τρόποι δημιουργίας και χειρισμού αρχείων. Επίσης, θα παρουσιαστούν οι βασικές λειτουργίες, όπως το άνοιγμα, κλείσιμο ενός αρχείου, το διάβασμα και γράψιμο μιας εγγραφής, μέσω της γλώσσας προγραμματισμού Python.

Διδακτικοί στόχοι

Μετά τη μελέτη του κεφαλαίου θα μπορούμε να:

- δημιουργούμε αρχεία
- χειριζόμαστε απλά αρχεία, όπως για παράδειγμα αρχεία κειμένου, μέσω βασικών εντολών (open, close, read, write) που χρησιμοποιεί η γλώσσα προγραμματισμού Python
- εκτελούμε πρόσθετες λειτουργίες σε αρχεία, όπως η αναζήτηση.

Λέξεις κλειδιά

Αρχείο κειμένου, άνοιγμα αρχείου, ανάγνωση αρχείου, εγγραφή σε αρχείο, κλείσιμο αρχείου, διαδρομή.

Διδακτικές Ενότητες

6.1 Εισαγωγή - δημιουργία, άνοιγμα, κλείσιμο αρχείων

Η δυνατότητα να δημιουργούμε, να διαβάζουμε και να γράφουμε σε αρχεία αποτελεί βασική εργασία σε πολλά προγράμματα και παρέχεται από όλες τις γλώσσες προγραμματισμού. Έτσι, στο κεφάλαιο αυτό θα ασχοληθούμε με τη διαχείριση αρχείων στη γλώσσα προγραμματισμού Python.

Τα περισσότερα προγράμματα που έχουμε δει μέχρι τώρα μπορούν να χαρακτηριστούν ως *προσωρινά*. Αυτό, με την έννοια ότι τρέχουν για ένα μικρό χρονικό διάστημα και παράγουν κάποια έξοδο, με τα δεδομένα τους (ως είσοδος από το πληκτρολόγιο ή ως έξοδος στην οθόνη), να χάνονται. Αυτό συμβαίνει, διότι τα δεδομένα ήταν αποθηκευμένα προσωρινά στην κύρια μνήμη του υπολογιστή, οπότε διαρκούσαν μόνο κατά την εκτέλεση του προγράμματος, λογική που έχει ως αποτέλεσμα, να απαιτείται να

δίνουμε τα δεδομένα από την αρχή σε κάθε εκτέλεση. Σε πολλές όμως περιπτώσεις θέλουμε τα δεδομένα να μη χάνονται. Θέλουμε να διαβάζουμε δεδομένα από ένα αρχείο του υπολογιστή στο οποίο βρίσκονται αποθηκευμένα και να γράφουμε ένα αποτέλεσμα στο ίδιο ή εναλλακτικά σε άλλο αρχείο. Αυτή η διεργασία ανάγνωσης και εγγραφής, ονομάζεται Είσοδος/Εξοδος Αρχείου και στην Python υλοποιείται μέσω ενσωματωμένων συναρτήσεων (μεθόδων).

Γενικά μπορούμε να πούμε ότι χρησιμοποιούμε δύο τύπους αρχείων: αυτά που περιέχουν το πρόγραμμα που θα εκτελέσουμε και εκείνα που περιέχουν τα δεδομένα, τα οποία, όταν εκτελεστεί το πρόγραμμα, τα διαβάζει και πιθανά τα ενημερώνει.

Ως αρχεία δεδομένων χρησιμοποιούνται συνήθως αρχεία κειμένου. Ένα **αρχείο κειμένου** είναι ένα αρχείο το οποίο περιέχει μια ακολουθία χαρακτήρων και βρίσκεται αποθηκευμένο σε ένα μέσο μόνιμης αποθήκευσης, όπως ο σκληρός δίσκος. Στα περιεχόμενα ενός τέτοιου αρχείου μπορούμε να έχουμε πρόσβαση μέσω ενός προγράμματος που περιέχει τις κατάλληλες εντολές της γλώσσας Python. Ένα αρχείο για να χρησιμοποιηθεί, πρέπει να το ανοίξουμε με την ενσωματωμένη συνάρτηση `open()` και στο τέλος να κλείσει με τη συνάρτηση `close()`.

Η συνάρτηση `open()` μας επιστρέφει ένα αντικείμενο του αρχείου και μπορούμε να το χρησιμοποιήσουμε για να εκτελέσουμε διάφορες λειτουργίες σε αυτό. Η συνάρτηση `open()` είναι ενσωματωμένη στην Python, δε χρειάζεται να φορτώσουμε κάποια βιβλιοθήκη, ενώ η σύνταξή της είναι:

`open("όνομα_αρχείου", "τρόπος προσπέλασης")`

Παρατηρούμε ότι η συνάρτηση `open()` δέχεται δυο ορίσματα: Το πρώτο είναι το όνομα του αρχείου, με το οποίο το αναγνωρίζει το λειτουργικό σύστημα. Το δεύτερο είναι ένα ειδικό σύμβολο (σημαία - flag) που καθορίζει τον τρόπο προσπέλασης του αρχείου, του οποίου οι επιτρεπόμενες τιμές της παραμέτρου για τον τρόπο προσπέλασης φαίνονται στον παρακάτω πίνακα 6.1. Αν δε χρησιμοποιηθεί το δεύτερο αυτό όρισμα, τότε θεωρείται εξ ορισμού (προεπιλογή) ότι είναι το "r".

Παράδειγμα

```
>>> fin = open('words.txt')
```

```
>>> print fin
```

Στην πρώτη γραμμή ανοίγουμε το αρχείο κειμένου “words.txt” και αποδίδουμε το αποτέλεσμα στον περιγραφέα αρχείου fin. Στη δεύτερη γραμμή τυπώνουμε τα χαρακτηριστικά του περιγραφέα fin. Όλες οι λειτουργίες που πρόκειται να εκτελεστούν στο αρχείο words.txt θα εκτελεστούν μέσω του περιγραφέα fin.

Πίνακας 6-1. Ορίσματα τρόπου προσπέλασης του αρχείου

Ορίσματα τρόπου προσπέλασης του αρχείου	Λειτουργία
“r”	Ανάγνωση
“w”	Εγγραφή (διαγραφή προηγούμενων περιεχομένων, αν υπάρχουν)
“a”	Προσθήκη (append) (διατήρηση προηγούμενων περιεχομένων)
“r+”	Άνοιγμα αρχείου και για ανάγνωση και για εγγραφή
“b”	Αρχείο δυαδικής μορφής

Μπορούμε να **δημιουργήσουμε** ένα αρχείο δεδομένων είτε χρησιμοποιώντας ένα συντάκτη (editor), όπως το notepad είτε τη συνάρτηση open με το όρισμα “w”, όπως για παράδειγμα: open(“words.txt”, “w”). Η συνάρτηση αυτή, αν δεν υπάρχει το αρχείο words.txt, τότε το δημιουργεί, ενώ αν αυτό υπάρχει, τότε θα έχουμε ως αποτέλεσμα να χαθούν τα περιεχόμενά του. Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε τη συνάρτηση με το όρισμα “a”, για παράδειγμα open(“words.txt”, “a”), η οποία, αν δεν υπάρχει το αρχείο, το δημιουργεί, ενώ, αν υπάρχει, το ανοίγει σε κατάσταση προσθήκης δεδομένων στο τέλος του.

Η γλώσσα Python διαθέτει μεθόδους για να επεξεργαστούμε τα αρχεία. Μπορούμε να ανοίξουμε και να χρησιμοποιήσουμε αρχεία χρησιμοποιώντας τις αντίστοιχες μεθόδους για ανάγνωση **read()** ή **readline()** και για εγγραφή **write()**, στο αρχείο. Η δυνατότητα να διαβάζουμε ή να γράφουμε στο αρχείο εξαρτάται από τον τρόπο προσπέλασης που έχουμε καθορίσει στη συνάρτηση **open()** κατά το άνοιγμά του.

Όταν ολοκληρώσουμε τις λειτουργίες που θέλουμε να εκτελεστούν στο αρχείο, καλούμε απαραίτητα τη συνάρτηση **close()**. Αυτό δηλώνει ότι τελειώσαμε με τη χρήση του και οδηγεί το Λειτουργικό Σύστημα στην αποθήκευση δεδομένων που βρίσκονται ακόμη στη μνήμη και την περάτωση των δικών του διεργασιών με το αρχείο. Η αντίστοιχη εντολή είναι:

```
>>> fin.close()
```

Η **close** κλείνει το αρχείο, παρόμοια με την επιλογή **File→Save** στον επεξεργαστή κειμένου.

Για να ελέγξουμε, αν ένα αρχείο όντως έκλεισε, πρέπει να χρησιμοποιήσουμε την ιδιότητα (attribute) **closed**, η οποία επιστρέφει την τιμή **True** ή **False**, ανάλογα, αν είναι κλειστό ή ανοιχτό το αρχείο.

Παράδειγμα

```
>>> fin = open("workfile.txt")
>>> fin.closed
# False
>>> fin.close()
>>> fin.closed
# True
```

6.2 Ανάγνωση και εγγραφή σε αρχείο

Στην ενότητα αυτή θα χρησιμοποιήσουμε για τα παραδείγματα ένα αρχείο, το **words.txt**, το οποίο έχει ως περιεχόμενο το:

```
This is line 1
This is line 2
This is line 3
```

Εγγραφή σε αρχείο

Υπενθυμίζουμε ότι για να γράψουμε σε ένα αρχείο, πρέπει πρώτα να το ανοίξουμε με το κατάλληλο όρισμα. Έτσι, με τη χρήση του ορίσματος 'w', θα διαγραφούν τα υπάρχοντα περιεχόμενα του αρχείου αν αυτό υπάρχει, ενώ με τη χρήση του 'a' τυχόν υπάρχοντα περιεχόμενα θα διατηρηθούν. Για την εγγραφή χρησιμοποιούμε τη μέθοδο **write()** και ως όρισμα τη συμβολοσειρά που θέλουμε να εισάγουμε στο αρχείο. Για παράδειγμα:

```
>>> fin = open("words.txt", "a")
>>> fin.write("This is line 4\n")
```

Το αρχείο words.txt έχει τώρα τη μορφή:

```
This is line 1
This is line 2
This is line 3
This is line 4
```

Σε ένα αρχείο κειμένου, αν θέλουμε να υπάρχουν γραμμές και όχι συνεχόμενοι χαρακτήρες, πρέπει να σημειώνουμε την αλλαγή γραμμής με το χαρακτήρα **"\n"**.

Το όρισμα της μεθόδου write πρέπει να είναι **συμβολοσειρά**. Αν θέλουμε να εισάγουμε κάτι διαφορετικό, όπως για παράδειγμα έναν αριθμό, ο ευκολότερος τρόπος είναι, να μετατρέψουμε τον αριθμό σε συμβολοσειρά, με χρήση της συνάρτησης str.

```
>>> x = 52
>>> keimeno.write(str(x))
```

Ανάγνωση περιεχομένων ενός αρχείου

Οι πιο διαδεδομένες μέθοδοι για διάβασμα των περιεχομένων ενός αρχείου, είναι η **read()** και η **readline()**, όπου διαβάζει ένα πλήθος χαρακτήρων από την αρχή του αρχείου.

Σύνταξη: `fileObject_descriptor.read([count]);`

Εδώ, η παράμετρος count καθορίζει τον αριθμό των χαρακτήρων που θα διαβασθούν από το αρχείο, ξεκινώντας την ανάγνωση από την αρχή του μέχρι το πλήθος των χαρακτήρων που ορίζονται με

την `count`. Στην περίπτωση που η παράμετρος `count` λείπει, τότε γίνεται ανάγνωση μέχρι να διαβαστεί η ένδειξη τέλους του αρχείου.

Παράδειγμα 1: Ανάγνωση και εμφάνιση ενός μόνο χαρακτήρα

```
>>> fin = open('words.txt', 'r')
```

```
>>> print fin.read(1)
```

T

Παράδειγμα 2: Διάβασμα και εμφάνιση των επόμενων 13 χαρακτήρων

```
>>> print fin.read(13)
```

his is line 1

Παράδειγμα 3: Εμφάνιση ολόκληρου του αρχείου

```
>>> print fin.read()
```

This is line 1

This is line 2

This is line 3

Αν η εντολή `print fin.read()` εκτελεστεί, αφού έχουν προηγηθεί οι προηγούμενες δύο εντολές, τότε αυτή θα επιστρέψει το υπόλοιπο τμήμα του αρχείου, δηλαδή τις δύο τελευταίες γραμμές.

Η **`readline`** διαβάζει μια γραμμή του αρχείου, δηλαδή διαβάζει διαδοχικούς χαρακτήρες από ένα αρχείο μέχρι να συναντήσει το χαρακτήρα νέας γραμμής και επιστρέφει το αποτέλεσμα:

Παράδειγμα

```
>>> fin.close()
```

```
>>> fin = open("words.txt")
```

```
>>> print fin.readline()
```

This is line 1

Το `fin` καταγράφει τη θέση που βρίσκεται μέσα στο αρχείο και έτσι αν ξανακαλέσουμε τη `readline` θα πάρουμε την επόμενη γραμμή:

```
>>> print fin.readline()
```

This is line 2

Η συνάρτηση `close()` αναλαμβάνει να κλείσει το αρχείο και να απελευθερώσει έτσι πόρους του συστήματος.

Παράδειγμα

Για ανάγνωση αλλά και εγγραφή, κατά γραμμές, από ένα αρχείο μπορούμε να σαρώσουμε με μια δομή επανάληψης το αρχείο. Για παράδειγμα για να τυπώσουμε τις γραμμές του αρχείου γράφουμε:

```
>>> for line in fin:
    print line,
```

Εντοπισμός θέσης στο αρχείο

Η μέθοδος **`fin.tell()`** επιστρέφει έναν ακέραιο που περιέχει την τρέχουσα θέση στο αρχείο, υπολογισμένη σε χαρακτήρες (bytes) από την αρχή του αρχείου. Με άλλα λόγια, η επόμενη ανάγνωση ή εγγραφή θα γίνει σε εκείνη τη θέση του αρχείου.

Για να αλλάξουμε την τρέχουσα θέση του αρχείου, μπορούμε να χρησιμοποιήσουμε την **`fin.seek() (offset[, from_what])`**. Η θέση υπολογίζεται προσθέτοντας `offset` (πλήθος bytes) σε ένα σημείο αναφοράς, το οποίο επιλέγεται από το `from_what` όρισμα. Αν το `from_what` έχει τιμή 0, μετρά από την αρχή του αρχείου, αν έχει 1, χρησιμοποιεί την τρέχουσα θέση του αρχείου και αν έχει 2, χρησιμοποιεί το τέλος του αρχείου.

```
>>> fin = open("workfile", "r+")
>>> fin.write("0123456789abcdef")
>>> fin.seek(5)    # πηγαίνει στο 6ο byte στο αρχείο
>>> fin.read(1)
'5'
>>> fin.seek(-3, 2) # πηγαίνει στο 3ο byte πριν το τέλος
>>> fin.read(1)
'd'
```

6.3 Πρόσθετες λειτουργίες σε αρχεία

Όταν ένα αρχείο έχει ανοίξει, μπορούμε να ανακτήσουμε διάφορες χρήσιμες πληροφορίες σχετικά με αυτό. Ακολουθεί μια λίστα με όλα τα σχετικά χαρακτηριστικά:

Χαρακτηριστικό	Περιγραφή
file.closed	Επιστρέφει true, αν το αρχείο είναι κλειστό, false, σε διαφορετική περίπτωση
file.mode	Επιστρέφει τον τρόπο προσπέλασης στο αρχείο που έχουμε ανοίξει
file.name	Επιστρέφει το όνομα του αρχείου

Παράδειγμα

```
fin= open("foo.txt", "wb")  
print "Όνομα του αρχείου: ", fin.name  
print "Κλειστό: ", fin.closed  
print "Τρόπος προσπέλασης: ", fin.mode
```

Αυτό παράγει το ακόλουθο αποτέλεσμα:

Όνομα του αρχείου: foo.txt

Κλειστό: False

Τρόπος προσπέλασης: wb

Όνομα αρχείου και η θέση του στο μέσο

Τα αρχεία αποθηκεύονται στο φυσικό μέσο από το Λειτουργικό Σύστημα (ΛΣ) οργανωμένα σε καταλόγους, που είναι γνωστοί και ως φάκελοι-folders.

Κάθε δε αρχείο διαθέτει από το ΛΣ μια μοναδική διαδρομή του τύπου:

φάκελος\υποφάκελος\.. \όνομα_αρχείου.

Για την Python, αυτό αποτελεί μια συμβολοσειρά και προσδιορίζει το κάθε αρχείο. Μια τέτοια διαδρομή, η οποία ξεκινάει από τον τρέχοντα κατάλογο, καλείται *σχετική*, ενώ αυτή η οποία ξεκινάει από τον ανώτατο κατάλογο στο σύστημα αρχείων καλείται *απόλυτη* διαδρομή. Όπως κάθε περιβάλλον, έτσι και η Python, έχει έναν προεπιλεγμένο κατάλογο, για αναζήτηση αρχείων, που ονομάζεται τρέχων φάκελος. Έτσι, για παράδειγμα, όταν ανοίγουμε ένα αρχείο για διάβασμα, τότε η Python το αναζητά στον τρέχοντα κατάλογο.

Η **μονάδα λογισμικού os** (os: από το operating system) παρέχει συναρτήσεις, για να μπορούμε να δουλέψουμε με αρχεία και καταλόγους. Για να χρησιμοποιήσουμε αυτό το άρθρωμα (module) απαιτείται πρώτα να το εισάγουμε και έπειτα να καλέσουμε όλες τις σχετικές συναρτήσεις.

```
>>> import os
```

Μετονομασία και διαγραφή αρχείων. Η μέθοδος **rename()**

Η **rename()** αποτελεί μέθοδο της Python για μετονομασία αρχείου και δέχεται δύο ορίσματα, ένα για το τρέχον και ένα για το νέο όνομα του αρχείου.

Σύνταξη: `os.rename(current_file_name, new_file_name)`

Παράδειγμα

```
import os
```

```
# Μετονομασία ενός αρχείου από test1.txt σε test2.txt
```

```
os.rename("test1.txt", "test2.txt")
```

Η μέθοδος **remove()**

Αποτελεί μέθοδο για τη διαγραφή αρχείων και δέχεται ως όρισμα το όνομα του αρχείου προς διαγραφή.

Σύνταξη: `os.remove(file_name)`

Παράδειγμα

```
import os
```

```
# Διαγραφή αρχείου test2.txt
```

```
os.remove("text2.txt")
```

Διαχείριση καταλόγων

Η Python χειρίζεται τους καταλόγους (φακέλους) με τη βοήθεια του `os module` παρέχοντας αρκετές μεθόδους, όπως για τη δημιουργία, διαγραφή και την αλλαγή καταλόγων.

Η μέθοδος **`mkdir()`**

Δημιουργεί (υπο)καταλόγους στον τρέχοντα κατάλογο και δέχεται ως όρισμα το όνομα του καταλόγου που θα δημιουργηθεί.

Σύνταξη: `os.mkdir("newdir")`

Παράδειγμα

```
import os
```

```
os.mkdir("test")
```

Η μέθοδος **`chdir()`**, αλλάζει τον τρέχοντα κατάλογο.

Σύνταξη: `os.chdir("newdir")`

Παράδειγμα

```
import os
```

```
os.chdir("/home/newdir")
```

Η μέθοδος **`getcwd()`**: Επιστρέφει τον τρέχοντα κατάλογο εργασίας.

Σύνταξη: `os.getcwd()`

Παράδειγμα

```
import os
```

```
os.getcwd()
```

Η μέθοδος **`rmdir()`**: Διαγράφει το φάκελο που δίνουμε ως όρισμα. Πριν διαγραφεί ο φάκελος, όλα τα περιεχόμενά του διαγράφονται.

Σύνταξη: `os.rmdir('dirname')`

Παράδειγμα

```
import os  
os.rmdir( "/tmp/test" )
```

Δραστηριότητες - Ερωτήσεις - Ασκήσεις κεφαλαίου**6.4 Δραστηριότητες****Δραστηριότητα 1**

Ας πειραματιστούμε γράφοντας κώδικα σε Python, ο οποίος διαβάζει το αρχείο "input_file.txt" και το ξαναγράφει στο νέο αρχείο "output_file.txt", όπου πριν από κάθε γραμμή θα προσθέτει τον αύξοντα αριθμό της. Τα αρχεία θεωρούμε ότι βρίσκονται στον τρέχοντα κατάλογο "TxtFiles".

```
inputfile = open("TxtFiles/input_file.txt","r")  
outputFile = open("TxtFiles/output_file.txt","w")  
linecounter = 1  
  
for line in inputfile:  
    outputFile.write(str(linecounter)+". "+line)  
    linecounter = linecounter + 1  
  
inputfile.close()  
outputFile.close()
```

Δραστηριότητα 2

Υποθέτουμε ότι έχουμε ένα πρόγραμμα σε Python που έχει αποθηκευθεί στο αρχείο με όνομα demo.py και με τον ακόλουθο κώδικα:

```
def linecount(filename):  
    count = 0  
    for line in open(filename):  
        count += 1  
    return count  
print linecount('demo.py')
```

Εκτελέστε το παραπάνω πρόγραμμα και συζητήστε το αποτέλεσμα. Υπόδειξη: Το αποτέλεσμα είναι ότι θα εμφανίσει το πλήθος των γραμμών του αρχείου.

Δραστηριότητα 3

Να γράψετε πρόγραμμα στη γλώσσα Python, το οποίο θα δέχεται ως είσοδο το όνομα ενός αρχείου, θα εμφανίζει τα περιεχόμενά του κατά γραμμή και στη συνέχεια θα γράφει σε ένα άλλο αρχείο, τις γραμμές του αρχείου στην αντίστροφη σειρά.

Δραστηριότητα 4

Τι πιστεύετε ότι θα συμβεί, όταν θα εκτελεστούν τα παρακάτω σενάρια. Τεκμηριώστε την άποψή σας.

```
my_list = [i**2 for i in range(1,11)]
f = open("output.txt", "w")
for item in my_list:
    f.write(str(item) + "\n") # δέχεται string όρισμα η write
f.close()

f = open("output.txt", "r")
print f.readline()
print f.read()
f.close()
```

6.4 Ερωτήσεις - Ασκήσεις

1. Σε τι χρησιμεύει ένα αρχείο δεδομένων;
2. Ποιες βασικές λειτουργίες - μεθόδους υποστηρίζει η γλώσσα προγραμματισμού Python για τη διαχείριση αρχείων;
3. Με ποια συνάρτηση (μέθοδο) ανοίγουμε ένα αρχείο για ανάγνωση;

4. Πώς ανοίγουμε ένα αρχείο για εγγραφή;
5. Με ποια συνάρτηση κλείνουμε ένα αρχείο;
6. Με ποια συνάρτηση γράφουμε περιεχόμενο σε ένα αρχείο;
7. Ποια συνάρτηση μπορούμε να χρησιμοποιήσουμε για να διαβάσουμε δεδομένα από μία γραμμή, από ένα αρχείο;

Σύνοψη

Στο κεφάλαιο αυτό ασχοληθήκαμε με τη διαχείριση αρχείων δεδομένων, κυρίως από τη σκοπιά της γλώσσας προγραμματισμού Python.

Επίσης, με θέματα ταυτοποίησής τους από το Λειτουργικό Σύστημα, όπως το όνομα, η θέση στο αποθηκευτικό μέσο και βασικές λειτουργίες σε αυτά.

7

**Προηγμένα στοιχεία
γλώσσας
προγραμματισμού**

7. Προηγμένα στοιχεία γλώσσας προγραμματισμού

Εισαγωγή

Στο κεφάλαιο αυτό θα αναπτυχθούν προηγμένα χαρακτηριστικά της γλώσσας προγραμματισμού Python. Συγκεκριμένα θα αναπτυχθούν οι ιδιότητες των υποπρογραμμάτων, η εμβέλεια μεταβλητών, τα άρθρωματά και τα πακέτα.

Διδακτικοί στόχοι

Μετά τη μελέτη του κεφαλαίου θα μπορούμε να:

- αναφέρουμε τα χαρακτηριστικά των υποπρογραμμάτων και τον τρόπο κλήσης τους, με την προσαρμογή τους στη γλώσσα προγραμματισμού
- εντοπίζουμε και απομονώνουμε κατάλληλα τμήματα κώδικα από έτοιμο πρόγραμμα για μετατροπή σε υποπρογράμματα
- σχεδιάζουμε τη λύση προγραμματιστικού προβλήματος με αρθρωτό τρόπο, όπου αυτό είναι σκόπιμο
- αναγνωρίζουμε την εμβέλεια μεταβλητών και παραμέτρων σε σύνθετα προγράμματα.

Λέξεις κλειδιά

Συνάρτηση, παράμετρος, εμβέλεια, άρθρωμα, τοπική μεταβλητή.

Διδακτικές Ενότητες

7.1 Υποπρογράμματα και τρόποι κλήσης τους

7.1.1 Υποπρογράμματα

Μεταξύ των θεμάτων της αλγοριθμικής επίλυσης προβλημάτων που έχουμε ήδη εξετάσει, είναι και εκείνα της ανάγκης επανάληψης κώδικα με τη χρήση μιας δομής επανάληψης ή ομαδοποίησης και χειρισμού πολλών δεδομένων με τη χρήση άλλων δομών δεδομένων, όπως η Λίστα.

Προχωρώντας σε πιο πολύπλοκα προβλήματα, θα δούμε ότι συχνά κάποια ενέργεια ή και ολόκληρο αλγοριθμικό τμήμα, είναι αναγκαίο να επαναλαμβάνεται, σχεδόν αυτούσια ως λογική επεξεργασία, σε πολλά διαφορετικά σημεία του αλγορίθμου, αλλά κάθε φορά να επεξεργάζεται διαφορετικά δεδομένα. Ο τρόπος

αυτός “επανάληψης” δεν μπορεί να υλοποιηθεί με μια δομή επανάληψης, λόγω της δυνατότητας επεξεργασίας διαφορετικών δεδομένων κάθε φορά. Φυσικά δεν είναι προγραμματιστικά ορθό να επαναλαμβάνουμε το τμήμα κώδικα με άλλα δεδομένα κάθε φορά, αλλά να επιτρέπει η γλώσσα προγραμματισμού τη δημιουργία ομάδας εντολών, ως οντότητα στο πρόγραμμα, που να δέχεται ως είσοδο παραμέτρους και να μπορεί να καλείται από σημεία του προγράμματος και με συγκεκριμένες τιμές για τις παραμέτρους ανά κλήση. Αυτό έχει ως αποτέλεσμα τη δυνατότητα πολλαπλής διενέργειας μιας επεξεργασίας με συγκεκριμένα δεδομένα κάθε φορά.

Μια από τις βασικότερες τεχνικές του διαδικαστικού προγραμματισμού είναι ο **Τμηματικός Προγραμματισμός**. Σύμφωνα με την τεχνική αυτή, μπορούμε να γράψουμε ένα πρόγραμμα ως ένα σύνολο από μικρότερα κομμάτια προγράμματος. Με αυτόν τον τρόπο δεν είναι απαραίτητο να ξαναγράφουμε τον ίδιο κώδικα σε διαφορετικές υλοποιήσεις. Τον γράφουμε μια φορά, δημιουργώντας ένα *υποπρόγραμμα* και τον χρησιμοποιούμε, όσες φορές θέλουμε σε ένα πρόγραμμα, καλώντας το αντίστοιχο υποπρόγραμμα. Ένα υποπρόγραμμα λοιπόν είναι ένα κομμάτι προγράμματος που έχει γραφεί ξεχωριστά από το υπόλοιπο πρόγραμμα και επιτελεί ένα αυτόνομο έργο.

Κάθε υποπρόγραμμα πρέπει να έχει τα παρακάτω βασικά χαρακτηριστικά:

- α) Έχει μόνο ένα σημείο εισόδου από το οποίο δέχεται τα δεδομένα του.
- β) Το (υπο)πρόγραμμα το οποίο καλεί ένα άλλο υποπρόγραμμα σταματάει την εκτέλεσή του όσο εκτελείται το καλούμενο υποπρόγραμμα. Μόνο ένα υποπρόγραμμα μπορεί να εκτελείται σε μια χρονική στιγμή.
- γ) Ο έλεγχος επιστρέφει στο (υπο)πρόγραμμα το οποίο καλεί, όταν το καλούμενο υποπρόγραμμα σταματήσει να εκτελείται.

Καλές πρακτικές

Πριν ξεκινήσουμε να γράφουμε ένα πρόγραμμα, μελετάμε πώς αυτό μπορεί να *αναλυθεί* σε επιμέρους τμήματα και αποφασίζουμε για τα αντίστοιχα υποπρογράμματα.

Εξετάζουμε αν κάποια υποπρογράμματα, τα οποία έχουμε ήδη γράψει ή υπάρχουν σε έτοιμες *βιβλιοθήκες* προγραμμάτων, μπορούν να χρησιμοποιηθούν, για να κερδίσουμε χρόνο και να αποφύγουμε λάθη.

Προσπαθούμε κάθε υποπρόγραμμα να είναι όσο το δυνατόν πιο *ανεξάρτητο* από τα άλλα. Αυτό μας προφυλάσσει από λάθη στο πρόγραμμά μας και επιτρέπει τη χρήση του σε άλλα προγράμματα αργότερα.

Πολλά από τα ανθρώπινα τεχνουργήματα, φυσικά ή τεχνητά, στηρίζονται στην ιδέα της επαναχρησιμοποίησης – συνδυασμού μικρότερων δομικών μονάδων, με αποτέλεσμα την απλότητα και την ποικιλία αντικειμένων και *χαρακτηριστικών*. Στην Python, η τεχνική αυτή υλοποιείται με τις συναρτήσεις, δίνοντας καλύτερο έλεγχο και *υψηλό επίπεδο αφαίρεσης*.

7.1.2 Συναρτήσεις στην Python

Κάθε γλώσσα προγραμματισμού διαθέτει ένα δικό της λεξιλόγιο για τις εντολές και τις ενσωματωμένες συναρτήσεις, ενώ *προγράμματα* που γράφονται πρέπει να ακολουθούν αυστηρούς γραμματικούς και συντακτικούς κανόνες για τη δόμηση των εντολών, τη δημιουργία συναρτήσεων χρήστη κ.ά. Στη συνέχεια θα δούμε πώς υλοποιείται ο Τμηματικός προγραμματισμός στη γλώσσα προγραμματισμού Python.

Η Python παρέχει ένα μόνο τύπο υποπρογραμμάτων, τις **συναρτήσεις**, τις οποίες τις θεωρεί ως *αντικείμενα*.

Για να εξασκηθούμε μπορούμε να δημιουργήσουμε, σε πρώτο επίπεδο, τις δικές μας συναρτήσεις και έτσι, να κατανοήσουμε τη λειτουργία και τη χρήση τους. Σε υψηλότερο επίπεδο, μπορούμε να χρησιμοποιούμε και να ενσωματώνουμε στο πρόγραμμα έτοιμες βιβλιοθήκες, που είτε παρέχει το περιβάλλον ή υπάρχουν

σε άλλα συστήματα. Έτσι, δεν είναι ανάγκη να ξαναγράψουμε κώδικα που υπάρχει διαθέσιμος, μειώνοντας την πολυπλοκότητα και την πιθανότητα σφαλμάτων (bugs).

Ορισμός και κλήση Συνάρτησης

Ο ορισμός μιας συνάρτησης περιλαμβάνει το όνομά της και τις παραμέτρους εισόδου και καλείται από σημεία του προγράμματος μέσω της λειτουργίας που ονομάζεται **κλήση** (calling) της συνάρτησης. Ορίζεται με τη λέξη κλειδί **def** που την ακολουθεί ένα όνομα το οποίο την ταυτοποιεί και ένα ζεύγος παρενθέσεων που μπορούν να περιέχουν ονόματα μεταβλητών, ενώ η δήλωση τελειώνει με διπλή τελεία (:). Κάτω από τη γραμμή αυτή τοποθετούνται, σε εσοχή, οι εντολές που καθορίζουν τη λειτουργία της συνάρτησης. Ο ορισμός συναρτήσεων στην Python είναι απλός σε σχέση με άλλες γλώσσες (ο τύπος των παραμέτρων δε δηλώνεται).

Παράδειγμα ορισμού συνάρτησης χωρίς χρήση παραμέτρων

```
def fun_name():  
    print "hello"
```

Η συνάρτηση αυτή, όταν κληθεί, εμφανίζει στην οθόνη τη λέξη hello.

Παράδειγμα, ορισμού συνάρτησης με χρήση δύο παραμέτρων

```
def find_sum(par1, par2):  
    result = par1+par2  
    return result
```

Η συνάρτηση αυτή επιστρέφει το άθροισμα των τιμών των δύο παραμέτρων.

Μια συνάρτηση μπορεί να καλείται από διάφορα σημεία του κύριου προγράμματος ή και μέσα από μια άλλη συνάρτηση, γράφοντας το όνομά της και τις κατάλληλες παραμέτρους μέσα σε παρενθέσεις.

Παράδειγμα 1: Κλήση συνάρτησης

```
>>> type(45)
<type 'int'>
```

Εδώ, το όνομα της συνάρτησης είναι `type` και η έκφραση μέσα στις παρενθέσεις, που ονομάζεται παράμετρος της συνάρτησης, είναι το 45, ενώ η συνάρτηση υπολογίζει και επιστρέφει τον τύπο του ορίσματος. Είναι σύνηθες να λέμε πως η συνάρτηση δέχεται ορίσματα και επιστρέφει ένα αποτέλεσμα. Το αποτέλεσμα αυτό ονομάζεται **επιστρεφόμενη τιμή** (return value).

Παράδειγμα 2

Κλήση της συνάρτησης `find_sum`, που δημιουργήσαμε παραπάνω:

```
>>> find_sum(3,4)
7
```

Εδώ, η συνάρτηση καλείται με το όνομά της και με παραμέτρους τις τιμές 3 και 4, ενώ επιστρέφει το άθροισμά τους που είναι η τιμή 7.

Αν καλέσουμε πάλι την ίδια συνάρτηση “περνώντας” τώρα διαφορετικές τιμές στις παραμέτρους, για παράδειγμα δύο συμβολοσειρές, η συνάρτηση επιστρέφει τη συνένωση των συμβολοσειρών αυτών ως αποτέλεσμα του **πολυμορφισμού** που ισχύει στον τελεστή + στην Python.

```
>>> find_sum("well","come")
"wellcome"
```

Σημείωση: Μια συνάρτηση πρέπει να έχει οριστεί πριν χρησιμοποιηθεί. Είναι προφανές ότι οι δηλώσεις (statements) μέσα στη συνάρτηση δεν εκτελούνται μέχρι αυτή να κληθεί.

Ροή εκτέλεσης

Ένας τρόπος να διαβάζουμε προγράμματα είναι να ακολουθούμε τη ροή εκτέλεσης των εντολών, κάτι το οποίο μπορεί να γίνει γρήγορα πολύπλοκο και δυσανάγνωστο. Ένας εναλλακτικός τρόπος είναι, όταν βρεθούμε σε μια κλήση συνάρτησης, αντί να ακολουθήσουμε τη *ροή της εκτέλεσης* στον εσωτερικό κώδικα της συνάρτησης, να υποθέτουμε ότι η συνάρτηση δουλεύει σωστά και επιστρέφει το σωστό αποτέλεσμα. Οπότε, να συνεχίζουμε με την

επόμενη εντολή του κυρίου προγράμματος. Στην πραγματικότητα εφαρμόζουμε κάτι ανάλογο με τη χρησιμοποίηση των ενσωματωμένων συναρτήσεων. Έτσι, όταν καλούμε τη `math.cos` ή `math.exp`, δεν εξετάζουμε τον εσωτερικό κώδικα αυτών των συναρτήσεων, αλλά θεωρούμε ότι δουλεύουν σωστά και δεχόμαστε το αποτέλεσμα. Το ίδιο ισχύει και όταν καλούμε μια από τις δικές μας συναρτήσεις.

Μια κλήση συνάρτησης είναι σαν μια παράκαμψη στη ροή της εκτέλεσης. Αντί η ροή να πάει στην επόμενη δήλωση, περνάει στο σώμα της συνάρτησης, εκτελεί όλες τις δηλώσεις εκεί και μετά επιστρέφει για να συνεχίσει από εκεί που σταμάτησε.

Κατηγορίες συναρτήσεων

Οι συναρτήσεις μπορούν να κατηγοριοποιηθούν με πολλούς τρόπους.

Μια πρώτη κατηγορία συναρτήσεων είναι:

α) αυτές οι οποίες δεν τροποποιούν το αντικείμενο στο οποίο εφαρμόζονται, όπως:

```
>>> a = 'Python'
>>> print(a.upper( ))
PYTHON
>>> print(a)
Python
```

β) εκείνες που μπορούν να αλλάξουν το αντικείμενο στο οποίο καλούνται, όπως στο παράδειγμα 2.

```
>>> b = ['a', 'b', 'c', 'd']
>>> print(b.reverse( ))
None
>>> print(b)
['d', 'c', 'b', 'a']
```

Μια άλλη κατηγορία συναρτήσεων είναι:

α) αυτές οι οποίες, όταν κληθούν, επιστρέφουν αποτέλεσμα (κάποια τιμή)

β) εκείνες που δεν επιστρέφουν κάποια τιμή (κενός/void συναρτήσεις), αλλά εκτελούν ενέργειες μέσω των εντολών τους.

Οι κενές αυτές συναρτήσεις μπορεί να εμφανίζουν αποτέλεσμα στην οθόνη ή να έχουν κάποιο άλλο αποτέλεσμα, αλλά δεν επιστρέφουν κάποια τιμή.

7.2 Μεταβλητές και παράμετροι

Όπως έχουμε αναφέρει, ένα υποπρόγραμμα αποτελεί ένα ανεξάρτητο τμήμα προγράμματος που μπορεί να καλείται από σημεία του προγράμματος. Δέχεται τιμές από το τμήμα προγράμματος που το καλεί (το κύριο πρόγραμμα ή άλλο υποπρόγραμμα) και μετά την εκτέλεση των εντολών του επιστρέφει σε αυτό νέες τιμές, δηλαδή τα αποτελέσματα. Οι τιμές αυτές που μεταβιβάζονται από το ένα υποπρόγραμμα στο άλλο λέγονται **παράμετροι** και γενικά σε γλώσσες προγραμματισμού διακρίνονται σε *παραμέτρους Εισόδου και Εξόδου*. Οι παράμετροι είναι μεταβλητές οι οποίες χρησιμοποιούνται για τη μεταβίβαση τιμών μεταξύ υποπρογραμμάτων ή υποπρογραμμάτων και κύριου προγράμματος.

7.2.1 Παράμετροι συναρτήσεων

Μια συνάρτηση μπορεί να δεχθεί παραμέτρους, οι οποίες είναι τιμές που δίνονται/εισέρχονται στη συνάρτηση, έτσι ώστε αυτή να μπορεί να λειτουργήσει αξιοποιώντας αυτές τις τιμές. Οι παράμετροι μοιάζουν με τις μεταβλητές, καθορίζονται μέσα στο ζευγάρι των παρενθέσεων, στον ορισμό της συνάρτησης και διαχωρίζονται με κόμμα, ενώ οι τιμές αυτών των μεταβλητών ορίζονται, όταν καλούμε τη συνάρτηση.

Παράδειγμα

```
def printMax(a, b):  
    if a > b:  
        print(a, 'είναι το μέγιστο')  
    elif a == b:  
        print(a, 'είναι ίσο με το', b)  
    else:  
        print(b, 'είναι το μέγιστο')
```

```
printMax(3, 4) # δίνουμε απ' ευθείας σταθερές τιμές
x = 5
y = 7
printMax(x, y) # δίνουμε μεταβλητές ως ορίσματα
Έξοδος:
4 είναι το μέγιστο
7 είναι το μέγιστο
```

Μέσα στη συνάρτηση, τα ορίσματα εκχωρούνται σε μεταβλητές οι οποίες ονομάζονται *παράμετροι*. Αυτό είναι ένα παράδειγμα συνάρτησης ορισμένης από το χρήστη, η οποία παίρνει ένα όρισμα:

```
def print_twice(x):
    print x
    print x
```

Αυτή η συνάρτηση εκχωρεί το όρισμα σε μια παράμετρο με όνομα *x*. Όταν καλείται η συνάρτηση, εμφανίζει την τιμή της παραμέτρου, όποια κι αν είναι αυτή, δύο φορές. Η συνάρτηση αυτή δουλεύει με οποιαδήποτε τιμή μπορεί να εμφανιστεί.

```
>>> print_twice('Spam')
```

```
Spam
```

```
Spam
```

```
>>> print_twice(25)
```

```
25
```

```
25
```

```
>>> print_twice(math.pi)
```

```
3.14159265359
```

```
3.14159265359
```

Όταν οι παράμετροι περνάνε με **αναφορά**, σημαίνει ότι αν αλλάξουμε μια παράμετρο μέσα στη συνάρτηση, η αλλαγή είναι μόνιμη και μετά την κλήση της συνάρτησης. Για παράδειγμα:

```
# Ορισμός συνάρτησης
```

```
def changeme( mylist ):
```

```
    # "Αλλάζει τη λίστα που περνά στη συνάρτηση"
```

```
    mylist.append([1,2,3,4]);
```

```
print "Τιμές μέσα στη συνάρτηση: ", mylist
return
```

```
# Τώρα μπορούμε να καλέσουμε τη changeme συνάρτηση
mylist = [10,20,30];
changeme( mylist );
print "Τιμές έξω από τη συνάρτηση: ", mylist
```

Οπότε, θα παραχθεί το ακόλουθο αποτέλεσμα:

Τιμές μέσα στη συνάρτηση: [10, 20, 30, [1, 2, 3, 4]]

Τιμές έξω από τη συνάρτηση: [10, 20, 30, [1, 2, 3, 4]]

Ας προσέξουμε το ακόλουθο παράδειγμα, όπου το όρισμα παρακάμπτεται μέσα στην καλούμενη συνάρτηση.

```
# Ορισμός συνάρτησης
def changeme( mylist ):
    mylist = [1,2,3,4]; # Δημιουργεί νέα αναφορά στη mylist
    print "Τιμές μέσα στη συνάρτηση: ", mylist
    return
# Τώρα μπορούμε να καλέσουμε τη changeme συνάρτηση
mylist = [10,20,30];
changeme( mylist );
print "Τιμές έξω από τη συνάρτηση: ", mylist
```

Η παράμετρος mylist είναι τοπική στη συνάρτηση changeme.

Αλλάζοντας την τιμή της mylist μέσα στη συνάρτηση, η αλλαγή αυτή εφαρμόζεται μόνο στην "περιοχή" της συνάρτησης και δεν αλλάζει την τιμή της παραμέτρου mylist στο πρόγραμμα. Οπότε τελικά θα παραχθεί το ακόλουθο αποτέλεσμα:

Τιμές μέσα στη συνάρτηση: [1, 2, 3, 4]

Τιμές έξω από τη συνάρτηση: [10, 20, 30]

7.2.2 Εμβέλεια των μεταβλητών

Όλες οι μεταβλητές σε ένα πρόγραμμα δεν μπορούν να είναι προσπελάσιμες από όλα τα μέρη του προγράμματος. Η **εμβέλεια** (scope) μιας μεταβλητής αναφέρεται στο τμήμα του

προγράμματος που μπορεί αυτή να έχει πρόσβαση. Οι γλώσσες προγραμματισμού επιτρέπουν τη χρήση των μεταβλητών, όχι μόνο στο τμήμα προγράμματος που δηλώνονται, αλλά και σε άλλα υποπρογράμματα και πιο συγκεκριμένα ισχύει:

Απεριόριστη εμβέλεια: Όλες οι μεταβλητές είναι ορατές και μπορούν να χρησιμοποιούνται σε οποιοδήποτε τμήμα του προγράμματος, ανεξάρτητα από το πού δηλώθηκαν. Αυτού του τύπου οι μεταβλητές χαρακτηρίζονται ως **καθολικές** (global). Το μειονέκτημα είναι ότι περιορίζεται έτσι η ανεξαρτησία των υποπρογραμμάτων.

Περιορισμένη εμβέλεια: Αυτές οι μεταβλητές είναι **τοπικές** (local), ισχύουν δηλαδή για το υποπρόγραμμα στο οποίο δηλώθηκαν. Η περιορισμένη εμβέλεια απαιτεί όλες οι μεταβλητές που χρησιμοποιούνται σε ένα τμήμα προγράμματος, να δηλώνονται σε αυτό το τμήμα.

Θα αναλύσουμε στη συνέχεια τους δύο βασικούς τύπους εμβέλειας, τις καθολικές και τις τοπικές μεταβλητές, στην Python.

Καθολικές και Τοπικές μεταβλητές

Οι μεταβλητές που έχουν οριστεί μέσα στο σώμα της συνάρτησης, έχουν τοπική εμβέλεια, ενώ αυτές που ορίστηκαν έξω από αυτό έχουν καθολική εμβέλεια. Αυτό σημαίνει ότι οι τοπικές μεταβλητές μπορούν να προσπελαστούν μόνο μέσα στη συνάρτηση στην οποία δηλώθηκαν, ενώ οι καθολικές μεταβλητές μπορεί να είναι προσβάσιμες από όλες τις συναρτήσεις. Ακολουθεί ένα απλό παράδειγμα.

```
total = 0; # Αυτή είναι μια καθολική μεταβλητή.
```

```
def sum( arg1, arg2 ):
    total = arg1 + arg2; # Η total είναι τοπική μεταβλητή.
    print "Μέσα στη συνάρτηση η τοπική total : ", total
    return total;
```

```
# Κλήση της sum συνάρτησης
sum( 10, 20 );
print "Έξω από τη συνάρτηση η καθολική total : ", total
```

Όταν εκτελεσθεί ο παραπάνω κώδικας, παράγεται το ακόλουθο αποτέλεσμα:

Μέσα στη συνάρτηση η τοπική total : 30

Έξω από τη συνάρτηση η καθολική total : 0

Τοπικές μεταβλητές

Όταν δηλώνουμε μεταβλητές μέσα σε έναν ορισμό συνάρτησης, αυτές δεν έχουν καμία σχέση με άλλες μεταβλητές που έχουν την ίδια ονομασία και χρησιμοποιούνται έξω από αυτήν τη συνάρτηση. Έτσι, σε μια συνάρτηση τα ονόματα των μεταβλητών χρησιμοποιούνται μόνο τοπικά. Όλες οι μεταβλητές έχουν την εμβέλεια του τμήματος κώδικα όπου έχουν δηλωθεί, αρχίζοντας από το σημείο στο οποίο ορίζεται το όνομα. Όπως αναφέρθηκε, η εμβέλεια μιας μεταβλητής ή μιας συνάρτησης, ορίζει την περιοχή του κώδικα στον οποίο αυτή έχει πρόσβαση. Οι τοπικές μεταβλητές που ορίζονται σε μια συνάρτηση, χάνονται όταν τελειώσει η εκτέλεση της συνάρτησης, ενώ η κλήση μιας συνάρτησης δημιουργεί νέες τοπικές μεταβλητές. Άλλωστε και οι παράμετροι είναι τοπικές μεταβλητές.

Παράδειγμα

x = 50

```
def func(x):
    print('Το x είναι', x)
    x = 2
    print('Το τοπικό x άλλαξε σε', x)
```

```
func(x)
print('Το x είναι ακόμα', x)
```

Έξοδος:

Το x είναι 50

Το τοπικό x άλλαξε σε 2

Το x είναι ακόμα 50

Χρήση της εντολής global

Οι μεταβλητές που δηλώνονται έξω από τις συναρτήσεις του προγράμματος, είναι **καθολικές μεταβλητές** (global) και

προσπελαύνονται από οπουδήποτε σημείο μέσα στο πρόγραμμα. Εάν θέλουμε μέσα σε μια συνάρτηση να αλλάξουμε την τιμή μιας καθολικής μεταβλητής, δηλαδή μιας μεταβλητής η οποία ορίζεται στο κορυφαίο επίπεδο του προγράμματος (δηλαδή όχι μέσα σε κάποιου είδους εμβέλεια, όπως σε συναρτήσεις ή κλάσεις), τότε πρέπει να δηλώσουμε στην Python ότι η μεταβλητή αυτή δεν είναι τοπική αλλά καθολική. Αυτό γίνεται με τη χρήση της εντολής **global**, με την οποία γίνεται ξεκάθαρο ότι η μεταβλητή βρίσκεται σε ένα εξωτερικό τμήμα εντολών.

Ας πειραματιστούμε με τα ακόλουθα:

Παράδειγμα

```
x = 50
```

```
def func():
```

```
    global x
```

```
    print('Το x είναι', x)
```

```
    x = 2
```

```
    print('Το καθολικό x άλλαξε σε', x)
```

```
func()
```

```
print('Η τιμή του x είναι', x)
```

Έξοδος:

Το x είναι 50

Το καθολικό x άλλαξε σε 2

Η τιμή του x είναι 2

Παράδειγμα επίδειξης χειρισμού της Python των μεταβλητών τοπικής και καθολικής εμβέλειας.

Έστω ότι έχουμε το εξής σενάριο:

```
myGlobal = 5
```

```
def func1():
```

```
    myGlobal = 42
```

```
def func2():
```

```
    print myGlobal
```

```
func1()
```

```
func2()
```

Τυπώνει, όχι την τιμή 42, αλλά το 5. Αν βέβαια προσθέσουμε μία δήλωση 'global' στη func1(), όπως στο παρακάτω παράδειγμα, τότε η func2() θα τυπώσει 42.

```
def func1():
```

```
    global myGlobal
```

```
    myGlobal = 42
```

Αυτό που συμβαίνει, είναι ότι η Python θεωρεί ότι αν μέσα σε μια συνάρτηση εκχωρηθεί σε οποιαδήποτε μεταβλητή μια τιμή, η μεταβλητή αυτή είναι τοπική και η τιμή της ισχύει μόνο για αυτή τη συνάρτηση, εκτός αν δηλωθεί ρητά διαφορετικά. Αν απλώς διαβάσει μια τιμή από τη μεταβλητή και αυτή δεν υπάρχει τοπικά, τότε προσπαθεί να την αναζητήσει σε οποιαδήποτε δυνατή εμβέλεια, για παράδειγμα καθολική. Όταν εκχωρούμε το 42 στη myGlobal, η Python δημιουργεί μια τοπική μεταβλητή, η οποία υπερκαλύπτει την καθολική μεταβλητή που έχει την ίδια ονομασία. Αυτή η τοπική μεταβλητή βγαίνει εκτός εμβέλειας και χάνεται, όταν η func1() ολοκληρώνεται. Η func2() δε μπορεί ποτέ να δει τι έγινε μέσα σε άλλη συνάρτηση και βλέπει μόνο την καθολική μεταβλητή (η οποία δεν μεταβλήθηκε).

7.3 Αρθρώματα (Modules)

7.3.1 Εισαγωγή

Έχουμε δει πώς μπορούμε να επαναχρησιμοποιήσουμε κώδικα στο πρόγραμμά μας, ορίζοντας συναρτήσεις. Τι κάνουμε όμως, αν θέλουμε να επαναχρησιμοποιήσουμε έναν αριθμό συναρτήσεων σε άλλα προγράμματα που γράφουμε; Η λύση είναι τα **αρθρώματα (module)**. Ένα άρθρωμα μας επιτρέπει να οργανώσουμε με λογικό τρόπο έναν κώδικα Python. Η ομαδοποίηση σχετικού κώδικα σε ένα module κάνει τον κώδικα ευκολότερο στην κατανόηση και χρήση. Ένα άρθρωμα είναι ένα Python αντικείμενο, όπως όλα στη γλώσσα αυτή. Με απλά λόγια,

ένα άρθρωμα είναι ένα αρχείο αποτελούμενο από κώδικα Python και μπορεί να ορίσει συναρτήσεις, κλάσεις και μεταβλητές.

Υπάρχουν διάφορες μέθοδοι για να γράφουμε αρθρώματα, αλλά ο απλούστερος τρόπος είναι δημιουργώντας ένα αρχείο με επέκταση **.py**, το οποίο θα περιέχει συναρτήσεις και μεταβλητές.

Για να χρησιμοποιήσουμε ένα άρθρωμα στο πρόγραμμά μας, θα πρέπει να το "εισάγουμε" σε αυτό. Αυτό γίνεται με τη δήλωση **import**.

Η δήλωση **import**

Η **import** έχει την ακόλουθη σύνταξη:

```
import module1[, module2[,... moduleN]]
```

Η **from...import**

Η **from...import** δίνει τη δυνατότητα εισαγωγής συγκεκριμένων χαρακτηριστικών - δυνατοτήτων του αρθρώματος, χωρίς να εισάγει όλο το άρθρωμα και έχει την ακόλουθη σύνταξη:

```
from modname import name1[, name2[, ... nameN]]
```

Η δήλωση **from...import ***

Είναι δυνατό να εισάγουμε όλα τα περιεχόμενα από ένα άρθρωμα, με χρήση της ακόλουθης δήλωσης **import**:

```
from modname import *
```

Όμως δεν ενδείκνυται η συχνή χρήση του.

Για να έχουμε πρόσβαση σε μια από τις συναρτήσεις, θα πρέπει να προσδιορίσουμε το όνομα της μονάδας και το όνομα της συνάρτησης, χωρισμένα με μία τελεία. Αυτή η μορφή ονομάζεται **συμβολισμός με τελεία (dot notation)**.

Σε ένα πρόγραμμα μπορούν να συνυπάρχουν τρεις κατηγορίες συναρτήσεων.

Μπορούν να υπάρχουν συναρτήσεις *ενσωματωμένες* στο περιβάλλον (built-in) που είναι πάντα διαθέσιμες για χρήση, συναρτήσεις που περιέχονται σε *εξωτερικά αρθρώματα*, τα οποία

πρέπει πρώτα να εισαχθούν και τέλος συναρτήσεις που **ορίζονται από τον προγραμματιστή** (με το `def`).

Για παράδειγμα στο παρακάτω πρόγραμμα εκτελούνται ορισμένες μαθηματικές πράξεις:

```
from math import sqrt
```

```
def cube(x):  
    return x * x * x
```

```
print abs(-1)  
print cube(9)  
print sqrt(81)
```

Για τις ακόλουθες δηλώσεις παρατηρούμε τα:

```
from math import sqrt
```

Η `sqrt()` συνάρτηση εισάγεται από το `math` module.

```
def cube(x):  
    return x * x * x
```

Η `cube()` συνάρτηση είναι μια συνάρτηση οριζόμενη από τον προγραμματιστή.

```
print abs(-1)
```

Η `abs()` συνάρτηση είναι μία ενσωματωμένη built-in συνάρτηση, προσβάσιμη καθολικά και αποτελεί μέρος του πυρήνα/core της γλώσσας.

7.3.2 Σύντομη περιγραφή της Πρότυπης βιβλιοθήκης (Standard Library)

Μια βιβλιοθήκη (library), σε οποιαδήποτε γλώσσα προγραμματισμού, είναι μια συλλογή εργαλείων που μπορεί να έχουν γραφτεί και από άλλους προγραμματιστές, προκειμένου να εκτελούνται συγκεκριμένες λειτουργίες. Οι βιβλιοθήκες είναι πολύ σημαντικές στον προγραμματισμό, γιατί μας δίνουν τη δυνατότητα να χρησιμοποιούμε τα εργαλεία που περιλαμβάνονται σε αυτές.

Η *πρότυπη βιβλιοθήκη* της Python περιέχει έναν τεράστιο αριθμό χρήσιμων αρθρωμάτων και είναι μέρος κάθε πρότυπης εγκατάστασης Python. Είναι σημαντικό να εξοικειωθούμε με την πρότυπη βιβλιοθήκη, επειδή πολλά προβλήματα μπορούν να λυθούν γρήγορα, αν αξιοποιηθεί το εύρος των δυνατοτήτων που έχουν οι βιβλιοθήκες. Περιλαμβάνει τμήματα για προγραμματισμό γραφικών (Tkinter), αριθμητική επεξεργασία, web συνδεσιμότητα, βάσεις δεδομένων (Anydbm), Βιοπληροφορική (Biopython) κ.ά. Επίσης, βιβλιοθήκες από πολλές άλλες γλώσσες προγραμματισμού, μπορούν να χρησιμοποιηθούν στην Python.

Η Python έχει μια μαθηματική μονάδα λογισμικού (**math module**), η οποία περιέχει τις δημοφιλέστερες μαθηματικές συναρτήσεις. Προτού χρησιμοποιήσουμε μια μονάδα, άρα και τη *math*, πρέπει να την εισάγουμε.

Παράδειγμα 1. Μαθηματική μονάδα λογισμικού

A)

```
>>> import math
>>> print math
<module 'math' (built-in)>
>>> print math.pi
3.14159265359
>>> print math.cos(math.pi / 4.0)
0.70710678118654757
```

Αν εισάγουμε τη *math*, θα πάρουμε ένα αντικείμενο το οποίο περιέχει σταθερές, όπως η *pi* και συναρτήσεις όπως η *sin* και η *exp*. Αλλά, αν προσπαθήσουμε να αποκτήσουμε απευθείας πρόσβαση στην *pi*, θα πάρουμε ένα μήνυμα λάθους.

B) Εναλλακτικά, μπορούμε να εισάγουμε ένα αντικείμενο από μια μονάδα ως εξής:

```
>>> from math import pi
```

Τώρα μπορούμε να έχουμε απευθείας πρόσβαση στην `pi`, χωρίς το συμβολισμό τελείας.

```
>>> print pi
```

```
3.14159265359
```

Επίσης μπορούμε να χρησιμοποιήσουμε τον **τελεστή αστεράκι** για να τα εισάγουμε όλα από τη μονάδα:

```
>>> from math import *
```

```
>>> cos(pi)
```

```
-1.0
```

Παράδειγμα 2. Μονάδα Λογισμικού `random`

Το **`random module`** διαθέτει εργαλεία δημιουργίας ψευδοτυχαίων αριθμών:

```
>>> import random
```

```
>>> random.choice(['apple', 'pear', 'banana'])
```

```
'apple'
```

```
>>> random.sample(xrange(100), 10) # τυχαία δειγματοληψία  
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
```

```
>>> random.random() # τυχαίος κινητής υποδιαστολής (float)  
0.17970987693706186
```

```
>>> random.randrange(6) # τυχαίος ακέραιος στο διάστημα  
range(6)
```

```
5
```

7.3.3 Πακέτα (Packages)

Μέχρι τώρα, πρέπει να έχουμε αρχίσει να παρατηρούμε την ιεραρχία με την οποία οργανώνονται τα προγράμματά μας. Οι μεταβλητές, συνήθως, πηγαίνουν μέσα στις συναρτήσεις, ενώ οι

συναρτήσεις και οι καθολικές μεταβλητές πηγαίνουν συχνά μέσα στα αρθρώματα. Τι θα γινόταν όμως, αν θέλαμε να οργανώσουμε τα αρθρώματα;

Τα πακέτα είναι ένα εργαλείο για την ιεραρχική οργάνωση των αρθρωμάτων. Η Python παρέχει ένα απλό σύστημα δημιουργίας πακέτων, ως επέκταση του μηχανισμού των αρθρωμάτων. Κάθε κατάλογος με ένα `__init__.py` αρχείο αναφέρεται ως ένα Python πακέτο.

Δραστηριότητες - Ερωτήσεις - Ασκήσεις κεφαλαίου

7.4 Δραστηριότητες

Δραστηριότητα 1

Να εκτελεστούν και να κατανοηθούν τα παρακάτω παραδείγματα:

Παράδειγμα 1

```
name = "black"
```

```
def f():  
    name = "white"  
    print "Μέσα στη συνάρτηση", name
```

```
print "Έξω από τη συνάρτηση", name  
f()
```

Έξοδος:

Έξω από τη συνάρτηση black

Μέσα στη συνάρτηση white

Εξήγηση: Μία μεταβλητή που ορίζεται στο σώμα της συνάρτησης έχει τοπική εμβέλεια, δηλαδή έχει ισχύ μόνο μέσα στο σώμα της συνάρτησης.

Παράδειγμα 2

```
name = "black"
```

```
def f():  
    print "Μέσα στη συνάρτηση", name
```

```
print "Εξω από τη συνάρτηση", name  
f()
```

Έξοδος:

Έξω από τη συνάρτηση black

Μέσα στη συνάρτηση black

Μπορούμε να πάρουμε τα περιεχόμενα μιας καθολικής global μεταβλητής μέσα στο σώμα μιας συνάρτησης. Αλλά αν θέλουμε να αλλάξουμε μία καθολική μεταβλητή σε μία συνάρτηση, πρέπει να χρησιμοποιήσουμε τη δήλωση global.

Παράδειγμα 3

```
name = "black"
```

```
def f():  
    global name  
    name = "white"  
    print "Μέσα στη συνάρτηση", name
```

```
print "Εξω από τη συνάρτηση", name  
f()
```

```
print "Εξω από τη συνάρτηση", name
```

Το παράδειγμα αυτό αλλάζει το περιεχόμενο της global μεταβλητής name μέσα στη συνάρτηση f().

Με τις εντολές:

```
global name  
name = "white"
```

Με χρήση της `global`, αναφερόμαστε στη μεταβλητή `name` που ορίζεται έξω από το σώμα της συνάρτησης. Ακολουθώς στη μεταβλητή αυτή δίνεται μια νέα τιμή.

Έξοδος:

Έξω από τη συνάρτηση `black`

Μέσα στη συνάρτηση `white`

Έξω από τη συνάρτηση `white`

7.5 Ερωτήσεις

1. Ποια είναι η διαφορά παραμέτρων και απλών μεταβλητών;
2. Τι ονομάζεται εμβέλεια μεταβλητών;
3. Τι χαρακτηρίζει τις καθολικές μεταβλητές;
4. Ποια η διαφορά της περιορισμένης από την απεριόριστη εμβέλεια;
5. Ποια είναι η λειτουργία της εντολής `global`;
6. Τι γνωρίζετε για τα αρθρώματα;
7. Ποια είναι η χρησιμότητα της δήλωσης `import`;
8. Τι γνωρίζετε για τα πακέτα της Python;
9. Ποια είναι η χρησιμότητα των βιβλιοθηκών;
10. Τι ορίζει η εντολή `def`;

Σύνοψη

Όπως οι συναρτήσεις είναι επαναχρησιμοποιούμενα μέρη προγραμμάτων, τα αρθρώματα είναι επαναχρησιμοποιούμενα προγράμματα. Τα πακέτα είναι μια άλλη ιεραρχία, για να οργανώνουμε αρθρώματα. Η πρότυπη βιβλιοθήκη που συνοδεύει την Python είναι ένα παράδειγμα τέτοιων πακέτων και αρθρωμάτων.

Ένα άρθρωμα (`module`) είναι μια συλλογή σχετικών συναρτήσεων και αποθηκεύεται σε αρχείο με κατάληξη `.py`. Μπορούμε, να γράψουμε και τα δικά μας αρθρώματα, αποθηκεύοντας τις συναρτήσεις μας σε ένα αρχείο `.py`. Για να χρησιμοποιήσουμε ένα

άρθρωμα σε ένα πρόγραμμα, θα πρέπει να το εισάγουμε με την εντολή `import`.

Εκτός από τις ενσωματωμένες βιβλιοθήκες (μονάδες) συναρτήσεων που περιλαμβάνονται στη γλώσσα Python, μπορούμε να βρούμε στους δικτυακούς τόπους υποστήριξης της γλώσσας και εξωτερικές μονάδες λογισμικού με πληθώρα επιπλέον συναρτήσεων για τη δημιουργία ισχυρών προγραμμάτων.

Ένα από τα καθοριστικά πλεονεκτήματα της γλώσσας Python, είναι ότι υποστηρίζεται από μια παγκόσμια κοινότητα προγραμματιστών που συνεισφέρουν με τη δημιουργία πολλών εξωτερικών βιβλιοθηκών. Τις βιβλιοθήκες αυτές μπορούμε να χρησιμοποιήσουμε, για να δημιουργήσουμε γρήγορα εντυπωσιακές και φιλικές προς το χρήστη εφαρμογές. Τις συλλογές αυτές μπορούμε να τις θεωρήσουμε ως πρόσθετες εργαλειοθήκες με έτοιμα εργαλεία, τα οποία μπορούμε να χρησιμοποιήσουμε μέσα στον κώδικά μας, αφού οι περισσότερες εργαλειοθήκες της Python ανήκουν στην κατηγορία του ελεύθερου λογισμικού.

8

Δομές Δεδομένων II

ΕΣΩΤΕΡΙΚΗ ΕΓΚΕΙΡΑ: έκδοση v201

8. Δομές Δεδομένων II

Εισαγωγή

Μια **δομή δεδομένων** μπορεί να οριστεί ως ένα σχήμα οργάνωσης σχετικών μεταξύ τους στοιχείων δεδομένων. Η επιλογή της κατάλληλης δομής δεδομένων παίζει σημαντικό ρόλο στην ανάπτυξη του αλγορίθμου, για την επίλυση ενός προβλήματος.

Στο κεφάλαιο αυτό θα αναπτυχθούν οι *ενσωματωμένες* δομές της Python, όπως τα αλφαριθμητικά, οι λίστες, οι πλειάδες και τα λεξικά. Σχετικά με τις λίστες και τα αλφαριθμητικά, που είχαν καλυφθεί και στην ύλη της Β' Λυκείου, θα γίνει μια πιο εκτενής παρουσίαση των βασικών λειτουργιών τους, μέσα από διάφορα παραδείγματα. Επίσης, με χρήση των ενσωματωμένων δομών της Python θα παρουσιαστούν υλοποιήσεις σύνθετων δομών δεδομένων, όπως είναι η στοίβα και η ουρά. Τέλος θα παρουσιαστούν θεωρητικά, οι δομές των γράφων και των δέντρων.

Διδακτικοί στόχοι

Μετά τη μελέτη του κεφαλαίου, θα μπορούμε να:

- αναλύουμε τρόπους αναπαράστασης στη μνήμη βασικών στατικών και δυναμικών δομών
- αναφέρουμε τα ιδιαίτερα χαρακτηριστικά των δομών: πίνακα, λίστας, στοίβας και ουράς
- κατονομάζουμε τη χρησιμότητα των δομών: δέντρα και γράφοι
- επιλέγουμε και χρησιμοποιούμε κατάλληλες δομές στα προγράμματα που υλοποιούμε.

Λέξεις κλειδιά

Δομές δεδομένων, λίστα, στοίβα, ουρά, λεξικό, γράφος.

Διδακτικές Ενότητες

8.1 Συμβολοσειρές (strings)

Τα **αλφαριθμητικά** ή **συμβολοσειρές** στην Python είναι ακολουθίες από χαρακτήρες που έχουν σταθερό μέγεθος και *μη μεταβαλλόμενα* περιεχόμενα. Δηλαδή, δεν μπορούμε να

προσθέσουμε ή να αφαιρέσουμε χαρακτήρες, ούτε να τροποποιήσουμε τα περιεχόμενα του αλφαριθμητικού. Γι' αυτό λέμε ότι η δομή αυτή ανήκει στις *μη μεταβαλλόμενες* (immutable) δομές της Python. Η αρίθμηση των χαρακτήρων σε ένα αλφαριθμητικό ξεκινάει από το 0. Για παράδειγμα: αν **word = "PYTHON"**, η αναπαράσταση του αλφαριθμητικού μοιάζει με το παρακάτω σχήμα:

0	1	2	3	4	5
'P'	'Y'	'T'	'H'	'O'	'N'

Για παράδειγμα: το στοιχείο word[3] αναφέρεται στο 4ο γράμμα ('H'), ενώ το στοιχείο word[0], στο 1ο γράμμα ('P') . Ο τύπος των αλφαριθμητικών στην Python ονομάζεται **str**, από τα αρχικά της λέξης string. Παρακάτω δίνονται μερικά παραδείγματα εντολών με αλφαριθμητικά στο διερμηνευτή της Python, όπου φαίνεται ότι:

<pre>>>> word = "PYTHON" >>> len(word) 6 >>> print word[5]+word[0] NP</pre>	<pre>>>> str(28) == '28' True >>> print int('496') + 4 500</pre>
--	--

- η συνάρτηση len επιστρέφει το μήκος, δηλαδή το πλήθος των χαρακτήρων του αλφαριθμητικού
- ο τελεστής + όταν εφαρμόζεται σε αντικείμενα τύπου string, έχει σαν αποτέλεσμα τη συνένωσή τους σε μια συμβολοσειρά
- η συνάρτηση str μετατρέπει μια τιμή σε συμβολοσειρά
- με τη συνάρτηση int μπορούμε να μετατρέψουμε ένα αλφαριθμητικό στον ακέραιο αριθμό που αναπαριστά.

Έλεγχος ύπαρξης

Ένας σημαντικός τελεστής που θα συναντήσουμε και αργότερα στις λίστες, είναι ο *υπαρξιακός τελεστής in*, ο οποίος ελέγχει, αν ένα αντικείμενο ανήκει σε ένα σύνολο αντικειμένων. Δεδομένου ότι οι συμβολοσειρές μπορούν να θεωρηθούν ως σύνολα

χαρακτήρων, μπορούμε να τον χρησιμοποιήσουμε όπως φαίνεται παρακάτω.

<pre>>>> "Py" in "Python" True >>> "a" in "Python" False >>> "a" not in "Python" True</pre>	<pre>>>> 'antonis' > 'antonia' True >>> '1000' < '2' True >>> 'babylon5' > 'babylon4' True</pre>
--	--

Επίσης, οι γνωστοί *συγκριτικοί τελεστές* (<, <=, >, >=, =) ισχύουν και στις συμβολοσειρές, η λειτουργία των οποίων βασίζεται στη λεξικογραφική διάταξη των χαρακτήρων.

Παράδειγμα 1: Σάρωση των χαρακτήρων μιας συμβολοσειράς

Μια συμβολοσειρά είναι μια ακολουθία (sequence) από αντικείμενα, τα οποία, στην προκειμένη περίπτωση, είναι χαρακτήρες. Αν θέλουμε να σαρώσουμε τα αντικείμενα αυτά ένα-ένα, μπορούμε να το κάνουμε με μια εντολή επανάληψης for. Το παρακάτω τμήμα κώδικα δημιουργεί μια νέα συμβολοσειρά η οποία είναι όμοια με την αρχική, αλλά χωρίς τα κενά ανάμεσα στις λέξεις:

```
def trimSpaces( sentence ):
    result = ""
    for char in sentence :
        if char != " " :
            result += char
    return result

>>> phrase = "Houston we have a problem"
>>> trimSpaces( phrase )
'Houstonwehaveaproblem'
```

Παράδειγμα 2: Καταμέτρηση φωνηέντων μιας φράσης

Η παρακάτω συνάρτηση υπολογίζει και επιστρέφει το πλήθος των φωνηέντων της λέξης που δέχεται ως παράμετρο.

```
def count_vowels( word ):
    vowels = "ΑΕΙΟΥαεiou"
    count = 0
    for letter in word :
        if letter in vowels:
            count += 1
    return count
```

Σημείωση: Παρατηρήστε το διαφορετικό τρόπο με τον οποίο χρησιμοποιείται ο τελεστής **in**, στη μια περίπτωση για τον καθορισμό του εύρους της επανάληψης και στην άλλη για τον έλεγχο ύπαρξης του γράμματος **letter** στη λέξη **vowels**.

Παραπάνω κάνουμε χρήση του ιδιώματος **for...in** για να επεξεργαστούμε τους χαρακτήρες της συμβολοσειράς, έναν κάθε φορά.

8.2 Λίστες

Η **λίστα** είναι μια διατεταγμένη ακολουθία αντικειμένων, όχι απαραίτητα του ίδιου τύπου και αποτελεί τη **βασική δομή δεδομένων της Python**. Η λίστα, σε αντίθεση με τη συμβολοσειρά, είναι μια **δυναμική δομή** στην οποία μπορούμε να προσθέτουμε ή να αφαιρούμε στοιχεία (**mutable**). Κάθε αντικείμενο της λίστας χαρακτηρίζεται από ένα μοναδικό αύξοντα αριθμό, ο οποίος ορίζει τη θέση του στη λίστα, ενώ η προσπέλαση στα στοιχεία της λίστας γίνεται όπως στις συμβολοσειρές, με το όνομα της λίστας και τον αύξοντα αριθμό του αντικείμενου μέσα σε αγκύλες.

Η εντολή $L = [3, 5, 8, 13, 21, 34]$ δημιουργεί τη μεταβλητή L που αναφέρεται στη λίστα $[3, 5, 8, 13, 21, 34]$, όπως φαίνεται στην εικόνα:

	0	1	2	3	4	5
L	3	5	8	13	21	34

Η αρίθμηση των στοιχείων, όπως στις συμβολοσειρές, έτσι και στις λίστες ξεκινάει από το 0. Άρα το 1ο στοιχείο της λίστας είναι το $L[0]$, το οποίο είναι ίσο με το 3, το 2ο το $L[1]$ και το τελευταίο το $L[5]$.

```
>>> L = [3, 5, 8, 13, 21, 34]
>>> print L[0]
3
>>> print L[5]
34
```

Μπορεί κανείς ανά πάσα στιγμή να προσθέσει, να αφαιρέσει ή να τροποποιήσει οποιοδήποτε στοιχείο της λίστας.

```
>>> daysofweek =
["Δευτέρα", "Τρίτη", "Τετάρτη", "Πέμπτη", "Παρασκευή"]
>>> print daysofweek[0] + daysofweek[4]
ΔευτέραΠαρασκευή
>>> daysofweek = daysofweek + ["Σάββατο"]
>>> daysofweek[0] = daysofweek[1] = "Κυριακή"
>>> print daysofweek
["Κυριακή", "Κυριακή", "Τετάρτη", "Πέμπτη", "Παρασκευή",
"Σάββατο"]
```

Έτσι, αν θέλουμε να προσθέσουμε ένα στοιχείο στο τέλος μιας λίστας, γράφουμε:

Λίστα = Λίστα + [στοιχείο]

ενώ στην αρχή της λίστας

Λίστα = [στοιχείο] + Λίστα

Οι λίστες στην Python:

- Δεν έχουν σταθερό μέγεθος, δηλαδή μπορούν να αυξάνονται και να μειώνονται κατά την εκτέλεση του προγράμματος.
- Η αρίθμηση των δεικτών ξεκινάει από το 0, όπως ακριβώς στις συμβολοσειρές.
- Είναι *δυναμικές δομές*, και χαρακτηρίζονται από μεγάλη ευελιξία. Έτσι για παράδειγμα, μπορούμε να έχουμε σε μια λίστα ακόμα και στοιχεία διαφορετικού τύπου.

```
>>> mix = [6, 3.14159, True, "Guido Van Rossum"]
>>> len(mix)
4
```

Στις λίστες μπορούμε να χρησιμοποιήσουμε τον υπαρξιακό τελεστή **in**, τη συνάρτηση **len** ή και τον τελεστή συνένωσης '+', ακριβώς όπως στις συμβολοσειρές.

<pre>>>> fruits = ['apple', 'orange'] >>> len(fruits) 2 >>> print fruits[0] apple >>> 'apple' in fruits True >>> powers = [2, 4, 8, 16]</pre>	<pre>>>> fib = [3, 5, 8, 13, 21] >>> fib + powers [3, 5, 8, 13, 21, 2, 4, 8, 16] >>> powers + fruits [2, 4, 8, 16, 'apple', 'orange'] >>> fib = fib + [fib[3] + fib[4]] >>> print fib [3, 5, 8, 13, 21, 34]</pre>
--	--

Επειδή οι λίστες και οι συμβολοσειρές ανήκουν και οι δύο σε μια πιο γενική κατηγορία δομών, τις ακολουθιακές δομές (sequences) της Python, ισχύουν οι ίδιοι τελεστές:

item in List: επιστρέφει True, αν το στοιχείο item υπάρχει μέσα στη λίστα List, αλλιώς επιστρέφει False.

item not in List: επιστρέφει True, αν το στοιχείο item δεν υπάρχει μέσα στη λίστα List, αλλιώς, αν υπάρχει επιστρέφει False.

Εκτός από τους τελεστές που αναφέραμε παραπάνω, υπάρχουν και συναρτήσεις που παίρνουν ως όρισμα μια λίστα. Οι συναρτήσεις των λιστών που θα χρησιμοποιήσουμε σε αυτό το κεφάλαιο είναι η len και η list.

len (List): Επιστρέφει το πλήθος των στοιχείων (ή μέγεθος) της λίστας.

list (String): Επιστρέφει μια λίστα με στοιχεία τους χαρακτήρες της συμβολοσειράς string. Η συνάρτηση αυτή μπορεί να μετατρέψει και άλλα είδη δομών σε λίστα, όπως είναι οι πλειάδες και τα λεξικά.

Οι λίστες, όπως και οι συμβολοσειρές, διαθέτουν μεγάλη ποικιλία μεθόδων, η χρήση των οποίων μπορεί να επεκτείνει, σε μεγάλο βαθμό, τη λειτουργικότητά τους. Στο κεφάλαιο αυτό θα χρησιμοποιήσουμε μόνο τις παρακάτω μεθόδους των λιστών, όπου L το όνομα της λίστας:

L.append(object): προσθήκη του στοιχείου object στο τέλος της λίστας L.

L.insert(index, object): προσθήκη του στοιχείου object, στη θέση index της λίστας L, μετακινώντας όλα τα στοιχεία από τη θέση index και μετά, κατά μια θέση.

L.pop([index]): Αφαίρεση από τη λίστα του στοιχείου που βρίσκεται στη θέση index. Αν δεν δοθεί θέση, τότε θα αφαιρεθεί το τελευταίο στοιχείο της λίστας.

Παρακάτω δίνονται μερικά παραδείγματα κλήσεων των μεθόδων που θα χρησιμοποιήσουμε και δίπλα, μέσα σε σχόλια, η νέα μορφή της λίστας.

```
>>> fib = [5, 8, 13, 21, 34]
>>> fib.pop(1)           # fib = [5, 13, 21, 34]
>>> fib.append(55)       # fib = [5, 13, 21, 34, 55]
>>> fib.pop()            # fib = [5, 13, 21, 34]
>>> fib.insert(2, 89)     # fib = [5, 13, 89, 21, 34]
```

Αν και η Python ορίζει μια μεγάλη ποικιλία μεθόδων για την επεξεργασία και διαχείριση λιστών, στο βιβλίο αυτό θα χρησιμοποιηθούν μόνο οι παραπάνω, για την επίλυση προβλημάτων.

Διάσχιση Λίστας

Μπορούμε να επεξεργαστούμε τα στοιχεία μιας λίστας, ένα κάθε φορά, κάνοντας χρήση του παρακάτω ιδιώματος της δομής επανάληψης for:

```
for item in List :
    <Εντολές Επεξεργασίας του αντικειμένου item>
```

Παράδειγμα 1: Δημιουργία και εμφάνιση στοιχείων λίστας

Οι παρακάτω εντολές αρχικά κατασκευάζουν μια λίστα η οποία περιέχει όλους τους αριθμούς από το 1 έως και το 6 και στη συνέχεια εμφανίζουν κάθε αριθμό σε διαφορετική γραμμή.

```
L = [1, 2, 3, 4, 5, 6]
```

```
for number in L :
```

```
    print number
```

Παράδειγμα 2: Μέσος όρος των στοιχείων μιας λίστας

Για να υπολογίσουμε το μέσο όρο των στοιχείων μιας λίστας, πρώτα χρειάζεται να υπολογίσουμε το άθροισμα των στοιχείων,

χρησιμοποιώντας μια μεταβλητή στην οποία προσθέτουμε, κάθε φορά, το επόμενο στοιχείο της λίστας:

```
sum = 0.0          # το sum είναι πραγματικός (float)
for number in L :
    sum = sum + number
average = sum / len( L )  # δεν θα γίνει ακέραια διαίρεση
print average
```

Παράδειγμα 3: Μέγιστη τιμή σε μια λίστα

```
maximum = L[0]
for number in L :
    if number > maximum :
        maximum = number
print maximum
```

Παράδειγμα 4: Ρέστα

Καλούμαστε να σχεδιάσουμε το λογισμικό μιας ταμειακής μηχανής, το οποίο θα διαβάζει το ποσό που έδωσε ο πελάτης και το κόστος των αγορών του και θα εμφανίζει το πλήθος των κερμάτων ή χαρτονομισμάτων που θα δοθούν για ρέστα. Θεωρήστε ότι όλες οι τιμές είναι σε ακέραια πολλαπλάσια του ευρώ:

```
values = [100, 50, 20, 10, 5, 2, 1]
cost = input( "Δώσε το κόστος των αγορών" )
payment = input( "Δώσε το ποσό της πληρωμής" )
change = payment - cost
counter = 0
for value in values :
    counter = counter + ( change / value )
    change = change % value
print counter
```

Εφαρμογή: Διαχωρισμός λίστας

Η λειτουργία του διαχωρισμού μιας λίστας σε δύο μέρη με βάση κάποια κριτήρια, αποτελεί μια τυπική επεξεργασία των λιστών.

```
positives = negatives = [ ]
for number in numbers :      # για κάθε αριθμό της λίστας
    if number > 0 :           # αν είναι θετικός
        positives = positives + [ number ]
        # πρόσθεσέ τον στη λίστα με τους θετικούς
    else :
        negatives = negatives + [ number ]
        # αλλιώς στη λίστα με τους αρνητικούς
print positives
print negatives
```

Το παραπάνω πρόγραμμα διαχωρίζει τους αριθμούς μιας λίστας σε αρνητικούς και θετικούς. Υποθέτουμε ότι όλοι οι αριθμοί είναι διάφοροι του μηδενός. Να σημειωθεί ότι η αρχική λίστα παραμένει ανέπαφη.

Εφαρμογή: Συγχώνευση διατεταγμένων λιστών

Ένας από τους πιο γνωστούς και χρήσιμους αλγόριθμους της Πληροφορικής είναι ο αλγόριθμος της συγχώνευσης των στοιχείων δυο ταξινομημένων λιστών σε μία νέα, επίσης ταξινομημένη, λίστα. Ο αλγόριθμος αξιοποιεί το γεγονός ότι οι αρχικές λίστες είναι ήδη ταξινομημένες, ώστε να μη χρειαστεί να ταξινομήσει από την αρχή την τελική λίστα, κάτι το οποίο έχει σημαντικό υπολογιστικό κόστος για πολύ μεγάλες λίστες.

```
def merge( A, B ) :
    L = [ ]
    while A != [ ] and B != [ ] : # όσο και οι δυο λίστες έχουν στοιχεία
        if A[0] < B[0] :          # Αν το πρώτο στοιχείο της A είναι το μικρότερο
            L = L + [ A.pop(0) ]  # το μεταφέρουμε στην L
```

```
else :
```

```
    L = L + [ B.pop(0) ]
```

```
    # αλλιώς μεταφέρουμε το πρώτο στοιχείο της B στην L
```

```
return L + A + B
```

```
# στο τέλος προσθέτουμε τα στοιχεία που έχουν μείνει
```

Αφού οι δύο λίστες είναι ταξινομημένες σε αύξουσα σειρά το ελάχιστο στοιχείο και των δύο λιστών είναι το μικρότερο από τα $A[0]$, $B[0]$. Στη συνέχεια μεταφέρουμε αυτό το στοιχείο στην τελική λίστα.

Κάποια στιγμή, μία από τις δυο λίστες θα αδειάσει, οπότε η επανάληψη θα σταματήσει. Η άλλη λίστα όμως θα έχει κάποια στοιχεία τα οποία πρέπει να προστεθούν στο τέλος της τρίτης λίστας. Έτσι θα έπρεπε να γράψουμε:

```
if A == [] :
    L = L + B
else :
    L = L + A
```

} L = L + B + A

Οπότε αν η A είναι κενή το αποτέλεσμα είναι $L + B + []$, ενώ αν η B είναι κενή το αποτέλεσμα είναι $L + [] + A$.

Η συνάρτηση range (έχει αναλυθεί και στην 4.1.3)

Η συνάρτηση range επιστρέφει, αν δώσουμε την αρχική, την τελική τιμή και το βήμα, μια λίστα από αριθμούς, όπως φαίνεται παρακάτω:

<pre>>>> range(4) [0, 1, 2, 3] >>> range(0, 4) [0, 1, 2, 3] >>> range(0, 4, 1) [0, 1, 2, 3]</pre>	<pre>>>> range(10, 30, 5) [10, 15, 20, 25] >>> range(30, 10, -5) [30, 25, 20, 15] >>> range(1, 2, 100) [1]</pre>
--	---

```
>>> range( 1, 1, 100 )
[]
>>> range( 1, 5, -1 )
[]
>>> range( 0 )
[]
```

Η συνάρτηση **range**(A, M, B) επιστρέφει μια λίστα αριθμών ξεκινώντας με τον αριθμό A μέχρι το M με βήμα B. Το M δεν συμπεριλαμβάνεται στη λίστα.

Έτσι τα παρακάτω τμήματα κώδικα εκτελούν την ίδια λειτουργία:

```
>>> L = [ 6, 28, 496, 8128 ]
>>> for item in L :
    print item ,
6 28 496 8128
```

```
>>> L = [ 6, 28, 496, 8128 ]
>>> for index in range(0,4) :
    print L[index] ,
6 28 496 8128
```

```
>>> L = [ 6, 28, 496, 8128 ]
>>> for index in [ 0, 1, 2, 3 ] :
    print L[index] ,
6 28 496 8128
```

Η range μας διευκολύνει επίσης στην εκτέλεση ενός τμήματος εντολών για έναν προκαθορισμένο αριθμό επαναλήψεων, όπως φαίνεται παρακάτω:

```
>>> sum = 0
>>> for i in range(100) :
    sum = sum + i
>>> print sum
5050
```

```
>>> for index in range(2,11,2) :
    print index ,
2 4 6 8 10
```

8.3 Στοιβά

Εισαγωγική Δραστηριότητα

Όλοι χρησιμοποιούμε διάφορα προγράμματα για πλοήγηση στον παγκόσμιο ιστό, τα οποία είναι γνωστά ως φυλλομετρητές (browsers) και πολλές φορές, κατά την πλοήγηση, θέλουμε να επιστρέψουμε στην αμέσως προηγούμενη ιστοσελίδα στην οποία βρισκόμασταν. Γι' αυτό υπάρχει η λειτουργία **Πίσω** (Back) σε όλους τους φυλλομετρητές, η οποία μας μεταφέρει στην προηγούμενη ιστοσελίδα. Για να επιτευχθεί αυτή η λειτουργία, το περιβάλλον διατηρεί ένα ιστορικό των ιστοσελίδων τις οποίες επισκεφθήκαμε. Η δομή δεδομένων που χρησιμοποιείται για την αποθήκευση του ιστορικού, πρέπει να είναι τέτοια, ώστε οι ιστοσελίδες να ανακτώνται με την αντίστροφη σειρά επίσκεψης. Δηλαδή στην πρώτη θέση θέλουμε να είναι η ιστοσελίδα που επισκεφτήκαμε πιο πρόσφατα και στην τελευταία η αρχική ιστοσελίδα από την οποία ξεκινήσαμε.

Ας υποθέσουμε για παράδειγμα ότι, ξεκινώντας από την ιστοσελίδα <http://www.python.org> μεταβαίνουμε στην ιστοσελίδα <http://www.pythontutor.com> και στην συνέχεια στην ιστοσελίδα <http://www.iep.edu.gr>. Το ιστορικό των επισκέψεων μέχρι στιγμής περιέχει τις τρεις αυτές ιστοσελίδες όπως φαίνεται παρακάτω:

http://www.iep.edu.gr
http://www.pythontutor.com
http://www.python.org

Η πρώτη ιστοσελίδα στην οποία έχουμε πρόσβαση είναι η τελευταία που επισκεφτήκαμε και βρίσκεται πάνω – πάνω στο ιστορικό. Αν μεταβούμε στην προηγούμενη ιστοσελίδα, τότε το ιστορικό θα πάρει τη μορφή:

http://www.pythontutor.com

http://www.python.org

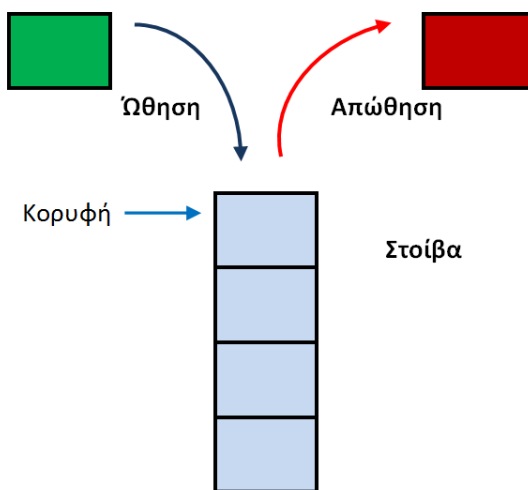
Αν στην συνέχεια μεταβούμε στην ιστοσελίδα <http://www.minedu.gov.gr/>, τότε αυτή θα προστεθεί στην κορυφή του ιστορικού:

http://www.minedu.gov.gr/

http://www.pythontutor.com

http://www.python.org

Από τα παραπάνω μπορούμε να συμπεράνουμε ότι η δομή που χρησιμοποιείται από το ιστορικό, είναι μια λίστα στην οποία οι εισαγωγές και οι διαγραφές στοιχείων γίνονται από το πάνω άκρο μόνο. Στη λίστα αυτή το στοιχείο που προστέθηκε τελευταίο είναι και το πρώτο που θα εξαχθεί, έχουμε δηλαδή μια λειτουργία τύπου LIFO (Last In First Out), δηλαδή ο τελευταίος που εισέρχεται στη λίστα, είναι και ο πρώτος που θα εξαχθεί. Η συγκεκριμένη δομή δεδομένων ονομάζεται **στοίβα** και οι λειτουργίες εισαγωγής και εξαγωγής είναι γνωστές στη βιβλιογραφία ως **ώθηση** και **απώθηση**.



Εικόνα 8-1. Οι βασικές λειτουργίες σε μια στοίβα είναι η ώθηση και η απώθηση

Ένα άλλο παράδειγμα που χρησιμοποιείται συχνά για να περιγράψει μια στοίβα, είναι αυτό του σωρού με πιάτα που περιμένουν να πλυθούν. Κάθε νέο πιάτο που έρχεται για πλύσιμο τοποθετείται πάνω στον σωρό, με αποτέλεσμα να είναι το πρώτο που θα πλυθεί.

Η στοίβα αποτελεί μια από τις σημαντικότερες δομές δεδομένων της επιστήμης της Πληροφορικής και χρησιμοποιείται σε πολλά πεδία της, όπως είναι η θεωρία αλγορίθμων, η ανάπτυξη μεταγλωττιστών, η τεχνητή νοημοσύνη κ.ά.

Όταν η στοίβα είναι άδεια, είναι προφανές ότι δεν μπορεί να γίνει απώθηση. Άρα, όταν απωθούμε ένα στοιχείο από τη στοίβα, θα πρέπει προηγουμένως να έχουμε εξασφαλίσει ότι η στοίβα δεν είναι κενή. Για αυτό το λόγο, εκτός από την ώθηση και την απώθηση, πρέπει να υλοποιήσουμε και τον έλεγχο, αν η στοίβα είναι κενή. Άρα οι βασικές λειτουργίες που πρέπει να υποστηρίξει η υλοποίηση μιας στοίβας είναι:

- Δημιουργία μιας κενής στοίβας.
- Έλεγχος, αν η στοίβα είναι κενή.
- Ωθηση ενός στοιχείου στη στοίβα.
- Απώθηση ενός στοιχείου από τη στοίβα.

Η δομή της στοίβας μπορεί να υλοποιηθεί στην Python με μια λίστα στην οποία οι εισαγωγές και οι εξαγωγές στοιχείων γίνονται μόνο από το ένα άκρο. Παρακάτω δίνονται δύο υλοποιήσεις της στοίβας. Στην πρώτη περίπτωση, οι εισαγωγές/διαγραφές στοιχείων γίνονται στο πίσω μέρος της λίστας, ενώ στη δεύτερη, από το εμπρός μέρος.

Υλοποίηση Στοίβας σε Python με δύο τρόπους	
<pre>def push(stack, item) : stack = stack + [item] def pop(stack) : return stack.pop() def isEmpty(stack) : return len(stack) == 0 def createStack() : return []</pre>	<pre>def push(stack, item) : stack = [item] + stack def pop(stack) : return stack.pop(0) def isEmpty(stack) : return len(stack) == 0 def createStack() : return []</pre>

Οι δύο υλοποιήσεις που δίνονται παραπάνω, διαφέρουν μόνο ως προς το σημείο της λίστας στο οποίο γίνονται οι εισαγωγές/διαγραφές των στοιχείων.

Εφαρμογή Στοίβας: Αντιστροφή αριθμών

Το παρακάτω πρόγραμμα δέχεται από το χρήστη αριθμούς μέχρι να δοθεί το 0 και τους εμφανίζει σε αντίστροφη σειρά από αυτή με την οποία δόθηκαν. Θέλουμε κάθε φορά να εμφανίσουμε πρώτο τον αριθμό που δόθηκε τελευταίος. Χρειαζόμαστε μια δομή δεδομένων που να υποστηρίζει τη λειτουργία LIFO, όπως η στοίβα.

```
stack = createStack()      # Δημιουργία της στοίβας stack
number = int( raw_input( ) ) # Διάβασε τον πρώτο αριθμό
while number != 0 :        # Όσο δεν δίνεται 0
    push(stack, number)    # Σπρώξε τον αριθμό στη στοίβα
    number = int( raw_input( ) ) # Διάβασε τον επόμενο αριθμό
```

<code>while ! isEmpty(stack) :</code>	<code># Μέχρι να αδειάσει η στοίβα</code>
<code> number = pop(stack)</code>	<code># βγάλε τον επόμενο αριθμό</code>
<code> print number</code>	<code># και εκτύπωσέ τον</code>

Παρατηρήστε ότι στο παραπάνω πρόγραμμα δεν φαίνεται ποια υλοποίηση χρησιμοποιείται. Αν τροποποιήσουμε την υλοποίηση της ώθησης, δε θα χρειαστεί να αλλάξουμε τίποτα στο πρόγραμμα. Αυτό είναι γνωστό ως διαχωρισμός **διεπαφής (διασύνδεσης) – υλοποίησης (interface – implementation)**, αφού οι εφαρμογές που χρησιμοποιούν δομές δεδομένων όπως η στοίβα, είναι απολύτως ανεξάρτητες από την υλοποίηση.

Ουσιαστικά δε μας ενδιαφέρει πώς έχει υλοποιηθεί η στοίβα, αφού εμείς θέλουμε μόνο να χρησιμοποιήσουμε τις συναρτήσεις που έχουμε ορίσει παραπάνω.

Το σύνολο των επικεφαλίδων των συναρτήσεων το οποίο είναι διαθέσιμο στον προγραμματιστή που χρησιμοποιεί τη στοίβα, το ονομάζουμε διεπαφή ή διασύνδεση (interface) της δομής αυτής. Η διεπαφή μιας δομής ορίζει τι μπορεί να κάνει η δομή και όχι τον τρόπο με τον οποίο το κάνει. Το τελευταίο είναι ο ρόλος της υλοποίησης.

Διεπαφή της δομής δεδομένων Στοίβα

```
def push(stack, item)
def pop(stack)
def isEmpty(stack)
def createStack()
```

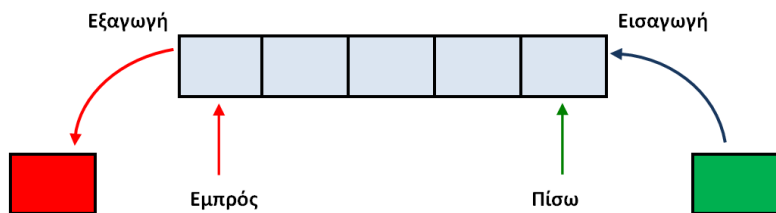
8.4 Ουρά

Μια δομή δεδομένων που χρησιμοποιείται για την μοντελοποίηση και προσομοίωση πραγματικών φαινομένων, είναι η δομή της ουράς. Τα φαινόμενα που μοντελοποιούνται αναφέρονται στην εξυπηρέτηση ανθρώπων, αντικειμένων ή προγραμμάτων. Τέτοια παραδείγματα ουρών είναι:

- Οι ουρές στις τράπεζες και τα σούπερ-μάρκετ.
- Η ουρά των προγραμμάτων που περιμένουν να εξυπηρετηθούν από τον επεξεργαστή του υπολογιστή σας.
- Η ουρά των αιτήσεων προς το διακομιστή ιστού (web server) ενός δικτυακού τόπου.

Τα φαινόμενα αυτά μελετώνται από διάφορους κλάδους των Μαθηματικών και της Πληροφορικής, όπως είναι η θεωρία ουρών (Queueing theory) και η Επιχειρησιακή έρευνα (Operations research).

Σε αντίθεση με τη στοίβα, που η λειτουργία της χαρακτηρίζεται ως LIFO (Last In First Out), η λειτουργία της ουράς είναι γνωστή στη βιβλιογραφία ως FIFO (First In First Out), αφού το κάθε στοιχείο της ουράς εξυπηρετείται με τη σειρά που έφτασε στην ουρά.



Εικόνα 8-2

Δύο είναι οι βασικές λειτουργίες μιας ουράς:

- Εισαγωγή στοιχείου, η οποία γίνεται στο πίσω μέρος της ουράς.
- Εξαγωγή στοιχείου, η οποία γίνεται από το εμπρός μέρος της ουράς.

Υλοποίηση Ουράς σε Python

```
def enqueue(queue, item) :
    queue = queue + [ item ]
def dequeue(queue) :
    return queue.pop( 0 )
def isEmpty(queue) :
```

```

return len(queue) == 0
def createQueue() :
    return []

```

8.5 Πλειάδες

Οι **πλειάδες** είναι σύνθετες δομές της Python, οι οποίες χρησιμοποιούνται για την αναπαράσταση οντοτήτων του πραγματικού κόσμου. Χαρακτηρίζονται από κάποιες ιδιότητες, όπως για παράδειγμα όνομα, επώνυμο, βαθμό, μισθό κλπ. Οι πλειάδες είναι ακολουθίες όπως οι λίστες, όμως δεν μπορούν να τροποποιηθούν. Ανήκουν στην κατηγορία των *μη τροποποιήσιμων* (immutables) δομών της Python, όπως είναι οι συμβολοσειρές. Στις πλειάδες τα στοιχεία χωρίζονται με κόμματα, όπως στις λίστες, αλλά, αντί να περικλείονται από αγκύλες, περικλείονται σε παρενθέσεις, οι οποίες όμως δεν είναι υποχρεωτικές.

Η πλειάδα μοιάζει με μια λίστα όπως φαίνεται στα παραδείγματα παρακάτω:

```

>>> rec = ("Edsger", "Dijkstra", 1940, 2006)
>>> print rec[0], rec[1], rec[2], rec[3]
Edsger Dijkstra 1830 1879
>>> len( rec )
4

```

Μια γνωστή εφαρμογή τους είναι η αμοιβαία αλλαγή των τιμών δυο μεταβλητών, χωρίς τη χρήση βοηθητικής μεταβλητής, λειτουργία η οποία είναι γνωστή ως *αντιμετάθεση* (swap):

```

>>> a = 496
>>> b = 28
>>> a , b = b, a      # αντιμετάθεση των τιμών των μεταβλητών
a, b
>>> print "a =", a, "b =", b
a = 28 b = 496

```

8.6 Λεξικά

Το **λεξικό** είναι μια εξαιρετικά χρήσιμη ενσωματωμένη (built-in) δομή της Python. Φανταστείτε το σαν ένα τηλεφωνικό κατάλογο ο οποίος μας δίνει τη δυνατότητα να βρούμε πολύ γρήγορα τα στοιχεία κάποιου, μόνο από το όνομά του. Για παράδειγμα, μπορούμε να γράψουμε `version["python"] = 3.4`. Όπως σε μια λίστα η θέση κάθε στοιχείου είναι μοναδική και δεν μπορεί να περιέχει ταυτόχρονα δυο τιμές, έτσι και στο λεξικό, η λέξη-κλειδί που χρησιμοποιούμε μέσα στις αγκύλες έχει μοναδική τιμή και εμφανίζεται το πολύ μία φορά. Είναι το λεγόμενο **κλειδί**.

Έτσι, ένα λεξικό είναι ουσιαστικά ένα σύνολο ζευγών κλειδιών-τιμών, όπου κάθε κλειδί δεν εμφανίζεται δεύτερη φορά. Μπορούμε να ορίσουμε ένα λεξικό όπως παρακάτω:

```
>>> empty_dictionary = dict() # δημιουργία ενός άδειου λεξικού
>>> print empty_dictionary
{ }
>>> dictionary = { 'Kilo':3, 'Mega':6, 'Giga':9, 'Tera':12 }
>>> print dictionary
{ 'Kilo': 3, 'Mega': 6, 'Giga': 9, 'Tera': 12 }
>>> dictionary['Giga'] # έτσι βρίσκω την τιμή του Giga
9
>>> len(dictionary) # το πλήθος των κλειδιών του λεξικού
4
```

Για να προσθέσουμε ένα ζεύγος κλειδί-τιμή στο λεξικό αρκεί να δώσουμε την αντίστοιχη εντολή ανάθεσης τιμής. Σε μια εντολή ανάθεσης τιμής :

Λεξικό[Κλειδί] = Τιμή

Αν το κλειδί υπάρχει ήδη στο λεξικό, η τιμή που είχε ενημερώνεται και στη θέση της μπαίνει η νέα τιμή Τιμή, ενώ αν δεν υπάρχει, τότε δημιουργείται ένα νέο ζεύγος Κλειδί:Τιμή. Παρακάτω προσθέτουμε

στο λεξικό το κλειδί 'Peta' και με τιμή 15 και στην συνέχεια τροποποιούμε την τιμή του 'Kilo' σε 1024.

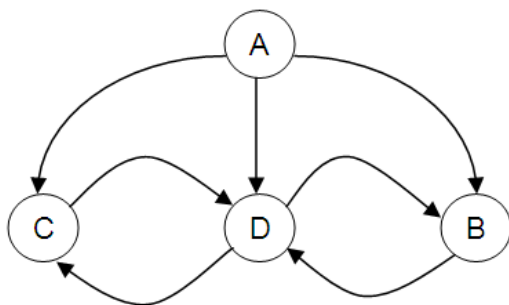
```
>>> dictionary ['Peta'] = 15
>>> print dictionary
{'Kilo': 3, 'Peta': 15, 'Mega': 6, 'Giga': 9, 'Tera': 12 }
>>> dictionary ['Kilo'] = 1024
>>> print dictionary
{'Kilo': 1024, 'Peta': 15, 'Mega': 6, 'Giga': 9, 'Tera': 12 }
```

Παρατηρήστε ότι τα στοιχεία του λεξικού δεν είναι διατεταγμένα και το νέο στοιχείο μπήκε σε μία τυχαία θέση. Στα λεξικά, όπως και στις λίστες και τις συμβολοσειρές, μπορούν να χρησιμοποιηθούν οι τελεστές **in** και **not in** για τον έλεγχο της ύπαρξης ενός κλειδιού στο λεξικό. Επίσης, για τη διαγραφή ενός κλειδιού από το λεξικό, χρησιμοποιείται ο τελεστής **del**.

>>> del dictionary ['Kilo']	>>> 'Kilo' in dictionary
>>> del dictionary ['Mega']	False
>>> print dictionary	>>> 'Peta' in dictionary
{ 'Peta': 15, 'Giga': 9, 'Tera': 12 }	True

8.7 Εισαγωγή στους γράφους και τα δέντρα

Ένας **γράφος** είναι μια δομή που αποτελείται από ένα σύνολο *κορυφών* ή *κόμβων* και ένα σύνολο *ακμών* μεταξύ των κορυφών. Οι κορυφές θα μπορούσαν να είναι οι πόλεις μιας χώρας και οι ακμές οι δρόμοι που τις ενώνουν. Όταν οι δρόμοι είναι διπλής κυκλοφορίας, λέμε ότι ο γράφος είναι **μη κατευθυνόμενος**, ενώ, όταν κάποιοι από τους δρόμους είναι μονόδρομοι, λέμε ότι ο γράφος είναι **κατευθυνόμενος**. Παρακάτω δίνεται ένας κατευθυνόμενος γράφος με 4 κόμβους A, B, C, D.



Εικόνα 8-3

Για να μεταβούμε από την κορυφή C στη B δεν υπάρχει απευθείας ακμή. Πρέπει πρώτα να περάσουμε από τη D. Η ακολουθία των ακμών από τις οποίες περνάμε για να φτάσουμε στον τελικό προορισμό, λέγεται *μονοπάτι*. Το μονοπάτι από τη C στη B συμβολίζεται $C \rightarrow D \rightarrow B$ και έχει μήκος 2 αφού αποτελείται από 2 ακμές. Δηλαδή, το *μήκος ενός μονοπατιού* είναι το πλήθος των ακμών του μονοπατιού. Σε αυτό το βιβλίο, κάνουμε την υπόθεση ότι όλες οι ακμές έχουν το ίδιο μήκος ή κόστος.

Παρατηρήστε ότι δεν υπάρχει κανένα μονοπάτι από κάποια κορυφή που να καταλήγει στην A. Ωστόσο, αν ακολουθήσουμε το μονοπάτι $B \rightarrow D \rightarrow C$, μπορούμε να επιστρέψουμε πίσω στη B, αφού υπάρχει δρόμος $C \rightarrow D \rightarrow B$. Αυτό ονομάζεται *κύκλος* ή *κυκλικό μονοπάτι*, επειδή, αν το ακολουθήσουμε επιστρέφουμε στο σημείο από το οποίο ξεκινήσαμε.

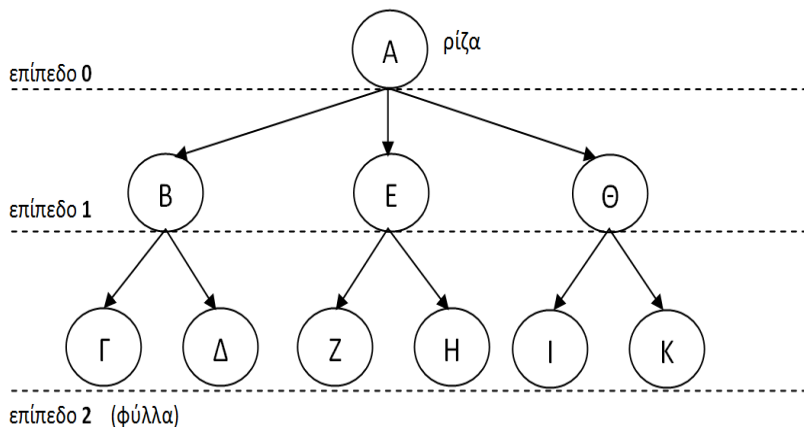
Η δομή ενός γράφου συναντάται σε πολλά προβλήματα της επιστήμης της Πληροφορικής, όπως για παράδειγμα στην αναπαράσταση του παγκόσμιου ιστού, την κατάταξη των ιστοσελίδων (page ranking), την εύρεση ενός μονοπατιού ή του συντομότερου μονοπατιού σε ένα δίκτυο πόλεων και άλλα. Μια ειδική περίπτωση γράφων είναι τα δέντρα.

Ένας γράφος στον οποίο υπάρχει μονοπάτι μεταξύ δυο οποιωνδήποτε κορυφών, και δεν περιέχει κυκλικές διαδρομές, λέγεται **δέντρο**.

Το δέντρο συνιστά μια ιεραρχική δομή η οποία χρησιμοποιείται για τη μοντελοποίηση διάφορων καταστάσεων, όπως για παράδειγμα ένα γενεαλογικό δέντρο ή η οργανωτική δομή μιας επιχείρησης. Αναφερόμαστε στα στοιχεία του δέντρου, όπως και του γράφου, ως *κόμβους*. Κάθε κόμβος συνδέεται με έναν ή περισσότερους κόμβους στους οποίους αναφερόμαστε, ως *παιδιά* του ή *απόγονούς* του. Οι κόμβοι που δεν έχουν απογόνους και βρίσκονται στο τελευταίο επίπεδο λέγονται *φύλλα* του δέντρου. Ο αριθμός των παιδιών που έχει ένας κόμβος ονομάζεται *βαθμός* του κόμβου. Ο μέγιστος βαθμός μεταξύ των κόμβων ενός δέντρου είναι ο *βαθμός του δέντρου*.

Κάθε κόμβος έχει ακριβώς έναν πρόγονο, εκτός από τη **ρίζα**, που βρίσκεται συνήθως στην κορυφή του δέντρου. Η απόσταση ενός κόμβου από τη ρίζα είναι το *επίπεδο* του κόμβου. Προφανώς, η ρίζα βρίσκεται στο μηδενικό (0) επίπεδο.

Το *ύψος* ενός δέντρου είναι η μέγιστη απόσταση κάποιου κόμβου από τη ρίζα. Όπως φαίνεται στο παρακάτω σχήμα, μέσω των επιπέδων, διακρίνεται πιο εύκολα η ιεραρχική δομή του δέντρου. Το δέντρο του σχήματος έχει ύψος 2, αφού το τελευταίο επίπεδο είναι το 2^ο.



Εικόνα 8-4. Ιεραρχική δομή του δέντρου

Δραστηριότητες - Ερωτήσεις - Ασκήσεις κεφαλαίου

8.8 Δραστηριότητες

Δραστηριότητα 1

Να γράψετε ένα πρόγραμμα το οποίο θα διαβάζει μία λέξη και θα εμφανίζει τα γράμματά της, ένα σε κάθε γραμμή.

Δραστηριότητα 2

Να γράψετε ένα πρόγραμμα το οποίο θα διαβάζει μία λέξη και θα εμφανίζει πόσα κεφαλαία αγγλικά γράμματα περιέχει η λέξη.

Δραστηριότητα 3

Να γράψετε ένα πρόγραμμα το οποίο θα διαβάζει μία λέξη και θα την εμφανίζει αντεστραμμένη, με τη χρήση μιας στοίβας.

Δραστηριότητα 4

Να γράψετε μια συνάρτηση `isSubstring(string, substring)` η οποία θα ελέγχει, αν η συμβολοσειρά `substring` περιέχεται στην συμβολοσειρά `string` και αν ναι, θα επιστρέφει `True`. Στη συνέχεια να χρησιμοποιήσετε αυτή τη συνάρτηση, για να βρείτε πόσες φορές εμφανίζεται μια συμβολοσειρά μέσα σε μια άλλη.

Δραστηριότητα 5

Να γράψετε ένα πρόγραμμα το οποίο θα διαβάζει αριθμούς από το πληκτρολόγιο, μέχρι να δοθεί ο αριθμός 0. Κάθε φορά που θα διαβάζει έναν θετικό αριθμό, θα τον προσθέτει σε μια στοίβα. Όταν διαβάζει έναν αρνητικό αριθμό θα αφαιρεί τόσους αριθμούς από τη στοίβα, όσο είναι η τιμή του αριθμού. Ο αλγόριθμος θα τερματίζει όταν αδειάσει η στοίβα.

Δραστηριότητα 6

Να γράψετε πρόγραμμα στη γλώσσα Python το οποίο θα δέχεται ως είσοδο ένα κείμενο και θα εμφανίζει πόσες φορές εμφανίζεται κάθε γράμμα του αγγλικού αλφαβήτου σε αυτό. Να χρησιμοποιήσετε ένα λεξικό.

Δραστηριότητα 7

Να γράψετε μια συνάρτηση σε Python η οποία θα δέχεται μια λίστα από λέξεις και θα επιστρέφει τη λέξη με το μεγαλύτερο μήκος.

Δραστηριότητα 8

Να γράψετε μια συνάρτηση σε Python η οποία θα δέχεται μια λέξη και θα επιστρέφει το πλήθος των φωνηέντων που έχει. Στη συνέχεια, να γράψετε μια δεύτερη συνάρτηση, η οποία θα δέχεται μια λίστα από λέξεις και θα επιστρέφει τη λέξη με τα περισσότερα φωνήεντα.

8.9 Ερωτήσεις

1. Να περιγράψετε τη δομή δεδομένων “Συμβολοσειρά” (str) στην Python και να αναφέρετε τους τελεστές επεξεργασίας της.
2. Να περιγράψετε τη δομή δεδομένων “Λίστα” στην Python και να αναφέρετε τους τελεστές επεξεργασίας της.
3. Να αναφέρετε τις δομές δεδομένων οι οποίες επιτρέπουν τροποποίηση των δεδομένων τους (mutables).
4. Να αναφέρετε τις δομές δεδομένων οι οποίες δεν επιτρέπουν τροποποίηση των δεδομένων τους (immutable).
5. Πότε χρησιμοποιούμε τη δομή δεδομένων του λεξικού;
6. Να αναφέρετε μια εφαρμογή των πλειάδων.
7. Ποιες είναι οι βασικές λειτουργίες σε μια στοίβα και ποιες σε μια ουρά;
8. Να αναφέρετε εφαρμογές της στοίβας.
9. Να αναφέρετε εφαρμογές της ουράς.
10. Σε τι διαφέρει ένα δέντρο από ένα γράφο;

Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκαν οι βασικές δομές δεδομένων της Python που είναι ενσωματωμένες στη γλώσσα και χωρίζονται σε δυο κατηγορίες. Αυτές που επιτρέπουν την τροποποίηση των περιεχομένων τους (mutables), όπως οι λίστες και τα λεξικά και αυτές που δεν το επιτρέπουν, όπως οι συμβολοσειρές και οι πλειάδες. Στη συνέχεια, με τη βοήθεια της δομής της λίστας υλοποιήθηκαν οι δομές δεδομένων της στοίβας και της ουράς και παρουσιάστηκαν κάποιες εφαρμογές τους.

Τέλος παρουσιάστηκε η δομή δεδομένων του γράφου και του δέντρου.

9

**Εφαρμογές σε γλώσσα
προγραμματισμού
με χρήση API**

9. Εφαρμογές σε γλώσσα προγραμματισμού με χρήση API

Εισαγωγή

Στο κεφάλαιο αυτό θα παρουσιαστούν ορισμένα θεωρητικά στοιχεία των διεπαφών και στη συνέχεια θα γίνει υλοποίηση μικρών Προγράμματος-εφαρμογών με την βιβλιοθήκη **Tkinter**. Η Tkinter είναι η πρότυπη βιβλιοθήκη της Python για την υλοποίηση γραφικών διεπαφών (Graphical User Interface, GUI). Με τον τρόπο αυτό θα έχουμε την ευκαιρία να δούμε πώς μπορούμε να κάνουμε τα προγράμματά μας πιο ελκυστικά και φιλικά προς το χρήστη, χρησιμοποιώντας μια βιβλιοθήκη με έτοιμα τμήματα κώδικα.

Η συνεισφορά της μεγάλης κοινότητας προγραμματιστών που υποστηρίζει τη γλώσσα, με συλλογές έτοιμου κώδικα για πάρα πολλές εφαρμογές, αποτελεί ένα ισχυρό πλεονέκτημα της γλώσσας προγραμματισμού Python. Οι περισσότερες από αυτές τις εφαρμογές ανήκουν στην κατηγορία του ελεύθερου λογισμικού και είναι γνωστές ως Διεπαφές Προγραμματισμού Εφαρμογών (Application Programming Interfaces-APIs) ή και ως βιβλιοθήκες λογισμικού.

Ωστόσο, ο βασικός σκοπός αυτού του κεφαλαίου δεν είναι να μάθουμε ένα συγκεκριμένο API, ούτε φυσικά να “αποστηθίσουμε” κάποιες βασικές συναρτήσεις των βιβλιοθηκών. Σκοπός του είναι να αντιληφθούμε ότι, κατά την ανάπτυξη μιας εφαρμογής, είναι καλή πρακτική η αξιοποίηση τυχόν διαθέσιμων τμημάτων έτοιμου κώδικα που ανήκουν σε κάποιο API και να πειραματιστούμε με τις δυνατότητες μιας τέτοιας βιβλιοθήκης.

Διδακτικοί στόχοι

Μετά τη μελέτη του κεφαλαίου θα μπορούμε να:

- περιγράψουμε τις βασικές αρχές της επικοινωνίας ανθρώπου υπολογιστή
- χρησιμοποιούμε περιβάλλοντα, Application Program Interfaces (APIs) και βιβλιοθήκες για την ανάπτυξη και

τροποποίηση εφαρμογών λογισμικού στη γλώσσα προγραμματισμού Python.

Λέξεις κλειδιά

Επικοινωνία ανθρώπου υπολογιστή, διεπαφή χρήστη, διεπαφές προγραμματισμού εφαρμογών API, βιβλιοθήκες λογισμικού.

Διδακτικές Ενότητες

9.1 Επικοινωνία ανθρώπου-υπολογιστή και διεπαφή χρήστη

Καθημερινά ερχόμαστε σε επαφή με πολλές συσκευές και εφαρμογές οι οποίες εκτελούν βασικές εργασίες, όπως για παράδειγμα τα μηχανήματα ATM (Automated Teller Machines) στις τράπεζες, τα tablet pc's, μια εφαρμογή αναζήτησης ενός βιβλίου σε ένα Διαδικτυακό βιβλιοπωλείο. Είναι ιδιαίτερα σημαντικό οι συσκευές και οι εφαρμογές τους να είναι κατάλληλα σχεδιασμένες, ώστε οι χρήστες να εκτελούν τις εργασίες που θέλουν εύκολα, αναγνωρίζοντας το επόμενο κάθε φορά βήμα, χωρίς λάθη και όσο το δυνατό πιο γρήγορα. Τα αντικείμενα, τα εικονίδια και τα μηνύματα που εμφανίζονται στην οθόνη των συσκευών αυτών, οι επιλογές σε μια οθόνη αφής (soft-touch), η είσοδος που δίνουν οι χρήστες με συσκευές όπως το ποντίκι, ο τρόπος ανατροφοδότησης του χρήστη μετά από κάποια ενέργειά του, αποτελούν αντικείμενο έρευνας του ερευνητικού πεδίου που ονομάζεται *αλληλεπίδραση ανθρώπου-υπολογιστή* (Human Computer Interaction, HCI).

Η **Επικοινωνία Ανθρώπου-Υπολογιστή** (EAY) αποτελεί ένα πολλά υποσχόμενο και ταχύτατα αναπτυσσόμενο τομέα που ασχολείται με το σχεδιασμό, την υλοποίηση και την αξιολόγηση διαδραστικών υπολογιστικών συστημάτων προορισμένων για ανθρώπινη χρήση και τη μελέτη σημαντικών φαινομένων γύρω από αυτά (πηγή Association for Computer Machinery-ACM).

Ο βασικός στόχος της Επικοινωνίας Ανθρώπου-Υπολογιστή είναι η βελτίωση της ευχρηστίας του συστήματος (usability). Με τον όρο *ευχρηστία ενός συστήματος* και σύμφωνα με το διεθνές πρότυπο ISO 9241, περιγράφουμε την ικανότητα ενός συστήματος να

λειτουργεί αποτελεσματικά και αποδοτικά, παρέχοντας υποκειμενική ικανοποίηση στους χρήστες του.

Η ευχρηστία του συστήματος επιτυγχάνεται όταν η επικοινωνία ενός χρήστη με ένα σύστημα υπολογιστή, γίνεται μέσω ενός περιβάλλοντος που χαρακτηρίζεται από:

- ευκολία εκμάθησης
- υψηλή απόδοση εκτέλεσης έργου (επίδοση)
- χαμηλή συχνότητα εμφάνισης λαθών χρήστη
- ευκολία συγκράτησης της γνώσης της χρήσης του και
- υποκειμενική ικανοποίηση του χρήστη, δηλαδή το κατά πόσο ικανοποιημένοι είναι οι χρήστες από τη χρήση του συστήματος.

Η αλληλεπίδραση μεταξύ χρηστών και υπολογιστών γίνεται στο επίπεδο της διεπαφής χρήστη (user interface), μέσω κατάλληλου λογισμικού και υλικού.

Η μελέτη ανάπτυξης της διεπαφής χρήστη επικεντρώνεται στην κατάλληλη δημιουργία και προσαρμογή εκείνων των στοιχείων του υλικού και του λογισμικού του υπολογιστή, με τα οποία ο άνθρωπος έρχεται σε άμεση επαφή και αποτελεί μέρος του ευρύτερου τομέα της Επικοινωνίας Ανθρώπου-Απολογητή για τη βελτίωση της ευχρηστίας ενός συστήματος (usability).

9.1.1 Παραδείγματα – Εφαρμογές

Η χρηστικότητα μιας διεπαφής σχετίζεται άμεσα με την πληροφορία που προσφέρει στο χρήστη, συνειδητά ή και υποσυνειδητά, για να εκτελέσει μια ενέργεια ή να έχει χρήσιμη ανατροφοδότηση. Ας παρατηρήσουμε, για παράδειγμα, την πληροφορία που μας δίνουν οι σχεδιαστές για να ανοίξουμε τις πόρτες που συναντάμε πολλές φορές σε καταστήματα. Πότε πρέπει να τραβήξουμε μια πόρτα για να την ανοίξουμε και πότε απλώς πρέπει να τη σπρώξουμε; Το μήνυμα «ωθήσατε», που πολλές φορές βρίσκεται πάνω στην πόρτα, παρότι μας πληροφορεί για την απαραίτητη ενέργεια, δεν μας οδηγεί άμεσα στο να κάνουμε αυτόματα την ενέργεια αυτή. Θα ήταν πιο εύκολο οι σχεδιαστές με τον κατάλληλο σχεδιασμό της πόρτας να παρείχαν στο χρήστη την ανάλογη πληροφορία, ώστε αυτόματα να εκτελεί αυτή την ενέργεια. Αρκεί στην πλευρά της πόρτας που

απαιτείται να τη σπρώξουμε (ώθηση), να μην είχε προσαρμοστεί χερούλι παρά μόνο μια μικρή βιδωμένη μεταλλική πλάκα, ενώ στη πλευρά που απαιτείται να την τραβήξουμε να είχε προστεθεί μία χειρολαβή.

Η αντίληψη κάποιας πληροφορίας από τη μεριά του χρήστη επηρεάζεται σημαντικά από τη *μεταφορά* που χρησιμοποιείται για την οπτική της αναπαράσταση και αποτελεί μια νοηματική σύνδεση της ενέργειας με ένα χαρακτηριστικό εικονίδιο. Ως παράδειγμα γνώριμης μεταφοράς σκεφθείτε τη χρήση του πράσινου και του κόκκινου εικονιδίου, που απεικονίζει ένα ακουστικό στο κινητό τηλέφωνο για να ξεκινήσουμε ή να τερματίσουμε μια κλήση αντίστοιχα. Άλλο παράδειγμα, είναι η χρήση του εικονιδίου της δισκέτας για να απεικονίσουμε τη διαδικασία αποθήκευσης μιας εργασίας μας στον επεξεργαστή κειμένου. Γενικά στο σχεδιασμό μιας διεπαφής, οι μεταφορές χρησιμοποιούνται για την απεικόνιση μιας νέας πληροφορίας ή διεργασίας με κάποιο οπτικό αντικείμενο, που συνήθως είναι γνώριμο, ώστε εύκολα να καταλάβουμε το νόημά της.

Για το σχεδιασμό μιας ATM συσκευής και των εφαρμογών της σε μια τράπεζα, αναδεικνύονται διάφορα σημαντικά ζητήματα ως προς το σχεδιασμό της διεπαφής με τους πελάτες της τράπεζας, που πρέπει να διερευνηθούν. Για παράδειγμα, ποίός είναι ο ελάχιστος αριθμός ενεργειών που απαιτούνται για να κάνει κάποιος μια ανάληψη σε ένα μηχάνημα ATM; Τι πρέπει να προβλεφθεί για να μπορεί να χειριστεί ένα ATM ένα ηλικιωμένο άτομο που δεν είναι εξοικειωμένο με τους υπολογιστές και πιθανά να έχει και προβλήματα όρασης χωρίς να κάνει λάθη; Τι πρέπει να προβλέπεται, ώστε να αποτρέπονται πιθανά λανθασμένες ενέργειες;

9.2 Γενικές αρχές σχεδίασης διεπαφής

Κατά τη διάρκεια του σχεδιασμού της διεπαφής, ο σχεδιαστής της λαμβάνει πάντοτε υπόψη του ότι σχεδιάζει ένα σύστημα για μια ευρύτερη ποικιλία ατόμων-χρηστών και δε περιορίζεται μόνο στους έμπειρους χρήστες. Κάτω από αυτό το πρίσμα η σχεδιαζόμενη διεπαφή πρέπει να προβλέπει τις ανθρώπινες ιδιαιτερότητες. Για παράδειγμα, η επιλογή των χρωμάτων, των

ετικετών, της τοποθέτησης των πλήκτρων και των λειτουργιών δεν θα πρέπει να αντιβαίνουν στις συνήθειες και τα χαρακτηριστικά του χρήστη. Αν η εφαρμογή απευθύνεται σε άτομα ηλικιωμένα ή άτομα με ειδικές ανάγκες, τότε πρέπει να ληφθούν υπόψη τα ιδιαίτερα χαρακτηριστικά τους. Για παράδειγμα, εφαρμογές για άτομα με προβλήματα όρασης, πρέπει να συμπεριλάβουν πολύ μεγαλύτερα εικονίδια, ηχητικά μηνύματα ανατροφοδότησης κ.ά.

Ο σκοπός της σχεδίασης μιας διεπαφής είναι η δημιουργία όσο το δυνατόν πιο εύχρηστων συστημάτων. Στο σχεδιασμό μιας διεπαφής πρέπει να ακολουθούνται ορισμένοι κανόνες, με διάφορους ερευνητές να έχουν προτείνει δικά τους συστήματα κανόνων που προτείνουν να τηρούνται. Ο Shneiderman (Shneiderman and Plaisant, 2005) για παράδειγμα για το σχεδιασμό μιας διεπαφής προτείνει διάφορους κανόνες, όπως:

1. Ομοιομορφία και συνέπεια (consistency) στη λειτουργία της διεπαφής και αποφυγή απροσδόκητης συμπεριφοράς του συστήματος στα παρακάτω:
 - 1) Ορολογία.
 - 2) Μηνύματα.
 - 3) Μενού.
 - 4) Οθόνες βοήθειας.
 - 5) Γραμματοσειρές.
 - 6) Χρώμα.
 - 7) Μορφή.
2. Σύντομοι χειρισμοί (shortcut) για τη διευκόλυνση των έμπειρων χρηστών:
 - 1) Συντμήσεις.
 - 2) Ειδικά πλήκτρα.
 - 3) Μακροεντολές.
3. Συνεχής ανατροφοδότηση της κατάστασης (informative feedback) του συστήματος. Για παράδειγμα το σύστημα πρέπει να απεικονίζει την πορεία εξέλιξης μιας τρέχουσας εργασίας.

Για την εργασία *αντιγραφή αρχείου* μια μπάρα απεικονίζει το ποσοστό από το αρχείο που έχει αντιγραφεί δίνοντας παράλληλα χρήσιμη πληροφορία στο χρήστη για το χρόνο αναμονής.

4. *Διάλογοι* χρήστη - υπολογιστή, που πρέπει να ολοκληρώνονται σε λίγα βήματα.
5. Πρόβλεψη για σφάλματα των χρηστών και χειρισμός σφαλμάτων.
6. Δυνατότητα αναίρεσης μιας ή περισσότερων ενεργειών με ευκολία.
7. Έλεγχος της αλληλεπίδρασης, από την πλευρά του χρήστη και όχι του συστήματος.
8. Ελαχιστοποίηση του αριθμού των αντικειμένων που καλείται κάθε φορά να συγκρατήσει ένας χρήστης στη βραχύχρονη μνήμη (κανόνας των 7 ± 2 αντικειμένων ως μέγιστη απαίτηση του συστήματος για το φόρτο της βραχύχρονης μνήμης του χρήστη).

9.3 Η βιβλιοθήκη Tkinter για ανάπτυξη γραφικών διεπαφών GUI στην Python

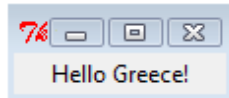
Στην ενότητα αυτή θα υλοποιήσουμε απλά παραδείγματα ανάπτυξης διεπαφών. Μέσα από τη δημιουργία μηνυμάτων ανατροφοδότησης, κουμπιών για την εκτέλεση κάποιων διεργασιών και πεδίων εισόδων για την εισαγωγή δεδομένων, θα έχουμε την ευκαιρία να διερευνήσουμε πώς να κάνουμε τα προγράμματά μας πιο φιλικά προς το χρήστη, δίνοντας τη δυνατότητα αλληλεπίδρασης με το περιβάλλον των εφαρμογών μας. Για το σκοπό αυτό θα χρησιμοποιήσουμε τη βιβλιοθήκη Tkinter, που μας προσφέρει πολλές δυνατότητες για να κατασκευάσουμε γραφικές διεπαφές.

Για να μπορέσουμε να αναπτύξουμε τα προγράμματά μας χρησιμοποιώντας τη Tkinter, πρέπει να ακολουθήσουμε τη βασική προγραμματιστική λογική στην οποία αυτή η βιβλιοθήκη βασίζεται. Βασικό στοιχείο της είναι τα παράθυρα, μέσα στα οποία μπορούμε

να τοποθετήσουμε και άλλα στοιχεία που μας παρέχει η βιβλιοθήκη όπως, μηνύματα, εικόνες, κουμπιά κ.ά. Τα στοιχεία αυτά αποτελούν αντικείμενα, που τα ονομάζουμε *widgets*. Για τη δημιουργία και τη διαχείριση των αντικειμένων αυτών, χρησιμοποιούμε κατάλληλα στιγμιότυπα κλάσεων που παρέχει η Tkinter.

Παράδειγμα 1

Ας ξεκινήσουμε με το πρώτο παράδειγμα για τη δημιουργία ενός μηνύματος χαιρετισμού, όπως στην παρακάτω εικόνα.



Αρχικά θα δημιουργήσουμε ένα παράθυρο με τίτλο `window_test`, το οποίο όμως δεν θα εμφανιστεί αμέσως. Στο αντικείμενο αυτό θα προσθέσουμε στη συνέχεια ένα αντικείμενο *ετικέτας κειμένου* (Label) για να εμφανιστεί ένα μήνυμα χαιρετισμού. Για το σκοπό αυτό θα δημιουργήσουμε ένα στιγμιότυπο της κλάσης Label με την ανάλογη έκφραση.

Μελετήστε τον κώδικα που ακολουθεί, διαβάζοντας τα επεξηγηματικά σχόλια. Στη συνέχεια, πληκτρολογήστε τον κώδικα στο προγραμματιστικό περιβάλλον IDLE, για να δείτε το αποτέλεσμα και δοκιμάστε να εμφανίζονται τα δικά σας μηνύματα χαιρετισμού.

```
from Tkinter import * #εισάγει το module Tkinter
```

```
window_test = Tk()
```

#δημιουργεί ένα αρχικό γραφικό στοιχείο –ένα απλό παράθυρο με μπάρα τίτλου, το οποίο δεν έχουμε εμφανίσει ακόμα. Πρέπει να δημιουργηθεί όμως πριν φτιαχτεί κάποιο άλλο γραφικό στοιχείο και είναι μοναδικό. Το όνομα `window_test` το χρησιμοποιούμε ως αναφορά στο παράθυρο που στη συνέχεια θα διαχειριστούμε για να τοποθετήσουμε τα υπόλοιπα γραφικά στοιχεία.

```
a = Label(window_test, text="Hello Greece!")
```

#δημιουργεί ένα γραφικό στοιχείο με όνομα `a` ως Label. Η πρώτη παράμετρος της Label είναι το όνομα του παραθύρου

γονέα δηλ του `window_test`. Η δεύτερη παράμετρος ορίζει το κείμενο που θα εμφανίζεται.

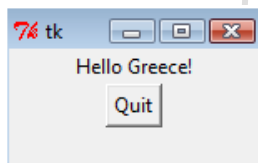
`a.pack()`

Η μέθοδος `pack` τοποθετεί το `a` γραφικό στοιχείο κειμένου στο παράθυρο και ελέγχουμε την εμφάνιση του παραθύρου.

`window_test.mainloop()`

#το script θα παραμείνει ενεργό μέχρι να κλείσει το παράθυρο. Η τελευταία εντολή θα εμφανίσει το παράθυρο

Στη συνέχεια θα τροποποιήσουμε κατάλληλα τον παραπάνω κώδικα, ώστε να αλλάξουμε το μέγεθος του παραθύρου με τη μέθοδο `wm_geometry()` και να προσθέσουμε ένα κουμπί με όνομα `buttonA`, χρησιμοποιώντας το widget `Button`. Το κουμπί αυτό θα το χρησιμοποιήσουμε, ώστε να τερματίζει το πρόγραμμα, όταν το πατήσει ο χρήστης.



```
from Tkinter import *
```

```
window_test = Tk()
```

```
window_test.wm_geometry("150x70+10+20")
```

#Η μέθοδος `wm_geometry` μας επιτρέπει να αλλάξουμε το μέγεθος του παραθύρου και να δηλώσουμε τις συντεταγμένες της πάνω αριστερά γωνίας

```
a = Label(window_test, text="Hello Greece!")
```

```
a.pack()
```

```
buttonA = Button(window_test, text="Quit",
                  command=window_test.destroy)
```

#δημιουργεί ένα γραφικό στοιχείο κουμπί που αναφέρεται με το όνομα `buttonA`. Η πρώτη παράμετρος της `Button` είναι το όνομα του παραθύρου γονέα δηλ του `window_test`. Η δεύτερη παράμετρος ορίζει το κείμενο που θα εμφανίζεται πάνω στο

κουμπί και η τρίτη παράμετρος ορίζει την ενέργεια που θα εκτελεστεί όταν πατηθεί το κουμπί.

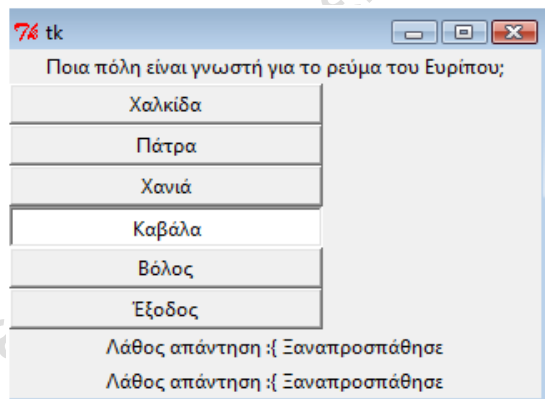
```
buttonA.pack()
```

Η μέθοδος pack τοποθετεί το γραφικό στοιχείο buttonA στο παράθυρο και ελέγχουμε την εμφάνιση του παραθύρου.

```
window_test.mainloop()
```

Παράδειγμα 2

Χρησιμοποιώντας το widget RadioButton, που χρησιμοποιείται για να εισάγουμε πολλαπλές επιλογές, θα δημιουργήσουμε ένα μικρό quiz γνώσεων. Αν η επιλογή είναι λανθασμένη, θα πρέπει να εμφανίζεται ένα μήνυμα λάθους, ενώ αν η απάντηση είναι σωστή θα εμφανίζεται ένα μήνυμα επιβράβευσης. Επιπρόσθετα, θα προσθέσουμε ένα κουμπί για έξοδο από το quiz, όπως φαίνεται στην παρακάτω εικόνα.



```
# -*- coding: utf-8 -*-
from Tkinter import *
root = Tk()
c = IntVar()
c.set(0) # αρχικοποίηση επιλογών
towns = [
    ("Χαλκίδα",1),
    ("Πάτρα",2),
```

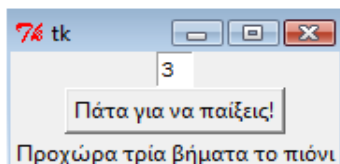
```
("Χανιά",3),
("Καβάλα",4),
("Βόλος",5),
("Εξοδος",6)
]
def ShowChoice():
    #συνάρτηση που ορίζει τις ενέργειες που θα εκτελεστούν
    ανάλογα με την επιλογή που θα κάνει ο παίκτης του
    παιχνιδιού
    if c.get()==6:
        root.destroy()
    elif c.get()==1:
        Label(root, text= 'Μπράβο!! Σωστά απάντησες.
        Κέρδισες 10 πόντους').pack()
    else:
        Label(root, text= 'Λάθος απάντηση,
        Ξαναπροσπάθησε').pack()

Label(root, text="Ποια πόλη είναι γνωστή για το ρεύμα του
Ευρίπου;", justify = LEFT,
    padx = 20).pack()
for txt, val in towns:
    Radiobutton(root,
        text=txt,
        indicatoron = 0,
        width = 20,
        padx = 20,
        variable=c,
        command= ShowChoice,
        value=val).pack(anchor=W)

mainloop()
```

Παράδειγμα 3

Σας έχουν αναθέσει τη δημιουργία ενός παιχνιδιού γνώσεων, όπου σε κάθε πάτημα ενός κουμπιού θα ορίζεται τυχαία ο αριθμός των βημάτων (1-6) που θα κάνει το πιόνι του παίκτη.



```
# -*- coding: utf-8 -*-
from Tkinter import *
from random import randint
window= Tk()
text= Text (window, width = 2, height =1)
#ορίζει τη λευκή περιοχή που θα εμφανίζεται ο αριθμός των
βημάτων που θα κάνει το πιόνι
text.pack()
def roll():
#συνάρτηση που ορίζει τις ενέργειες που τις οδηγίες που θα
εμφανιστούν ανάλογα με τον αριθμό των βημάτων.
    text.delete(0.0, END)
    a=str (randint(1,6))
    text.insert(END,a)
    if a=="1":
        w=Label(window, text='Προχώρα ένα βήμα το πιόνι')
    elif a=="2":
        w=Label(window, text='Προχώρα δύο βήματα το πιόνι')
    elif a=="3":
        w=Label(window, text='Προχώρα τρία βήματα το πιόνι')
    elif a=="4":
        w=Label(window, text='Προχώρα τέσσερα βήματα το
πιόνι')
    elif a=="5":
        w=Label(window, text='Προχώρα πέντε βήματα το πιόνι')
    else:
        w=Label(window, text= 'Προχώρα έξι βήματα το πιόνι')
    w.pack()
    return a
buttonA = Button (window, text = 'Πάτα για να παίξεις!',
command = roll)
```

#δημιουργεί ένα γραφικό στοιχείο κουμπί που αναφέρεται με το όνομα buttonA. Η πρώτη παράμετρος της Button είναι το όνομα του παραθύρου γονέα δηλ του window. Η δεύτερη παράμετρος ορίζει το κείμενο που θα εμφανίζεται πάνω στο κουμπί και η τρίτη παράμετρος ορίζει την ενέργεια που θα εκτελεστεί όταν πατηθεί το κουμπί δηλ. καλεί τη συνάρτηση roll () που δημιουργήσαμε παραπάνω.

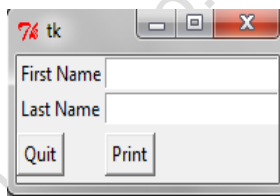
buttonA.pack()

Η μέθοδος pack τοποθετεί το γραφικό στοιχείο buttonA στο παράθυρο και ελέγχουμε την εμφάνιση του παραθύρου.

window.mainloop()

Παράδειγμα 4: Εισαγωγή δεδομένων

Πολύ συχνά χρειάζεται σε μια εφαρμογή να διαβάσουμε κάποια δεδομένα από το πληκτρολόγιο, εμφανίζοντας στο χρήστη μια φόρμα εισαγωγής δεδομένων. Στη συνέχεια τα δεδομένα αυτά μπορούμε να τα χρησιμοποιήσουμε σε κάποιους υπολογισμούς ή απλά να τα εμφανίσουμε στην οθόνη του υπολογιστή.



Για την εισαγωγή δεδομένων χρησιμοποιούμε τα widgets Entry, τα οποία δημιουργούν ένα πλαίσιο μέσα στο οποίο ο χρήστης μπορεί να εισάγει μια γραμμή κειμένου. Για να μπορέσουμε να έχουμε πρόσβαση και να διαβάσουμε τα περιεχόμενα που εισάγει ο χρήστης χρησιμοποιούμε τη μέθοδο get().

Στο παράδειγμα που ακολουθεί χρησιμοποιούμε δύο πεδία Entry για να εισάγουμε το όνομα (fname) και το επίθετο (lname) ενός μαθητή. Στη συνέχεια χρησιμοποιούμε δύο κουμπιά, το Quit για να τερματίσει το πρόγραμμα και το κουμπί Print για να εμφανιστούν τα περιεχόμενα. Το κουμπί Print όταν πατηθεί, καλεί τη συνάρτηση show_eisodo() που έχουμε δημιουργήσει. Η

συνάρτηση χρησιμοποιεί τη μέθοδο `get()` για να διαβάσει το όνομα `fname.get()` και το επίθετο `lname.get(0)` ενός μαθητή και με τη χρήση της `print` εμφανίζονται στη συνέχεια στην οθόνη του υπολογιστή. Για να διαγράψουμε τα στοιχεία από τη φόρμα, ώστε να εισάγουμε ξανά νέα στοιχεία χρησιμοποιούμε τη μέθοδο `delete()` για διαγραφή. Η μέθοδος `grid()` χρησιμοποιείται για να οργανώσει κατά γραμμές και στήλες την εμφάνιση των γραφικών αντικειμένων (`Label`, `Entry`, `Button`) που εμφανίζονται στο παράθυρο. `from Tkinter import *`

```
def show_eisodo():  
    print("First Name: %s\nLast Name: %s" % (fname.get(),  
    lname.get()))  
    fname.delete(0,END)  
    lname.delete(0,END)  
  
window = Tk()  
Label(window, text="First Name").grid(row=0)  
Label(window, text="Last Name").grid(row=1)  
  
fname = Entry(window)  
lname = Entry(window)  
fname.grid(row=0, column=1)  
lname.grid(row=1, column=1)  
  
Button(window, text='Quit', command=window.quit).grid(row=3,  
column=0, sticky=W, pady=4)  
Button(window, text='Print',  
command=show_eisodo).grid(row=3, column=1, sticky=W,  
pady=4)  
mainloop( )
```

Δραστηριότητες - Ερωτήσεις - Ασκήσεις κεφαλαίου

9.4 Δραστηριότητες

Δραστηριότητα 1

Τροποποιήστε κατάλληλα τον κώδικα του 1^{ου} παραδείγματος για τη δημιουργία μηνυμάτων, ώστε να εμφανίσετε τα δικά σας μηνύματα με χρήση του Label.

Δραστηριότητα 2

Τροποποιήστε κατάλληλα το 2^ο παράδειγμα, που παρουσιάστηκε παραπάνω, ώστε να δημιουργήσετε το δικό σας quiz γνώσεων.

Δραστηριότητα 3

Δοκιμάστε στο 4ο παράδειγμα, που παρουσιάστηκε παραπάνω, να εμπλουτίσετε κατάλληλα τη φόρμα εισαγωγής στοιχείων, ώστε να εισάγονται και στη συνέχεια να εμφανίζονται περισσότερες πληροφορίες για το μαθητή, όπως το τμήμα και η τάξη του, η ηλικία, το φύλο, ο M_O των βαθμών τριμήνου κ.ά.

Αν θέλετε να βρείτε περισσότερες πληροφορίες για τις δυνατότητες που μας παρέχει η βιβλιοθήκη Tkinter επισκεφτείτε τις προτεινόμενες πηγές καθώς και τη βοήθεια του Tkinter που είναι διαθέσιμη στην ιστοσελίδα της Python.

9.4.1 Εργαστηριακές ασκήσεις

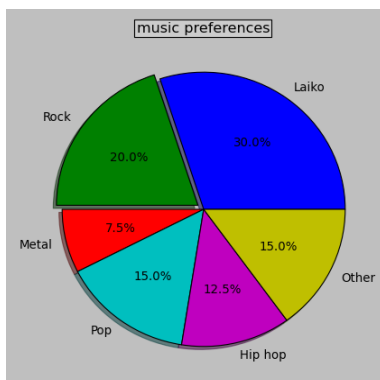
Άσκηση 1

Επισκεφτείτε το δικτυακό τόπο υποστήριξης της βιβλιοθήκης matplotlib <http://matplotlib.org/examples>, που χρησιμοποιείται για την οπτικοποίηση δεδομένων. Δείτε μερικά έτοιμα παραδείγματα και συζητήστε στη τάξη για τις δυνατότητες που μας δίνει η βιβλιοθήκη, προγραμματίζοντας σε γλώσσα Python και για την αξία που έχει η οπτικοποίηση των δεδομένων. Δώστε χαρακτηριστικά παραδείγματα. Στη συνέχεια μελετήστε τα έτοιμα παραδείγματα οπτικοποίησης στατιστικών δεδομένων για τη δημιουργία ενός γραφήματος πίτας (pie).

Επέκταση της άσκησης

Ακολουθήστε τις οδηγίες για την εγκατάσταση της βιβλιοθήκης αν χρειαστεί, ανάλογα με το προγραμματιστικό περιβάλλον και το ΛΣ του υπολογιστή σας. Για παράδειγμα σε ΛΣ Debian\Ubuntu αρκεί η: **sudo apt-get build-dep python-matplotlib** για να εγκατασταθεί η βιβλιοθήκη μαζί με όλες τις άλλες βιβλιοθήκες που συσχετίζεται.

Στη συνέχεια τροποποιήστε το παράδειγμα οπτικοποίησης δεδομένων με πίτα, ώστε να απεικονιστούν, όπως στην εικόνα, τα δεδομένα του παρακάτω προβλήματος.



Εικόνα: Η οπτικοποίηση των δεδομένων του παρακάτω προβλήματος σε πίτα με αξιοποίηση της βιβλιοθήκης matplotlib και χρήση της pie() με αυτόματη απεικόνιση των ποσοστών.

Το πρόβλημα (από το βιβλίο Μαθηματικών Β' Γυμνασίου).

Μια δισκογραφική εταιρεία προσπαθεί να επεκτείνει τις πωλήσεις της σε εφήβους. Προτού να επενδύσει σε είδη μουσικής που προτιμούν οι μαθητές, αποφασίζει να κάνει μια έρευνα ανάμεσα σε 200 μαθητές που επέλεξε τυχαία απ' όλη την Ελλάδα. Ο υπεύθυνος, που έκανε την έρευνα, παρουσίασε στο διευθυντή της εταιρείας τα παρακάτω δεδομένα:

60 μαθητές προτιμούν το	Λαϊκό
40 μαθητές προτιμούν το	Rock

15 μαθητές προτιμούν το	Metal
30 μαθητές προτιμούν το	Pop
25 μαθητές προτιμούν το	Hip hop
30 μαθητές δήλωσαν	άλλο

9.5 Σύντομα Θέματα για συζήτηση στην τάξη

Διαβάστε τα παρακάτω θέματα και αναπτύξτε στη τάξη τις προτάσεις σας. Τεκμηριώστε ανάλογα την άποψή σας.

Θέμα 1ο

Σε ένα «έξυπνο» τηλέφωνο πόσες κινήσεις χρειάζονται για να κάνουμε μία κλήση από το ευρετήριο; Σε σχέση με ένα «παραδοσιακό» κινητό τηλέφωνο είναι πιο σύνθετη ή πιο απλή διαδικασία; Τι θα προτείνατε για να βελτιώσετε τη διαδικασία αυτή;

Θέμα 2ο

Ποια εικονίδια πιστεύετε ότι είναι τα πιο κατάλληλα να χρησιμοποιήσουμε σε μια εφαρμογή για να δηλώσουμε στο χρήστη ότι με αυτά μπορεί αντίστοιχα: να αλλάξει τη γλώσσα γραφής από το πληκτρολόγιο, να εκτυπώσει ένα κείμενο, να ζητήσει βοήθεια για τη χρήση του, να αναζητήσει ένα στοιχείο, να διαγράψει ένα στοιχείο;

Θέμα 3ο

Στις υπηρεσίες μιας τράπεζας για την εξυπηρέτηση των πελατών της, προστίθεται ως νέα υπηρεσία η δυνατότητα πληρωμής λογαριασμών τηλεφώνων μέσω των ATM με αυτόματα χρέωση του λογαριασμού του ενδιαφερόμενου πελάτη. Λαμβάνοντας υπόψη σας τους βασικούς κανόνες του Shneiderman, που παρουσιάστηκαν στην ενότητα για το σχεδιασμό μιας διεπαφής, να αναπτυχθεί στη τάξη συζήτηση με βάση τα παρακάτω ερωτήματα:

- Ποια είναι τα βασικά στοιχεία που πρέπει να προσέξουν οι σχεδιαστές της εφαρμογής για τη νέα αυτή υπηρεσία της Τράπεζας, ώστε να μη γίνουν λάθη κατά τη διαδικασία πληρωμής;

- Πώς μπορούν να αποτραπούν λάθη κατά την είσοδο της ταυτότητας του λογαριασμού τηλεφώνου, που πρόκειται να πληρωθεί;
- Πώς ο πελάτης θα γνωρίζει ότι τελικά η διαδικασία πληρωμής ολοκληρώθηκε με επιτυχία;

9.6 Ερωτήσεις - Ασκήσεις

9.6.1 Ερωτήσεις

1. Τι είναι η επικοινωνία ανθρώπου υπολογιστή;
2. Τι είναι η διεπαφή χρήστη;
3. Τι είναι η ευχρηστία συστήματος και ποια βασικά χαρακτηριστικά έχει ένα σύστημα με αυξημένη ευχρηστία;
4. Ποιοι είναι οι βασικοί κανόνες για το σχεδιασμό μιας διεπαφής;
5. Τι είναι οι διεπαφές προγραμματισμού εφαρμογών – Application Programming Interfaces (APIs);
6. Σε τι μας χρησιμεύει η βιβλιοθήκη Tkinter της γλώσσας Python;
7. Πώς μπορούμε να εμφανίσουμε ένα απλό μήνυμα ανατροφοδότησης προς το χρήστη με χρήση της Tkinter στην γλώσσα Python;
8. Σε τι μας χρησιμεύουν τα RadioButtons;

Σύνοψη κεφαλαίου

Στην ενότητα αυτή αρχικά προσεγγίσαμε τις βασικές αρχές σχεδίασης διεπαφής. Στη συνέχεια είχαμε την ευκαιρία να υλοποιήσουμε απλά παραδείγματα ανάπτυξης διεπαφών, χρησιμοποιώντας τη βιβλιοθήκη Tkinter της γλώσσας Python. Βιβλιοθήκη που μας προσφέρει πολλές δυνατότητες για να κατασκευάσουμε γραφικές διεπαφές. Μέσα από τη δημιουργία μηνυμάτων ανατροφοδότησης, κουμπιών για την εκτέλεση κάποιων διεργασιών και πεδίων εισόδων για την εισαγωγή δεδομένων, διερευνήσαμε πώς να κάνουμε τα προγράμματά μας πιο φιλικά προς το χρήστη δίνοντας τη δυνατότητα αλληλεπίδρασης με το περιβάλλον των εφαρμογών μας.

10

Βάσεις Δεδομένων

ΕΣΩΤΕΡΙΚΗ ΕΓΚΛΟ. ΈΚΔΟΣΗ v201

10. Βάσεις δεδομένων

Εισαγωγή

Στο κεφάλαιο αυτό προσεγγίζονται θέματα που αφορούν στη δημιουργία και διαχείριση Βάσεων Δεδομένων μέσα από εντολές της γλώσσας Python.

Ένας συνηθισμένος τρόπος αποθήκευσης δεδομένων στον προγραμματισμό είναι αυτός που χρησιμοποιεί αρχεία. Τον τρόπο αυτό τον γνωρίσαμε τόσο στη Β' τάξη όσο και σε προηγούμενα κεφάλαια αυτού του βιβλίου. Ο τρόπος αυτός είναι ικανοποιητικός μόνο, όταν τα δεδομένα είναι σχετικά περιορισμένα σε μέγεθος, απλά και χρησιμοποιούνται από λίγους χρήστες. Όταν τα δεδομένα είναι πολύπλοκα και υπάρχουν πολλοί χρήστες, τότε η αποθήκευσή τους σε ένα μόνο αρχείο ή σε πολλά αρχεία παρουσιάζει πολλά προβλήματα, τα οποία λύνουν οι Βάσεις Δεδομένων.

Μια **Βάση Δεδομένων**-ΒΔ (Data Base) είναι μια συλλογή δεδομένων με υψηλό βαθμό οργάνωσης. Αναπόσπαστο τμήμα μιας βάσης δεδομένων αποτελεί ένα πακέτο προγραμμάτων λογισμικού που ονομάζεται *Σύστημα Διαχείρισης Βάσης Δεδομένων* – ΣΔΒΔ (Data Base Management System), το οποίο επιτρέπει τη δημιουργία της ΒΔ και τη διαχείριση των δεδομένων που περιέχει, κρύβοντας την πολυπλοκότητά της από το χρήστη.

Διδακτικοί στόχοι

Μετά τη μελέτη του κεφαλαίου θα μπορούμε να:

- περιγράψουμε τους βασικούς τύπους δεδομένων που χρησιμοποιεί η βιβλιοθήκη της Python, sqlite3, για τη δημιουργία και διαχείριση ΒΔ
- περιγράψουμε το Μοντέλο Δεδομένων και τα επίπεδά του
- χρησιμοποιούμε τις βασικές εντολές για τη δημιουργία μιας ΒΔ
- χρησιμοποιούμε τις βασικές εντολές για τη δημιουργία ενός πίνακα σε μια ΒΔ
- χρησιμοποιούμε τις βασικές εντολές για την εισαγωγή, τροποποίηση και διαγραφή εγγραφών σε ένα πίνακα

- χρησιμοποιούμε παραλλαγές της εντολής SELECT για την αναζήτηση στοιχείων σε μια ΒΔ.
- διαγράφουμε έναν πίνακα.

Λέξεις κλειδιά

Βάση δεδομένων, μοντέλο δεδομένων, σχεσιακό μοντέλο δεδομένων, sqlite3, εισαγωγή, τροποποίηση και διαγραφή δεδομένων, πίνακας.

Διδακτικές Ενότητες

10.1 Αναφορά στο Μοντέλο Δεδομένων

Κάθε ΒΔ ακολουθεί ένα σύστημα οργάνωσης και συσχέτισης των δεδομένων της, που ονομάζεται *μοντέλο δεδομένων*. Συνεπώς ένα μοντέλο δεδομένων καθορίζει τον τρόπο που οργανώνονται τα δεδομένα, καθώς επίσης και τον τρόπο με τον οποίο αυτά σχετίζονται μεταξύ τους. Ένα βασικό συστατικό ενός μοντέλου δεδομένων είναι ότι παρέχει ακρίβεια αναπαράστασης, δηλαδή τα στοιχεία και οι συσχετίσεις τους μπορούν να ερμηνευτούν με ένα και μοναδικό τρόπο. Επίσης το μοντέλο δεδομένων παρέχει το μέσο με το οποίο η ομάδα ανάπτυξης μιας ΒΔ, που ίσως αποτελείται από άτομα με διαφορετικές γνώσεις και ικανότητες, όπως για παράδειγμα από λογιστές, πρόσωπα της Διεύθυνσης, προγραμματιστές, να έχουν τη δυνατότητα να επικοινωνούν μεταξύ τους.

Στο παρόν κεφάλαιο θα ασχοληθούμε με τις **Σχεσιακές Βάσεις Δεδομένων** (Relational Data Base), οι οποίες βασίζονται στο αντίστοιχο σχεσιακό μοντέλο δεδομένων. Σύμφωνα με αυτό, τα δεδομένα οργανώνονται σε *πίνακες* (tables ή relations) οι οποίοι σχετίζονται μεταξύ τους με *συσχετίσεις* (relationships). Ένας πίνακας αποτελείται από γραμμές και στήλες, όπου τοποθετούμε τα στοιχεία σε οριζόντια και κάθετη μορφή. Κάθε γραμμή περιέχει τις πληροφορίες για ένα στοιχείο του πίνακα και αποκαλείται *πλειάδα* (tuple) ή *εγγραφή* (record). Κάθε στήλη χαρακτηρίζει κάποια ιδιότητα του πίνακα και αποκαλείται *χαρακτηριστικό* (attribute) ή *πεδίο* (field) κάθε εγγραφής.

Παράδειγμα

Πίνακας: Student

Χαρακτηριστικό ή πεδίο

Code	First.name	Surname	Birth	Phone
A234	Παύλος	Σιδηρόπουλος	11-03-96	6943456456
A345	Γεώργιος	Νταλάρας	08-05-96	6973456782
B456	Νικόλαος	Δήμου	02-02-96	6973245678
B654	Αμαλία	Καβάφη	18-07-96	6974356932

Πλειάδες-εγγραφές

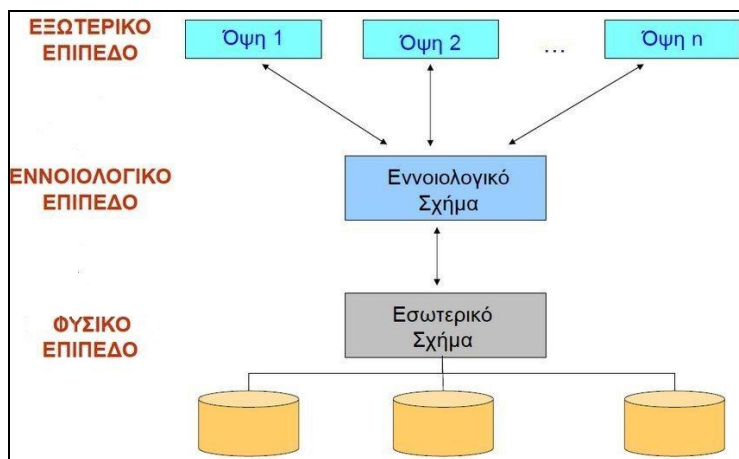
Κάθε μοντέλο δεδομένων έχει μια **αρχιτεκτονική τριών επιπέδων**. Τα επίπεδα αυτά είναι:

- Το Εξωτερικό επίπεδο (external level), η όψη της ΒΔ από τους χρήστες.
- Το Εννοιολογικό επίπεδο (conceptual level), η λογική όψη της ΒΔ.
- Το Φυσικό επίπεδο (internal level), η φυσική οργάνωση και αποθήκευση των δεδομένων).

Το **εξωτερικό** επίπεδο περιλαμβάνει όλες τις όψεις (views) της βάσης δεδομένων που έχουν οι διάφοροι χρήστες. Το εξωτερικό επίπεδο είναι ουσιαστικά ο τρόπος με τον οποίο βλέπει ο κάθε χρήστης τα περιεχόμενα της βάσης. Έτσι, για παράδειγμα, άλλες πληροφορίες βλέπει το λογιστήριο και άλλες η διαχείριση υλικών.

Το **εννοιολογικό** επίπεδο περιλαμβάνει τη συνολική λογική οργάνωση των δεδομένων της βάσης δεδομένων. Σε αυτό το επίπεδο περιγράφονται αναλυτικά τα δεδομένα καθώς και ο τρόπος με τον οποίο αυτά συσχετίζονται μεταξύ τους. Στο εννοιολογικό επίπεδο, του σχεσιακού μοντέλου το οποίο θα χρησιμοποιήσουμε, περιγράφονται όλοι οι πίνακες της βάσης δεδομένων καθώς και ο τρόπος με τον οποίο συσχετίζονται μεταξύ τους.

Το **φυσικό** επίπεδο περιλαμβάνει τη φυσική αποθήκευση των δεδομένων σε αρχεία πάνω σε φυσικά μέσα αποθήκευσης, όπως για παράδειγμα σε σκληρούς δίσκους.



Εικόνα 10-1. Αρχιτεκτονική τριών επιπέδων στις ΒΔ

10.2 Εισαγωγή στη διαχείριση Βάσεων Δεδομένων με προγραμματισμό

Έχει αναφερθεί ότι τα δεδομένα μιας Βάσης Δεδομένων αποθηκεύονται με κάποια μορφή αρχείων, σε φυσικά μέσα αποθήκευσης. Αυτό επιτρέπει σε γλώσσες τρίτης γενιάς να μπορούν να διαχειριστούν τα στοιχεία μιας σχεσιακής ΒΔ. Στην πράξη βέβαια, η διαχείριση της ΒΔ γίνεται μέσα από το ΣΔΒΔ.

Τη δεκαετία του 1980 έγινε προσπάθεια να δημιουργηθούν ειδικές γλώσσες προγραμματισμού οι οποίες να συνεργάζονται με βάσεις δεδομένων και να είναι εύκολο να τις χρησιμοποιήσουν και απλοί χρήστες, χωρίς ειδικές γνώσεις προγραμματισμού. Οι γλώσσες αυτές ονομάστηκαν γλώσσες τέταρτης γενιάς (4GL). Στο χειρισμό σχεσιακών βάσεων δεδομένων επικράτησε η SQL (Structure Query Language-Δομημένη γλώσσα ερωταποκρίσεων).

10.2.1 Η γλώσσα SQL

Η γλώσσα **SQL** ξεκίνησε από τα εργαστήρια της IBM και τυποποιήθηκε το 1987 (ANSI SQL). Οι διάφοροι κατασκευαστές ακολουθούν το πρότυπο αυτό, αλλά παράλληλα επεκτείνουν τις δυνατότητες της γλώσσας με την εισαγωγή νέων εντολών και συναρτήσεων. Η γλώσσα αυτή είναι προσανατολισμένη στη διαχείριση σχεσιακών βάσεων δεδομένων και όχι στον κλασικό

προγραμματισμό. Έτσι, δε διαθέτει δομές επιλογής τύπου AN ... TOTE (if ... then ...) ή δομές επανάληψης. Παρέχει όμως τη δυνατότητα πρόσβασης στα στοιχεία της Βάσης Δεδομένων, μπορεί να εκτελέσει αριθμητικές και λογικές πράξεις, να χειριστεί συμβολοσειρές (strings), καθώς και να παράγει αναφορές (εκθέσεις) σε οποιαδήποτε μορφή.

Η γλώσσα SQL περιέχει δύο μέρη:

- Τη **Γλώσσα Ορισμού Δεδομένων** (Data Definition Language -DDL), που χρησιμοποιείται για τη δημιουργία της δομής της ΒΔ.
- Τη **Γλώσσα Χειρισμού Δεδομένων** (Data Manipulation Language -DML), που χρησιμοποιείται για την εισαγωγή, διαγραφή, ενημέρωση, αναζήτηση και ταξινόμηση δεδομένων της βάσης δεδομένων.

Η SQL είναι εύκολη στην εκμάθηση και μπορεί να χρησιμοποιηθεί και από απλούς χρήστες μετά από εξάσκηση.

Για παράδειγμα η εντολή:

```
SELECT Name, Phone  
FROM customers  
WHERE City = "Λάρισα"
```

θα εμφανίσει το όνομα και το τηλέφωνο των πελατών που περιέχονται στον πίνακα customers και είναι από τη Λάρισα.

10.2.2 Η βιβλιοθήκη SQLite της Python

Η **SQLite** είναι ένα Σύστημα Διαχείρισης Βάσεων Δεδομένων συμβατό με το ISO SQL με χαρακτηριστικά όπως:

- Δε χρειάζεται καμία ρύθμιση.
- Αποθηκεύει μια Βάση Δεδομένων σε ένα μόνο αρχείο.
- Είναι εξαιρετικά γρήγορη, για λογικό όγκο εγγραφών.

Η SQLite είναι γραμμένη σε C και είναι το ευρύτερα διαδεδομένο ελεύθερο σύστημα ΣΔΒΔ. Ο κώδικας της SQLite είναι ελεύθερος για χρήση για οποιονδήποτε σκοπό (public domain), εμπορικό ή ιδιωτικό και υποστηρίζεται από μια μεγάλη κοινότητα.

Η SQLite υποστηρίζει πέντε τύπους δεδομένων:

- NULL - Αντιπροσωπεύει το "τίποτα" (όχι το μηδέν).
- INTEGER - Ακέραιος.

- REAL - Πραγματικός (δεκαδικοί αριθμοί).
- TEXT - Κείμενο.
- BLOB - χρησιμοποιείται για την αποθήκευση δυαδικών αρχείων κάθε τύπου (π.χ. φωτογραφίες).

Η SQLite μπορεί να χρησιμοποιηθεί ως αυτόνομο προϊόν, αλλά και μέσα από τη γλώσσα Python, όπως θα γνωρίσουμε στο πλαίσιο του κεφαλαίου αυτού. Η ενσωμάτωση των λειτουργιών της SQLite σε ένα πρόγραμμα Python γίνεται εισάγοντας στην αρχή του τη δήλωση:

```
import sqlite3
```

10.3 Δημιουργία ή σύνδεση με μια Βάση Δεδομένων στην Python

Η δημιουργία μιας νέας κενής βάσης δεδομένων σε ένα πρόγραμμα Python με χρήση της SQLite είναι πολύ απλή:

Αρχικά εισάγουμε τη βιβλιοθήκη SQLite

```
import sqlite3
```

Η δημιουργία ή σύνδεση με μια βάση δεδομένων επιτυγχάνεται με την εντολή:

```
conn = sqlite3.connect('example.db')
```

Η παραπάνω εντολή δημιουργεί ένα αντικείμενο *χειριστή* που θα συνδεθεί με το αρχείο που περιέχει τη βάση δεδομένων example.db. Αν δεν υπάρχει τέτοιο αρχείο (δεν υπάρχει η αναφερόμενη ΒΔ), τότε θα δημιουργηθεί μια νέα βάση δεδομένων με αυτό το όνομα. Η βάση θα δημιουργηθεί στο φάκελο που υπάρχει και η Python, ενώ αν επιθυμούμε να αποθηκευτεί σε κάποιο διαφορετικό φάκελο θα πρέπει να το δηλώσουμε μαζί με το όνομα της βάσης, όπως για παράδειγμα:

```
conn = sqlite3.connect('c:/SQL/example.db')
```

Αν θέλουμε να δημιουργήσουμε προσωρινά στην κύρια μνήμη τη βάση δεδομένων, τότε θα γράψουμε:

```
conn =sqlite3.connect('c:/SQL/example.db:memory')
```

Τέλος, στο όνομα του αρχείου της ΒΔ μπορούμε να δώσουμε όποια επέκταση θέλουμε (ή χωρίς επέκταση), αλλά καλό είναι να χρησιμοποιούμε είτε το .db είτε το .sql.

Μετά τη δημιουργία της *σύνδεσης* με τη ΒΔ, χρειαζόμαστε τη δημιουργία ενός αντικειμένου τύπου `cursor` μέσω του οποίου θα αλληλεπιδρούμε με τη ΒΔ, όπως για παράδειγμα αν θέλουμε να προσθέσουμε εγγραφές:

```
cursor = conn.cursor()
```

Μετά και τη δημιουργία του *αντικειμένου cursor*, μπορούμε να εκτελέσουμε μια εντολή SQL μέσω της εντολής `.execute()`. Τις εντολές SQL δεν έχει σημασία αν τις γράψουμε με πεζά ή κεφαλαία γράμματα, αλλά στο κεφάλαιο αυτό οι εντολές γράφονται με ΚΕΦΑΛΑΙΑ γράμματα για να ξεχωρίζουν.

Στο τέλος της εργασίας μας, θα πρέπει να θυμηθούμε να *κλείσουμε* τη σύνδεση καθώς και το αντικείμενο `cursor` με τις εντολές:

```
cursor.close()
```

```
conn.close()
```

10.4 Εισαγωγή, ενημέρωση και διαγραφή δεδομένων

Η βάση δεδομένων `example.db` που έχουμε δημιουργήσει, με τις εντολές της προηγούμενης παραγράφου, είναι κενή και δεν περιέχει δεδομένα. Για να εισαγάγουμε ορισμένα δεδομένα, ας βασιστούμε στο παράδειγμα μιας ΒΔ που χρησιμοποιείται από μια εταιρία μεταχειρισμένων αυτοκινήτων. Στη ΒΔ αυτή θέλουμε να δημιουργήσουμε έναν πίνακα που να περιέχει τα στοιχεία των αυτοκινήτων που διαθέτει η εταιρία.

Η δομή του πίνακα είναι:

Πεδίο	Περιγραφή	Τύπος
id	Κωδικός εγγραφής (αύξων αριθμός)	Ακέραιος
firma	Μάρκα (Εργ. κατασκευής) –μοντέλο	Κείμενο
cyear	Έτος κατασκευής	Κείμενο
fuel	Καύσιμο	Κείμενο
cc	Κυβισμός αυτοκινήτου	Ακέραιος
km	Χιλιόμετρα που έχει διανύσει	Ακέραιος
price	Τιμή	Πραγματικός

Η εντολή που θα χρησιμοποιήσουμε για τη δημιουργία του παραπάνω πίνακα είναι:

```
# create a table
curs.execute("""CREATE TABLE cars
              (id INTEGER PRIMARY KEY not null,
               firma TEXT,
               cyear TEXT,
               fuel TEXT,
               cc INTEGER,
               km INTEGER,
               price REAL)
              """);
```

Η εντολή αυτή θα δημιουργήσει μέσα στη ΒΔ example.db τον κενό πίνακα cars. Το πρώτο πεδίο αποτελεί το *πρωτεύον κλειδί* του πίνακα, το οποίο ορίστηκε ως ακέραιος αριθμός, αύξων αριθμός εγγραφής, πεδίο που θα μπορούσε να είχε δηλωθεί ως INTEGER PRIMARY KEY AUTOINCREMENT

Το *πρωτεύον κλειδί* (primary key) ενός πίνακα είναι ένα πεδίο ή ένας συνδυασμός πεδίων, που καθορίζει μονοσήμαντα κάθε εγγραφή του.

Η επόμενη λογική ενέργεια είναι να προσθέσουμε στοιχεία στον πίνακα που μόλις δημιουργήσαμε. Η εισαγωγή μιας εγγραφής επιτυγχάνεται με την εντολή **INSERT**:

```
# Insert a row of data
curs.execute("INSERT INTO cars VALUES (1, 'Audi
TT', '10/2010', 'Benzin', 2000, 24000, 23500)"),
```

Αν θέλουμε να εισάγουμε τιμές μόνο σε ορισμένα από τα πεδία, τότε θα πρέπει δίπλα στο όνομα του πίνακα να καθορίσουμε ποια πεδία θα είναι αυτά:

```
curs.execute("INSERT INTO cars (firma, fuel,
price) VALUES ('Audi A4', 'Diesel', 19000)")
```

Οι εισαγωγές που έχουμε πραγματοποιήσει, βρίσκονται ακόμα στη μνήμη του υπολογιστή και δεν έχουν καταχωρηθεί στο αρχείο (πίνακα) της βάσης δεδομένων. Για να καταχωρηθούν, θα πρέπει να εκτελέσουμε την παρακάτω εντολή:

```
# save data to database
conn.commit()
```

Η εικόνα του πίνακα cars μετά τις δύο εισαγωγές είναι η ακόλουθη:

Πίνακας 12-1. Ο πίνακας cars μετά από τις εισαγωγές

id	firma	cyear	fuel	cc	km	price
1	Audi TT	10/2010	Benzin	2000	24	23500.0
2	Audi A4	NULL	Diesel	NULL	NULL	19000.0

Παρατηρούμε ότι για τις τιμές που δεν έχουν οριστεί έχει εισαχθεί η τιμή NULL, εκτός από το πεδίο id. Ο τρόπος που έχει οριστεί το πεδίο id, επιτρέπει στη ΒΔ να συμπληρώνει με αύξοντες αριθμούς το πεδίο αυτό.

Μπορούμε να εισάγουμε πολλές εγγραφές μαζί, αν χρησιμοποιήσουμε μια λίστα σε συνδυασμό με την εντολή- μέθοδο `executemany()` :

```
# insert multiple records
autos = [(3, 'Audi A1', '07/2002', 'Benzin', 1300, 60000, 9000),
          (4, 'BMW 3', '02/2009', 'Diesel', 1800, 45000, 11000),
          (5, 'BMW 5', '04/2010', 'Benzin', 1800, 50000, 20000),
          (6, 'Citroen C1', '01/2012', 'Benzin', 998, 15000, 8000)]
curs.executemany("INSERT INTO cars VALUES
(?,?,?,?,?,?)", autos)
# save data to database
conn.commit()
```

Παρατηρήστε τη χρήση επτά ερωτηματικών “?” για τη διαδοχική αντικατάσταση των τιμών.

Πίνακας 12-2. Ο πίνακας cars μετά από τις πολλαπλές εισαγωγές

id	firma	cyear	fuel	cc	km	price
1	Audi TT	10/2010	Benzin	2000	24	23500.0
2	Audi A4	NULL	Diesel	NULL	NULL	19000.0
3	Audi A1	07/2002	Benzin	1300	60	9000.0
4	BMW 3	02/2009	Diesel	1800	45	11000.0
5	BMW 5	04/2010	Benzin	1800	50	20000.0
6	Citroen C1	01/2012	Benzin	998	15	8000.0

Τέλος, θα πρέπει να κλείσουμε τις συνδέσεις:

```
curs.close()
```

```
conn.close()
```

10.4.1 Ενημέρωση δεδομένων

Τα δεδομένα που περιέχει μια Βάση Δεδομένων συνεχώς αλλάζουν και για το λόγο αυτό, αν θέλουμε να αναφερθούμε σε μία συγκεκριμένη μορφή της, μιλάμε για στιγμιότυπο της Βάσης Δεδομένων. Η εντολή sql η οποία επιτρέπει την τροποποίηση των δεδομένων ενός πίνακα είναι η **UPDATE**.

Για παράδειγμα, θέλουμε στην εγγραφή με id =2 του πίνακα cars να εισαγάγουμε το μήνα και το έτος κατασκευής του αυτοκινήτου Audi A4 που είναι 05/2000. Το πρόγραμμα που θα χρησιμοποιήσουμε είναι:

```
# update
import sqlite3
conn = sqlite3.connect('example.db')
curs = conn.cursor()

curs.execute("""UPDATE cars
                SET cyear = '05/2000'
                WHERE id = 2
                """)

conn.commit()
```

Πίνακας 12-3. Ο πίνακας cars μετά από την τροποποίηση

id	firma	cyear	fuel	cc	km	price
1	Audi TT	10/2010	Benzin	2000	24	23500.0
2	Audi A4	05/2000	Diesel	NULL	NULL	19000.0
3	Audi A1	07/2007	Diesel	1200	50	8000.0

Αν θέλαμε να μειώσουμε όλες τις τιμές των αυτοκινήτων κατά 10%, η αντίστοιχη εντολή θα ήταν:

```
curs.execute("""UPDATE cars
                SET price = price * 0,9
                """)
```

Διαγραφή δεδομένων

Η διαγραφή δεδομένων είναι μια σχετικά εύκολη διαδικασία και για το λόγο αυτό χρειάζεται προσοχή, ώστε να μη διαγραφούν χρήσιμες πληροφορίες, κατά λάθος. Η διαγραφή δεδομένων από έναν πίνακα μέσω της sql, γίνεται με την εντολή **DELETE FROM**.

Παράδειγμα

Θέλουμε να διαγράψουμε από τον πίνακα cars όλα τα αυτοκίνητα που χρησιμοποιούν ως καύσιμο το πετρέλαιο (fuel = Diesel). Το αντίστοιχο πρόγραμμα είναι:

```
# delete rows
import sqlite3
conn = sqlite3.connect('example.db')
curs = conn.cursor()
curs.execute(""" DELETE FROM cars
                WHERE fuel = 'Diesel'
                """)
conn.commit()
```

Πίνακας 12-3. Ο πίνακας cars μετά από τις διαγραφές

id	firma	cyear	fuel	cc	km	price
1	Audi TT	10/2010	Benzin	2000	24	23500.0
3	Audi A1	07/2002	Benzin	1300	60	9000.0
5	BMW 5	04/2010	Benzin	1800	50	20000.0
6	Citroen C1	01/2012	Benzin	998	15	8000.0

10.4.2 Διαγραφή πίνακα

Εκτός από τη διαγραφή συγκεκριμένων δεδομένων, μπορούμε να διαγράψουμε ολόκληρο τον πίνακα με την εντολή **DROP TABLE** της sql, όπως φαίνεται στο ακόλουθο πρόγραμμα:

```
# Delete table
import sqlite3
conn = sqlite3.connect('example.db')
curs = conn.cursor()

curs.execute("DROP table cars")
conn.commit()
```

10.5 Αναζήτηση και ταξινόμηση δεδομένων

Η κυριότερη ίσως λειτουργία μιας ΒΔ, είναι η αναζήτηση δεδομένων που περιέχει. Για την αναζήτηση χρησιμοποιείται η εντολή **SELECT** της sql, η οποία έχει διάφορες μορφές, ανάλογα με το είδος της αναζήτησης. Τις μορφές αυτές θα γνωρίσουμε στη συνέχεια.

```
import sqlite3

conn = sqlite3.connect('example.db')
curs = conn.cursor()

# A listing of all the records in the table
curs.execute("SELECT * FROM cars")
print curs.fetchall()
```

```
# A listing of all benzin cars in the table
curs.execute("SELECT * FROM cars WHERE fuel =
'Benzin'")
print curs.fetchall()

print "A listing of all Audi cars"
for row in curs.execute("SELECT * FROM cars
WHERE firma LIKE 'Audi%'");
    print row
conn.commit()
```

Η πρώτη αναζήτηση χρησιμοποιεί το χαρακτήρα μπαλαντέρ "*", για να επιστρέψει όλα τα στοιχεία των εγγραφών από τον πίνακα cars. Οι εγγραφές εμφανίζονται με την εντολή print. Η εντολή αυτή συνδυάζεται με την fetchall(), που επιστρέφει όλες τις εγγραφές τις οποίες έχει ανακτήσει η προηγούμενη SELECT. Αντίστοιχη της fetchall() είναι η fetchone(), η οποία επιστρέφει μόνο μία εγγραφή. Αν θέλουμε την εμφάνιση μόνο συγκεκριμένων πεδίων, θα πρέπει να αναφέρουμε τα ονόματά τους ακριβώς. Για παράδειγμα:

```
curs.execute("SELECT firma, cyear, price FROM
cars")
```

Η δεύτερη αναζήτηση χρησιμοποιεί την επιλογή WHERE για να περιορίσει το εύρος της αναζήτησης. Στον περιορισμό μπορούν να χρησιμοποιηθούν και οι λογικοί τελεστές AND και OR. Για παράδειγμα:

```
curs.execute("SELECT * FROM cars
WHERE fuel = 'Benzin' AND price > 10000 ")
```

Η τρίτη αναζήτηση εισάγει ένα πολύ ενδιαφέρον όρισμα στην επιλογή WHERE, το όρισμα **LIKE**, το οποίο, σε συνδυασμό με χαρακτήρες μπαλαντέρ, χρησιμοποιείται για την αναζήτηση τμήματος αλφαριθμητικού. Στη συγκεκριμένη περίπτωση επιλέγονται οι εγγραφές στις οποίες το πεδίο firma αρχίζει με τους χαρακτήρες "Audi". Υπάρχουν πολλοί τρόποι χρήσης του ορίσματος LIKE, αλλά δεν θα αναφερθούν εδώ, γιατί ξεφεύγουν από το σκοπό του κεφαλαίου αυτού.

Αναζήτηση με ταξινόμηση

Σε μια αναζήτηση, η εμφάνιση των εγγραφών γίνεται με τη σειρά που αυτές είναι αποθηκευμένες στον πίνακα. Αυτός ο τρόπος εμφάνισης συχνά δεν μας ικανοποιεί, εφόσον θέλουμε τα αποτελέσματα να είναι ταξινομημένα. Στην περίπτωση αυτή, η εντολή `SELECT` συμπληρώνεται με το όρισμα **ORDER BY**.

Για παράδειγμα, στην πρώτη αναζήτηση όλων των στοιχείων του πίνακα `cars` επιθυμούμε τα αποτελέσματα να είναι ταξινομημένα κατά αύξουσα σειρά σε σχέση με την τιμή των αυτοκινήτων, δηλαδή τα φτηνότερα πρώτα. Η εντολή είναι:

```
curs.execute("SELECT * FROM cars ORDER BY price ASC")
```

Αξίζει να σημειωθεί ότι η αύξουσα ταξινόμηση `"ASC"` δεν χρειάζεται να δηλωθεί, σε αντίθεση με την φθίνουσα `"DESC"` η οποία πάντοτε δηλώνεται.

Η ταξινόμηση μπορεί να αναφέρεται σε δύο ή περισσότερα πεδία. Για παράδειγμα, στη δεύτερη αναζήτηση, των βενζινοκίνητων αυτοκινήτων, η εντολή θα μπορούσε να ήταν:

```
curs.execute("""SELECT * FROM cars
              WHERE fuel = 'Benzin'
              ORDER BY firma ASC, Price DESC""")
```

Σημείωση: Τα στοιχεία της ΒΔ που δημιουργήσαμε, μπορούμε να τα δούμε με χρήση άλλων προγραμμάτων. Μια εύκολη και δωρεάν λύση είναι το πρόσθετο *SQLite manager* που μπορούμε να εισαγάγουμε στο φυλλομετρητή ιστού Firefox. Ένα άλλο πρόγραμμα που μπορούμε να χρησιμοποιήσουμε, είναι το *SQLite inspector*. Οι εικόνες που έχουν χρησιμοποιηθεί στο κεφάλαιο, είναι από το πρόγραμμα *SQLiteManager* της *SQLABS*, το οποίο είναι εμπορικό προϊόν, αλλά με περιορισμένες δυνατότητες, διατίθεται ελεύθερα.

Τα τριπλά εισαγωγικά `" " "` και `"""` είναι ισοδύναμα και χρησιμοποιούνται, όταν μια παράσταση καταλαμβάνει περισσότερες από μία γραμμές.

Δραστηριότητες - Ερωτήσεις - Ασκήσεις κεφαλαίου

10.6 Δραστηριότητες

Δραστηριότητα 1. Επαναληπτική δραστηριότητα κεφαλαίου

Μελετήστε και εκτελέστε το παρακάτω πρόγραμμα, όπου δημιουργείται η Βάση Δεδομένων `example2.db` και ο πίνακας `users`, ο οποίος περιέχει το όνομα, το τηλέφωνο, το mail και το συνθηματικό των χρηστών ενός συστήματος.

```
# Εισαγωγή του module sqlite3
import sqlite3

# Δημιουργία ή άνοιγμα της βάσης 'example2.db'
conn = sqlite3.connect('example2.db')
# Δημιουργία του αντικειμένου cursor object
cursor = conn.cursor()
# Έλεγχος αν ο πίνακας δεν υπάρχει και
δημιουργία του
cursor.execute('''CREATE TABLE IF NOT EXISTS
    users(id INTEGER PRIMARY KEY,
    name TEXT,
    phone TEXT,
    email TEXT unique,
    password TEXT)''')
# Καταχώρηση αλλαγών στη βάση
conn.commit()
# Κλείσιμο της σύνδεσης conn
conn.close()

db = sqlite3.connect('example2.db')
cursor = db.cursor()
name1 = 'Panagioris'
phone1 = '2105666858'
email1 = 'panos@example.com'
```

```
password1 = '12345'

name2 = 'Ioannis'
phone2 = '2105657241'
email2 = 'john@example.com'
password2 = 'abcdef'

# Εισαγωγή χρήστη 1 με χρήση τεχνικής
υποκατάστασης με ερωτηματικά
cursor.execute(''INSERT INTO users(name, phone,
email, password)VALUES(?,?,?,?)'', (name1,
phone1, email1, password1))
print('First user inserted')

# Εισαγωγή χρήστη 2
cursor.execute(''INSERT INTO users(name, phone,
email, password)VALUES(?,?,?,?)'', (name2,
phone2, email2, password2))
print('Second user inserted')

db.commit()
```

Παρατήρηση: Στη δημιουργία του πίνακα, το πεδίο mail δηλώθηκε unique, δήλωση που επιβάλλει το κάθε mail που εισάγεται να είναι διαφορετικό.

10.7 Ερωτήσεις

1. Πώς εισάγονται εντολές της sqlite3 μέσα σε κώδικα Python;
2. Με ποια εντολή μπορούμε να δημιουργήσουμε μια κενή βάση δεδομένων με όνομα trade.db;
3. Με ποιες εντολές κώδικα μπορούμε να δημιουργήσουμε έναν πίνακα customers με στοιχεία το AFM, Name, City και Phone στη ΒΔ trade.db;

Οι επόμενες ερωτήσεις - ασκήσεις αναφέρονται στη ΒΔ trade.db και ειδικότερα στον πίνακα customers.

4. Με ποια εντολή μπορούμε να εισάγουμε στον πίνακα customers τον πελάτη με στοιχεία 012231212, Dimou Manolis, Athens, 212456789120;
5. Να εισαχθεί επιπλέον ο πελάτης 0123456789, Gatos Nikos, Athens, 212923876543
6. Να τροποποιηθεί η πόλη (city) του πελάτη Gatos σε Larisa.
7. Να διαγραφούν όλοι οι πελάτες που ως πόλη (city) έχουν την τιμή Athens.

10.8 Εργαστηριακές ασκήσεις

1. Δημιουργήστε μια βάση δεδομένων με όνομα school.db
2. Εισάγετε ένα πίνακα με όνομα students, ο οποίος θα περιέχει στοιχεία των μαθητών της τάξης σας. Το πλήθος και τον τύπο των στοιχείων που θα περιέχονται, θα το αποφασίσετε εσείς.
3. Εισάγετε τα στοιχεία ενός μαθητή και στη συνέχεια εισάγετε, όλα μαζί, τα στοιχεία 9 άλλων μαθητών.
4. Τροποποιήστε κάποιο από τα στοιχεία ενός μαθητή, όπως για παράδειγμα τη Διεύθυνση ή το τηλέφωνό του.
5. Διαγράψτε ένα μαθητή.
6. Εμφανίστε όλους τους μαθητές ταξινομημένους ως προς το επίθετό τους. Εμφανίστε ξανά όλους τους μαθητές ταξινομημένους ως προς το όνομά τους.

Σύνοψη

Στο κεφάλαιο αυτό γνωρίσαμε τη χρήση της βιβλιοθήκης sqlite3, η οποία επιτρέπει τη χρήση εντολών της SQL μέσα σε προγράμματα Python. Παρουσιάστηκε το σχεσιακό μοντέλο δεδομένων που χρησιμοποιείται ευρέως στις βάσεις δεδομένων και οι βασικές εντολές της sqlite3 για τη δημιουργία μιας ΒΔ. Στη συνέχεια παρουσιάστηκε η δημιουργία ενός ή περισσότερων πινάκων μέσα στη ΒΔ, καθώς και οι εντολές που είναι απαραίτητες για τη διαχείριση μιας βάσης δεδομένων. Οι εντολές αυτές είναι η INSERT, για την εισαγωγή δεδομένων σε πίνακα, η εντολή UPDATE, για την τροποποίηση στοιχείων ενός πίνακα και

η εντολή DELETE, για τη διαγραφή των στοιχείων ενός πίνακα. Παρουσιάστηκαν οι κυριότερες ίσως λειτουργίες σε μια ΒΔ, όπως η αναζήτηση και εμφάνιση στοιχείων της ΒΔ, λειτουργίες που βασίζονται σε παραλλαγές της εντολής SELECT.

Τέλος, θα πρέπει να σημειωθεί ότι τα παραδείγματα των Βάσεων Δεδομένων που χρησιμοποιήθηκαν στο κεφάλαιο, εξυπηρετούν εκπαιδευτικούς σκοπούς και δεν είναι οι βέλτιστες λύσεις.

ΕΣΩΤΕΡΙΚΗ ΕΓΚΕΚΡ. ΈΚΔΟΣΗ v201

ΕΣΩΤΕΡΙΚΗ ΕΓΚΕΚΡ. ΕΚΔΟΣΗ v201

11

Αντικειμενοστρεφής
προγραμματισμός

11. Αντικειμενοστρεφής Προγραμματισμός

Εισαγωγή

Στο κεφάλαιο αυτό αναπτύσσονται οι έννοιες που αφορούν τον αντικειμενοστρεφή προγραμματισμό.

Στη Β΄ τάξη ΕΠΑ.Λ. καθώς και στην αρχή του παρόντος διδακτικού υλικού, είχε γίνει μια σύντομη αναφορά στον αντικειμενοστρεφή προγραμματισμό με παραδείγματα. Στο κεφάλαιο αυτό θα γίνει εμβάθυνση στις σχετικές έννοιες και εξάσκηση στη δημιουργία κλάσεων και αντικειμένων με χρήση της Python.

Διδακτικοί στόχοι

Μετά τη μελέτη του κεφαλαίου θα μπορούμε να:

- αναγνωρίζουμε και να κατονομάζουμε χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού
- περιγράφουμε τις διαφορές του αντικειμενοστρεφούς προγραμματισμού σε σχέση με τον δομημένο προγραμματισμό
- δημιουργούμε απλά αντικείμενα και κλάσεις
- διακρίνουμε την έννοια της κλάσης από εκείνη του αντικειμένου
- αναγνωρίζουμε το ρόλο της κληρονομικότητας και του πολυμορφισμού στη δημιουργία επαναχρησιμοποιήσιμου κώδικα.

Λέξεις κλειδιά

Αντικειμενοστρεφής προγραμματισμός, κλάση, αντικείμενο, μέθοδος, κληρονομικότητα, πολυμορφισμός, προγραμματισμός οδηγούμενος από γεγονότα.

11.1 Αντικείμενα και Κλάσεις

Πριν την εμφάνιση του αντικειμενοστρεφούς προγραμματισμού, τα προγράμματα δομούνταν πάνω στην ιδέα της προτεραιότητας των ενεργειών έναντι των εννοιών/δεδομένων/οντοτήτων. Έτσι, περιγράφαμε διαδικασίες και συναρτήσεις που επεξεργάζονταν και τροποποιούσαν δεδομένα, τις οποίες θεωρούσαμε δομικά στοιχεία των προγραμμάτων μας. Οι εφαρμογές ακολουθούσαν το

Διαδικαστικό προγραμματισμό (procedural programming) και αποτελούνταν από ακολουθίες εντολών, επαναλήψεις, δομές ελέγχου και υποπρογράμματα, προσέγγιση ιδιαίτερα λειτουργική στις περισσότερες των περιπτώσεων. Όμως, όταν κανείς ασχολιόταν με βάσεις δεδομένων ή με την Τεχνητή Νοημοσύνη, γινόταν φανερό ότι εξυπηρετούσε καλύτερα μια προσέγγιση βασισμένη στην προτεραιότητα των εννοιών-οντοτήτων έναντι των ενεργειών/διεργασιών. Επίσης, όσο οι εφαρμογές με βάση το διαδικαστικό προγραμματισμό γίνονταν μεγαλύτερες και πιο περίπλοκες, η παρακολούθηση, η διόρθωση και η συντήρησή τους γινόταν δυσκολότερη.

Τι είναι λοιπόν ο Αντικειμενοστρεφής προγραμματισμός και πού εξυπηρετεί;

Ο Αντικειμενοστρεφής προγραμματισμός αλλάζει την εστίαση του προγραμματισμού από τις διαδικασίες στις έννοιες. Σε αυτές αναθέτει χαρακτηριστικά, τα οποία ονομάζουμε ιδιότητες (attributes), και τα οποία επεξεργάζεται μέσω ειδικών συναρτήσεων που ονομάζουμε μεθόδους. Οι ιδιότητες δεν είναι άλλες από τα σχετικά με το αντικείμενο δεδομένα, και οι μέθοδοι αποτελούν περιγραφές ενεργειών που αφορούν συγκεκριμένα σε αυτές τις ιδιότητες. Με τα θέματα αυτά θα ασχοληθούμε περισσότερο αργότερα.

Έχει ενδιαφέρον να υπενθυμίσουμε ότι η Python, που είδαμε ότι πολύ καλά μπορεί να εξυπηρετήσει ένα διαδικαστικού τύπου προγραμματισμό, είναι στη βάση της μια αντικειμενοστρεφής γλώσσα. Έτσι, οι βασικές βιβλιοθήκες της έχουν δομηθεί με αντικειμενοστρεφή λογική και όλα στην Python αποτελούν στιγμιότυπα μιας κλάσης ή ενός τύπου δεδομένων. Μάλιστα και οι τύποι δεδομένων λειτουργούν στην ουσία σαν κλάσεις.

Εισαγωγή στις κλάσεις και τα αντικείμενα

Τι είναι ένα αντικείμενο στον αντικειμενοστρεφή προγραμματισμό; Πολύ απλά είναι ένα αντικείμενο της πραγματικής ζωής που το περιγράφουμε με χρήση κώδικα. Για παράδειγμα ένα αντικείμενο, που γνωρίζουμε όλοι από την πραγματική ζωή, είναι ένα αυτοκίνητο. Πώς όμως θα περιγράφαμε το αυτοκίνητο που έχουμε στην οικογένειά μας ή ανήκει σε κάποιον άλλον, αν μας το

ζητούσαν; Θα αναφέραμε προφανώς τα χαρακτηριστικά που διαθέτει, όπως για παράδειγμα τη μάρκα του, το μοντέλο, το έτος κατασκευής του, το χρώμα του, τα χιλιόμετρα που έχει κάνει, το πόσο κοστίζει. Όλα αυτά τα χαρακτηριστικά αναφέρονται συγκεκριμένο αυτοκίνητο, ενώ υπάρχουν και χαρακτηριστικά, όπως ο αριθμός πλαισίου, που το χαρακτηρίζουν με μοναδικό τρόπο. Στον αντικειμενοστρεφή προγραμματισμό, τα **χαρακτηριστικά** (attributes) ενός αντικειμένου ονομάζονται και **ιδιότητες** (properties).

Μια επόμενη ερώτηση θα μπορούσε να είναι, τι μπορούμε να κάνουμε με ένα αυτοκίνητο;

Μπορούμε:

- Να το ξεκινήσουμε.
- Να το επιταχύνουμε
- Να φρενάροουμε (να το επιβραδύνουμε) κλπ.

Στον Αντικειμενοστρεφή προγραμματισμό, αυτές οι ενέργειες ονομάζονται **μέθοδοι (methods)**. Οι μέθοδοι επιτρέπουν στα αντικείμενα να κάνουν διάφορες ενέργειες, μέσω των οποίων, μπορούμε να ελέγχουμε τις ιδιότητες του αντικειμένου, όπως για παράδειγμα να ελέγχουμε την ταχύτητα του αυτοκινήτου.

Δημιουργία αντικειμένων

Τώρα που έχουμε διευκρινίσει τις ιδιότητες, τις μεθόδους και τρόπους για να ρυθμίσουμε ή να αλλάξουμε τις ιδιότητες μέσω μεθόδων, το ερώτημα είναι: πώς δημιουργούμε τελικά ένα αντικείμενο στον αντικειμενοστρεφή προγραμματισμό;

Η απάντηση είναι ότι ένα αντικείμενο δημιουργείται από μια ειδική μέθοδο που στην αντικειμενοστρεφή ορολογία ονομάζεται **κατασκευαστής** (constructor). Στην πραγματική ζωή, ο κατασκευαστής θα ήταν η κατασκευαστική γραμμή του εργοστασίου που θα παρήγαγε το αυτοκίνητο. Σε αυτή τη διαδικασία κατασκευής ορισμένα από τα χαρακτηριστικά του αυτοκινήτου θα είχαν συγκεκριμένες τιμές. Για παράδειγμα: η μάρκα θα ήταν TheCar, το μοντέλο A20, τα χιλιόμετρα μηδέν(0) κλπ. Η αντίστοιχη διαδικασία στον προγραμματισμό, δηλαδή η απόδοση αρχικών τιμών σε ιδιότητες ενός αντικειμένου που δημιουργείται, ονομάζεται **αρχικοποίηση** (initialization). Όπως

είμαστε σίγουροι ότι η συγκεκριμένη κατασκευαστική γραμμή θα παράγει πάντα αυτοκίνητα TheCar, τύπου A20 με μηδέν χιλιόμετρα. Έτσι και στον προγραμματισμό, όταν ένας κατασκευαστής παράγει ένα αντικείμενο, γνωρίζουμε ότι ορισμένες από τις ιδιότητές του θα έχουν συγκεκριμένες τιμές. Στα αυτοκίνητα γνωρίζουμε ότι μπορούμε να παραγγείλουμε ένα συγκεκριμένο μοντέλο, το οποίο στη στάνταρ έκδοση έχει συγκεκριμένα χαρακτηριστικά, μπορούμε όμως παράλληλα να επιλέξουμε ορισμένα από τα χαρακτηριστικά του, όπως για παράδειγμα τις ζάντες του. Άρα, ορισμένες από τις ιδιότητες ενός αντικείμενου μπορούν να καθοριστούν αργότερα από τη βασική κατασκευή του.

Φυσικά, αφού υπάρχει μια ειδική μέθοδος η οποία κατασκευάζει αντικείμενα, η οποία παρεμπιπτόντως δεσμεύει μνήμη στον υπολογιστή, θα πρέπει να υπάρχει και μια μέθοδος η οποία να καταστρέφει τα αντικείμενα αυτά και να απελευθερώνει την αντίστοιχη μνήμη. Η μέθοδος αυτή ονομάζεται **αποδομητής** (destructor).

Μέχρι τώρα, με τον όρο αυτοκίνητο σκεφτόμαστε συνήθως ένα επιβατικό, κλειστό αυτοκίνητο (sedan). Η εταιρία όμως κατασκευάζει εκτός από τα επιβατικά, και φορτηγά και λεωφορεία. Τι γίνεται, αν θέλουμε να περιγράψουμε και αυτά; Μια λύση βέβαια θα ήταν, να περιγράψουμε κάθε είδος ως ξεχωριστό αντικείμενο. Θα διαπιστώναμε όμως, ότι όλα αυτά τα αντικείμενα έχουν κοινές ιδιότητες, για παράδειγμα όλα έχουν τροχούς και όλα έχουν κοινές μεθόδους, ότι όλα επιταχύνουν και φρενάρουν. Δε θα ήταν λογικό και αποδοτικό να κάνουμε την ίδια εργασία για κάθε ένα αντικείμενο χωριστά.

Μια λύση που χρησιμοποιεί ο Αντικειμενοστρεφής προγραμματισμός είναι η δημιουργία ενός *προτύπου* με όνομα *όχημα*, το οποίο έχει όλες τις ιδιότητες και τις μεθόδους που είναι κοινές μεταξύ των επιβατικών των φορτηγών και των λεωφορείων. Αυτά τα πρότυπα που συγκεντρώνουν τα κοινά στοιχεία άλλων αντικειμένων, έχουν ένα ειδικό όνομα και ονομάζονται **κλάσεις**

(classes). Οι κλάσεις μας επιτρέπουν να γνωρίζουμε πράγματα για αντικείμενα που ακόμα δεν έχουν δημιουργηθεί.

Πολλές προσεγγίσεις του αντικειμενοστρεφούς προγραμματισμού αρχίζουν από τις κλάσεις τις οποίες προσομοιάζουν με το σχέδιο ενός αυτοκινήτου ή το αρχιτεκτονικό σχέδιο ενός σπιτιού. Εφαρμόζοντας αυτό το σχέδιο, μπορούμε να δημιουργήσουμε αντικείμενα, για παράδειγμα αυτοκίνητα ή σπίτια, που έχουν κοινά στοιχεία, αλλά διαφέρουν και σε πολλά. Στις κλάσεις ενσωματώνονται και οι κατασκευαστές των αντικειμένων.

Για να κατανοήσουμε καλύτερα τη διαφορά μεταξύ κλάσεων και αντικειμένων, θα χρησιμοποιήσουμε το αγαπημένο παράδειγμα στους προγραμματιστές, της τάρτας. Σε αυτό η κλάση τάρτα είναι μια συνταγή για τάρτες. Αν την έχεις, μπορείς να φτιάξεις τάρτες, όμως, όσο έχεις μόνο την συνταγή, δεν έχεις τάρτα για να τη φας. Το αντικείμενο τάρτα είναι μια «πραγματική» τάρτα, μια υλοποίηση της συνταγής με συγκεκριμένα χαρακτηριστικά, ενώ μπορούμε να έχουμε πολλά διαφορετικά αντικείμενα τάρτες κατασκευασμένα με την ίδια συνταγή.

Αν ξαναγυρίσουμε στο παράδειγμά μας, έχουμε μια κλάση που ονομάζεται όχημα και περιέχει τις κοινές ιδιότητες και μεθόδους των τριών τύπων οχημάτων. Αυτή η κλάση, ονομάζεται *κλάση γονέας* (parent class) και οι τρεις τύποι των οχημάτων αποτελούν τις υπο-κλάσεις (subclasses). Είναι σημαντικό να προσέξουμε ότι οι κοινές ιδιότητες και μέθοδοι ορίζονται μόνο στην κλάση γονέα. Οι υποκλάσεις (κλάσεις παιδιά) αυτόματα, έχουν τις ίδιες ιδιότητες και μεθόδους, χωρίς να είναι απαραίτητο να τις ορίσουμε ξανά. Αυτή, η μεταβίβαση ιδιοτήτων και μεθόδων ονομάζεται **κληρονομικότητα** (inheritance). Σε κάθε υποκλάση ορίζονται οι ιδιότητες και οι μέθοδοι που δεν καλύπτονται από την πατρική κλάση.

Επίσης παρατηρούμε ότι τα αντικείμενα που ανήκουν στις τρεις υποκλάσεις κάνουν όλα το ίδιο, μεταφέρουν άτομα και είδη, αλλά με ένα διαφορετικό τρόπο, φαινόμενο που ονομάζεται **πολυμορφισμός** (polymorphism).

Ορισμός κλάσης στην Python

Ο ορισμός μιας κλάσης στην Python είναι παρόμοιος με τον αυτόν μιας συνάρτησης, που γνωρίσαμε στα προηγούμενα κεφάλαια. Μόνο που, αντί για τη λέξη `def`, χρησιμοποιούμε τη λέξη κλειδί **`class`**. Για παράδειγμα, αν θέλουμε να ορίσουμε την κλάση όχημα με ορισμένες ιδιότητες και μεθόδους, από αυτές που αναφέραμε παραπάνω, η σύνταξη είναι:

`class` `Vehicle`:

```
def __init__( self, color, price, wheels, speed):  
    self.color=color  
    self.price=price  
    self.wheels=wheels  
    self.speed=speed  
  
def accelerate(self, amount) :  
    self.speed += amount  
    return self.speed  
  
def decelerate(self, amount) :  
    self.speed -= amount  
    return self.speed
```

Παρατηρούμε ότι οι ιδιότητες της κλάσης εισάγονται σε μια ιδιαίτερη συνάρτηση την `__init__` ως παράμετροι. Η συγκεκριμένη έχει μια ιδιαιτερότητα στο όνομα, αφού αρχίζει και τελειώνει με διπλές κάτω παύλες `__`. Για τη μεταβλητή `self` που εμφανίζεται στην κλάση, θα μιλήσουμε αργότερα. Τέλος, οι μέθοδοι της κλάσης, έχουν τη μορφή συναρτήσεων, τοποθετούνται όμως, μέσα στο σώμα της κλάσης.

Ας δούμε πιο αναλυτικά τι σημαίνουν τα παραπάνω.

11.2 Στιγμιότυπα (αυτόματη αρχικοποίηση αντικειμένων)

Στην προηγούμενη παράγραφο ορίσαμε την κλάση `Vehicle` η οποία περιγράφει την έννοια του οχήματος. Η ύπαρξη της κλάσης δε σημαίνει και την αυτόματη ύπαρξη ενός αντικειμένου αυτής της κλάσης. Σημαίνει μόνο ότι υπάρχει απλώς η περιγραφή με την οποία μπορούμε να δημιουργήσουμε τέτοιο/α αντικείμενο/α.

Το αρχικό τμήμα κώδικα της κλάσης:

```
def __init__( self, color, price, wheels, speed):  
    self.color=color  
    self.price=price  
    self.wheels=wheels  
    self.speed=speed
```

είναι η μέθοδος που αποτελεί τον κατασκευαστή των αντικειμένων και καλείται επίσης *μέθοδος αρχικοποίησης τιμών*. Παρατηρούμε ότι δίνει τιμές στις ιδιότητες του αντικειμένου που ορίζονται στις παραμέτρους της `__init__`. Κανονικά, μπορούμε να ορίσουμε εμείς τα ονόματα των μεθόδων, αλλά με το όνομα `__init__`, η γλώσσα αντιλαμβάνεται ότι είναι μια μέθοδος κατασκευαστής και έτσι καλείται αυτόματα με κάθε νέα δημιουργία αντικειμένου, τύπου `Vehicle`.

Για να δημιουργήσουμε ένα τέτοιο αντικείμενο πρέπει να καλέσουμε την κλάση `Vehicle`, με ορίσματα ή τιμές για κάθε μία από τις ιδιότητες του αντικειμένου που αντιστοιχούν στις παραμέτρους της μεθόδου `__init__`. Στο αντικείμενο που δημιουργείται δίνουμε ένα όνομα, όπως για παράδειγμα:

```
mybeetle = Vehicle('yellow', 2000.00, 4, 80).
```

Έχουμε τώρα δημιουργήσει ένα όχημα κίτρινο, με τιμή 2000 ευρώ, τέσσερις τροχούς και ταχύτητα 80 χιλιόμετρα την ώρα. Το όνομά του είναι mybeetle. Το mybeetle είναι ένα αντικείμενο, ανήκει στην κλάση όχημα (`Vehicle`) και έχει συγκεκριμένες τιμές για τις ιδιότητες του οχήματος, αυτές που ορίσαμε με την κλήση της. Ένα τέτοιο αντικείμενο ονομάζεται **στιγμιότυπο** της κλάσης.

Δεν καλέσαμε ονομαστικά τη μέθοδο `__init__`, απλώς καλέσαμε την κλάση σχεδόν ως συνάρτηση, με τις τιμές που αφορούν το αντικείμενο που θέλουμε να δημιουργήσουμε για τις ιδιότητες της κλάσης, όπως αυτές εμφανίζονται ως μεταβλητές της μεθόδου.

Η χρήση της δεσμευμένης λέξης `self`

Η δεσμευμένη λέξη `self` εμφανίζεται ως πρώτη παράμετρος σε όλες τις μεθόδους της κλάσης (δείτε στην προηγούμενη παράγραφο). Αυτή η παράμετρος επιτρέπει στη μέθοδο να αναφέρεται στο ίδιο το αντικείμενο και όχι σε ολόκληρη την κλάση, πράγμα που τουλάχιστον ισχύει στη γενική περίπτωση. Τοποθετώντας στη `self` το όνομα του αντικειμένου, διασφαλίζουμε ότι μια μέθοδος που θα κληθεί μέσω ενός αντικειμένου, θα επιδράσει μόνο σε αυτό.

Συγκεκριμένα, με μια κλήση, όπως η `mybeetle.accelerate(10)`, ενεργοποιούμε τις οδηγίες της μεθόδου `accelerate` για το αντικείμενο `mybeetle`. Σε αυτή την περίπτωση η `def accelerate(self, amount)` αντιλαμβάνεται ως `self` το `mybeetle`.

Άρα, όταν γράφουμε `def accelerate(self, amount)` και μετά δίνουμε οδηγίες, αυτό που εννοούμε είναι “ορίστε πώς θα εφαρμόσεις την `accelerate`, σε όποιο αντικείμενο της κλάσης `Vehicle` την καλέσει”. Και υλοποιούμε με αυτόν τον τρόπο την αύξηση ταχύτητας στο συγκεκριμένο αντικείμενο, στην περίπτωση μας, στο `mybeetle` μου.

Η ιδιαιτερότητα της μεθόδου `__init__`

Ας επιστρέψουμε τώρα στο λόγο που δεν καλούμε άμεσα την ίδια την μέθοδο, αλλά, αυτή καλείται έμμεσα μέσα από κλήση της κλάσης. Ο λόγος γι' αυτό είναι ότι στο πρόγραμμά μας σπάνια, γι' αυτό, θα θελήσουμε να αναφερθούμε σε οποιαδήποτε ιδιότητα ή μέθοδο της κλάσης γενικά. Έτσι, όταν αναφερόμαστε στην κλάση, πέρα από τον ορισμό της, θέλουμε να είμαστε βέβαιοι ότι έχουμε δημιουργήσει ένα αντικείμενο στο οποίο θα εφαρμόσουμε μεθόδους της κλάσης πάνω στις ιδιότητές του. Οπότε, όταν καλούμε την κλάση, έχουμε φροντίσει να καλείται η μέθοδος-συνάρτηση αρχικοποίησης `__init__` η οποία κατασκευάζει αντικείμενα, λόγος για τον οποίο η μέθοδος `__init__` ονομάζεται κατασκευαστής.

Φυσικά, με βάση τα παραπάνω, η χρήση της `self` στην `__init__` έχει λιγότερο νόημα από ότι σε άλλες μεθόδους. Στις υπόλοιπες αναφέρεται στο αντικείμενο μέσα από το οποίο καλείται, ενώ εδώ αναφέρεται σε ένα αντικείμενο που κατασκευάζεται τώρα. Όμως,

αυτή η μικρή αβαρία από πλευράς λογικής κάνει τη ζωή μας πιο εύκολη.

Σημαντικό είναι όταν ορίζουμε κατασκευαστές, να θυμόμαστε ότι κάθε ιδιότητα που χρησιμοποιείται από άλλες μεθόδους, πρέπει να αρχικοποιείται, πριν χρησιμοποιηθεί. Ο πιο εύκολος τρόπος για την αποφυγή λαθών είναι, να ορίζονται όλες μέσα στην `__init__`, καθώς δεν υπάρχει τρόπος, να πούμε στο πρόγραμμα να τρέξει δύο μεθόδους για αρχικοποίηση. Ας δούμε ένα παράδειγμα προβληματικά ορισμένης κλάσης:

class `Vehicle1`:

```
def __init__( self, color, price, wheels):
```

```
    self.color=color
```

```
    self.price=price
```

```
    self.wheels=wheels
```

```
def set_speed(self, speed = 100)
```

```
    self.speed=speed
```

```
def accelerate(self, amount) :
```

```
    self.speed += amount
```

```
    return self.speed
```

```
def decelerate(self, amount) :
```

```
    self.speed -= amount
```

```
    return self.speed
```

Στην κλάση `Vehicle1` αρχικοποιούνται στην `__init__` όλες οι ιδιότητες, εκτός από την ταχύτητα (`speed`) που αρχικοποιείται στην `set_speed`. Το προβληματικό εδώ είναι ότι οι ακόλουθες δύο γραμμές κώδικα οδηγούν σε πρόβλημα, αφού γίνεται απόπειρα να μεταβληθεί η ταχύτητα που δεν έχει αρχικοποιηθεί:

```
betty=Vehicle1('yellow', 2000.00, 4)
```

```
betty.accelerate(10)
```

Επομένως, η αρχικοποίηση ιδιοτήτων έξω από την `__init__` οδηγεί εύκολα σε λογικά λάθη. Φυσικά, αυτό δεν μας εμποδίζει να ορίσουμε με τέτοιο τρόπο την κλάση μας, απλά απαιτεί ιδιαίτερη

προσοχή στην αρχικοποίηση. Ας δούμε πώς χειριζόμαστε μια τέτοια περίπτωση, στο επόμενο παράδειγμα.

Αρχικοποίηση ιδιοτήτων αντικειμένων

Όπως έχουμε ήδη αναφέρει, τις περισσότερες φορές θέλουμε οι ιδιότητες διαφορετικών αντικειμένων να έχουν διαφορετικές τιμές. Ας δούμε ένα πιο ολοκληρωμένο παράδειγμα.

Θα δημιουργήσουμε μια κλάση με όνομα `dog` που θα έχει τις ιδιότητες `breed` (ράτσα), `size` (μέγεθος), `color` (χρώμα) και τις μεθόδους `eat` (τρώω) και `bark` (γαυγίζω). Στη συνέχεια θα δημιουργήσουμε δύο στιγμιότυπα της κλάσης, αρχικοποιώντας τις ιδιότητές τους και θα καλέσουμε τις μεθόδους τους:

```
class Dog:
```

```
    def __init__(self, breed, size, color):
        self.breed=breed
        self.size = size
        self.color = color

        print "breed:",self.breed,"size:",self.size,"color:",self.color

    def eat(self,food):
        print "I am eating ", food

    def bark(self):
        print "I am barking "
```

```
Max = Dog("terrier", "medium", "brown")
```

```
Rocky = Dog( "labrador", "large", "light brown")
```

```
Max.eat("bone")
```

```
Rocky.eat("meat")
```

Όπως εξηγήσαμε και νωρίτερα η δήλωση `self.breed = breed` αποθηκεύει την τιμή της παραμέτρου **breed**, ως τιμή της αντίστοιχης ιδιότητας του αντικειμένου που κάλεσε την `__init__`. Η κατασκευή των αντικειμένων μας γίνεται εδώ στις δύο πρώτες γραμμές μετά τον ορισμό της κλάσης. Η αρχικοποίησή τους όμως τελειώνει εκεί που τελειώνει ο κώδικας που γράψαμε! Δεν πρέπει να παρεμβάλλουμε κάποια εντολή που να αφορά ένα από τα δύο

αυτά στιγμιότυπα ανάμεσα στις τέσσερις τελευταίες γραμμές του παραδείγματός μας.

Δραστηριότητα εμπέδωσης παραγράφου

Δίνεται η παρακάτω κλάση:

class Car:

```
def __init__(self, make):  
    self.make=make  
    self.speed = 60  
def speed_up(self,speed):  
    self.speed = speed  
    print "I am driving at a speed", self.speed, "km/h"  
def turn(self):  
    print "I am turning ... "
```

1. Ποιος είναι ο κατασκευαστής (constructor) της κλάσης;
2. Να καταγράψετε τις ιδιότητες της κλάσης, τις μεθόδους της, καθώς και τις ενέργειες που πραγματοποιούν.
3. Να προσθέσετε τις ιδιότητες color και year που αντιπροσωπεύουν το χρώμα και το έτος κυκλοφορίας του αυτοκινήτου αντίστοιχα και να αρχικοποιούνται στον κατασκευαστή.
4. Να αλλάξετε τη μέθοδο turn, έτσι ώστε να δέχεται ως παράμετρο μια συμβολοσειρά που ορίζει αν το αυτοκίνητο θα στρίψει αριστερά ή δεξιά.
5. Να δημιουργήσετε τα παρακάτω στιγμιότυπα της κλάσης:
 - I. Αντικείμενο με όνομα convertible και μάρκα "bmw", χρώμα "μαύρο" και έτος κυκλοφορίας "2013".
 - II. Αντικείμενο με όνομα sedan και μάρκα "toyota", χρώμα "κόκκινο" και έτος κυκλοφορίας "2009".
6. Να καλέσετε την κατάλληλη μέθοδο, ώστε το αντικείμενο convertible να στρίψει δεξιά.
7. Να καλέσετε την κατάλληλη μέθοδο, ώστε το αντικείμενο sedan να τρέχει με 90 χιλ/ώρα.

Πρόσβαση σε ιδιότητες αντικειμένων

Προκειμένου να αποκτήσουμε πρόσβαση στις ιδιότητες των αντικειμένων που δημιουργήσαμε, χρησιμοποιούμε το λεγόμενο *dot notation*. Δηλαδή:

όνομα_αντικειμένου.ιδιότητα

Παράδειγμα

```
class C:                                #δημιουργούμε την κλάση C
    classattr = "attr on class"

...
>>> cobj = C()                        #δημιουργούμε το αντικείμενο cobj
>>> cobj.instattr = "attr on instance"
>>>
>>> cobj.instattr
'attr on instance'
>>> cobj.classattr
'attr on class'
```

Η έκφραση **cobj.classattr** σημαίνει “ Πήγαινε στο αντικείμενο στο οποίο αναφέρεται η μεταβλητή cobj και πάρε την τιμή της ιδιότητας **classattr**”.

Ορισμός μεθόδων

Οι μέθοδοι είναι σημασιολογικά ίδιες με τις συναρτήσεις, αλλά υπάρχουν δύο *συντακτικές* διαφορές:

- Οι μέθοδοι ορίζονται μέσα στον ορισμό μιας κλάσης, προκειμένου να γίνει σαφής η σχέση μεταξύ της κλάσης και της μεθόδου.
- Η σύνταξη για την κλήση μιας μεθόδου είναι διαφορετική από τη σύνταξη για την κλήση μιας συνάρτησης

Ως **υπογραφή** ορίζεται το όνομα, τα ορίσματα και η τιμή επιστροφής της μεθόδου.

Οι μέθοδοι ορίζονται μέσα στην κλάση, χρησιμοποιώντας την ίδια σύνταξη που χρησιμοποιούμε, για να ορίσουμε μια συνάρτηση (βλ. κεφάλαιο 7). Αυτό με τη διαφορά ότι πάντα η πρώτη

παράμετρος πρέπει να είναι το στιγμιότυπο (αντικείμενο) της κλάσης του οποίου η μέθοδος καλείται. Επειδή το αντικείμενο που καλεί δεν είναι προκαταβολικά γνωστό, χρησιμοποιούμε στη θέση του μια παράμετρο την οποία ονομάζουμε πάντα με το όνομα **self**. Με αυτόν τον τρόπο, η Python γνωρίζει σε ποιο αντικείμενο ανήκει η μέθοδος που καλείται και ποίου αντικειμένου τις ιδιότητες αλλάζει.

Παράδειγμα

class C:

```
    classattr = "attr on class"
```

```
    def my_print(self):
```

```
        print "my attribute is ", self.classattr
```

Πρόσβαση σε μεθόδους

Προκειμένου να αποκτήσουμε πρόσβαση στις μεθόδους των αντικειμένων που δημιουργήσαμε, χρησιμοποιούμε το λεγόμενο *dot notation*. Δηλαδή:

```
    όνομα_αντικειμένου.μέθοδος()
```

Είναι φανερό ότι οι μέθοδοι αυτές ορίζονται στην κλάση, αλλά όταν καλούνται αφορούν πάντα ένα αντικείμενο ή στιγμιότυπο.

Παράδειγμα

Δημιουργούμε ένα αντικείμενο της κλάσης που ορίσαμε στο προηγούμενο παράδειγμα:

```
>>> cobj = C ()
```

Στη συνέχεια, καλούμε τη μέθοδο `my_print` του αντικειμένου ως εξής:

```
>>> cobj.my_print()
```

```
"my attribute is attr on class"
```

Μέθοδοι που δεν αφορούν στιγμιότυπα

Μια ιδιαίτερη περίπτωση μεθόδου είναι οι λεγόμενες **στατικές μέθοδοι**. Στο παράδειγμα της κλάσης σκύλος (**Dog**) είχαμε τη μέθοδο `bark()`.

```
class Dog:
```

```
...
def bark(self):
    print "I am barking "
```

Ανεξάρτητα με το ποιο στιγμιότυπο καλεί τη συνάρτηση, το αποτέλεσμα θα είναι το ίδιο έτσι όπως την έχουμε ορίσει, μια δήλωση του εκάστοτε σκύλου ότι γαβγίζει. Σε αυτή την περίπτωση η παράμετρος `self` είναι περιττή. Για να το κάνουμε αυτό σαφές έχουμε ένα ειδικό διακριτικό, το `@staticmethod`, που προηγείται της συνάρτησης. Έτσι ο ορισμός γίνεται:

```
class Dog:
```

```
...
    @staticmethod
    def bark():
        print "I am barking "
```

Μια μέθοδος με το διακριτικό `@staticmethod` είναι στην πράξη μια κλασική συνάρτηση την οποία όμως έχουμε τοποθετήσει μέσα σε μια κλάση. Σε περίπτωση μιας υποκλάσης που κληρονομεί την κλάση (δες επόμενη παράγραφο), ο ορισμός δεν επηρεάζεται και μένει ακριβώς ο ίδιος που ορίστηκε στην κλάση. Βέβαια, εφόσον χρησιμοποιήσουμε το διακριτικό `@staticmethod`, δεν χρησιμοποιούμε το αντικείμενο ως παράμετρο.

Μπορούμε επίσης, να χρησιμοποιήσουμε μια *μέθοδο κλάσης*, μέσω του διακριτικού `@classmethod`. Όσο δεν ασχολούμαστε με υποκλάσεις και κληρονομικότητα, μια μέθοδος κλάσης λειτουργεί όπως μια στατική μέθοδος. Όμως μια μέθοδος κλάσης είναι κληρονομήσιμη και άρα, αν κληθεί από μια υποκλάση, θα αφορά αυτή. Για τον ορισμό μιας μεθόδου κλάσης περνάμε ως πρώτη παράμετρο την κλάση, αντί του αντικειμένου, με τη δεσμευμένη λέξη `cls` που εννοεί την κλάση που την καλεί.

Περισσότερο νόημα έχει η χρήση μεθόδων κλάσης όταν χρησιμοποιούμε κληρονομικότητα (δες επόμενη παράγραφο), με μια συνηθισμένη χρήση να είναι η παρακολούθηση του αριθμού των στιγμιότυπων της κλάσης.

Παράδειγμα

```
class Dog:
    no_inst = 0
    def __init__(self, breed, size, color):
        ...
        Dog.no_inst = Dog.no_inst + 1
    ...
    @classmethod
    def get_no_of_dogs(cls):
        return cls.no_inst
```

11.4 Χαρακτηριστικά: Κληρονομικότητα, Πολυμορφισμός

11.4.1 Κληρονομικότητα (Inheritance)

Το πιο σημαντικό χαρακτηριστικό του Αντικειμενοστρεφούς Προγραμματισμού είναι η Κληρονομικότητα (Inheritance).

Κληρονομικότητα ονομάζεται η ιδιότητα των κλάσεων να μεταβιβάζουν τις ιδιότητές τους σε νέες κλάσεις, που λέγονται κληρονόμοι ή *υποκλάσεις*. Οι υποκλάσεις μπορούν να επαναχρησιμοποιήσουν τις μεταβιβάσιμες μεθόδους και ιδιότητες της κλάσης από την οποία κληρονομούν, αλλά και να προσθέσουν δικές τους.

Χρησιμοποιώντας λίγο πιο επίσημη ορολογία, λέμε ότι η κλάση που ορίζεται πρώτη είναι η *βασική κλάση* (base class) και η κλάση που κληρονομεί τη βασική κλάση είναι η *παράγωγη κλάση* (derived class). Εναλλακτικά, οι βασικές κλάσεις ονομάζονται και *υπερκλάσεις*, ενώ οι παράγωγες κλάσεις ονομάζονται *υποκλάσεις*.

Στιγμιότυπα των υποκλάσεων μπορούν να χρησιμοποιηθούν όπου απαιτούνται στιγμιότυπα των υπερκλάσεων, εφόσον η υποκλάση είναι, κατά κάποιον τρόπο, μια πιο εξειδικευμένη εκδοχή της υπερκλάσης, αλλά το αντίστροφο δεν ισχύει.

Παράδειγμα κληρονομικότητας, είναι μια υπερκλάση Vehicle (=Όχημα) και οι δύο πιο εξειδικευμένες υποκλάσεις της Car (=Αυτοκίνητο) και Bicycle (=Ποδήλατο), οι οποίες λέμε ότι "κληρονομούν" από αυτήν.

Από μια υποκλάση μπορούν να προκύψουν νέες υποκλάσεις που κληρονομούν από αυτήν, με αποτέλεσμα μια ιεραρχία κλάσεων που συνδέονται μεταξύ τους, "ανά γενιά", με σχέσεις κληρονομικότητας. Για παράδειγμα, δυο υποκλάσεις της κλάσης Car (Αυτοκίνητο), είναι η Convertible (ανοικτό) και η Sedan (Κλειστού τύπου).

Γενικότερα, η *Κληρονομικότητα χρησιμοποιείται όταν έχουμε μια ιεραρχία αντικειμένων τα οποία πηγαίνουν από το γενικότερο στο ειδικότερο*. Χαρακτηριστικό παράδειγμα, είναι το προηγούμενο που αναφέραμε (Όχημα $\Rightarrow$ Αυτοκίνητο $\Rightarrow$ Κλειστού τύπου).

Αν μελετήσουμε τις σχέσεις μεταξύ των διαδοχικών κλάσεων, θα δούμε ότι η "κάθε υποκλάση **είναι** η υπερκλάση της επόμενης". Δηλαδή:

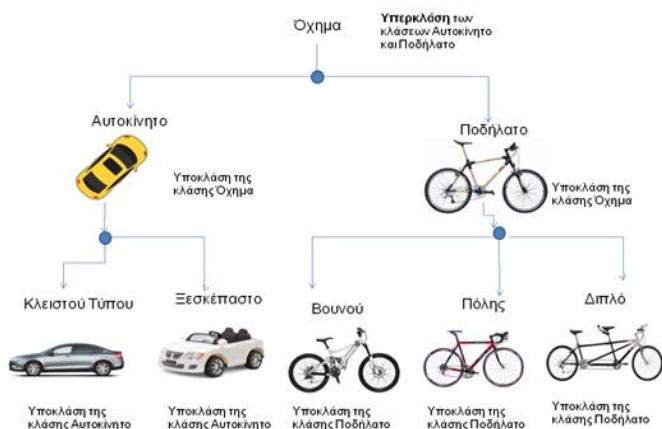
- Ένα αυτοκίνητο κλειστού τύπου είναι αυτοκίνητο.
- Ένα αυτοκίνητο είναι όχημα.

Η σχέση αυτή ισχύει όχι μόνο για την αμέσως ανώτερη υπερκλάση, αλλά για κάθε υπερκλάση. Δηλαδή:

- Ένα αυτοκίνητο κλειστού τύπου είναι αυτοκίνητο.
- Ένα αυτοκίνητο κλειστού τύπου είναι και όχημα.

Κάθε φορά λοιπόν που έχουμε μια παρόμοια σχέση μεταξύ αντικειμένων, δηλαδή μπορούμε να πούμε ότι *κάτι είναι κάτι άλλο* (is-a relationship), τότε μπορούμε/πρέπει να χρησιμοποιήσουμε κληρονομικότητα.

Με αυτόν τον τρόπο δημιουργείται μια ιεραρχία, που για το παράδειγμα του οχήματος, θα μπορούσε να είναι αυτή που δίνεται στο παρακάτω σχήμα.



Τα πλεονεκτήματα από τη χρήση της κληρονομικότητας φαίνονται, αν παρατηρήσουμε τις υποκλάσεις μιας κλάσης. Στο παράδειγμα που αναφέραμε παραπάνω, οι υποκλάσεις *αυτοκίνητο* και *ποδήλατο* έχουν κάποια κοινά χαρακτηριστικά. Ορίζουμε αρχικά τα κοινά χαρακτηριστικά αυτών των κλάσεων στην υπερκλάση *Όχημα*, τα κληρονομούμε και τα εξειδικεύουμε καταλλήλως στις υποκλάσεις.

Με τον ίδιο τρόπο, αν στο πρόγραμμά μας χρειαζόμαστε να αναπαραστήσουμε *Ψάρια*, *Έντομα* και *Θηλαστικά*, τότε μπορούμε τα κοινά χαρακτηριστικά τους να τα ορίσουμε στην υπερκλάση *Ζώο*, από την οποία θα τα κληρονομούν. Στη συνέχεια βέβαια, η κάθε μία από τις υποκλάσεις *Ψάρι*, *Έντομο* και *Θηλαστικό* θα έχει το δικό της ιεραρχικό δέντρο, στο οποίο θα εξειδικεύεται καταλλήλως.

Με αυτόν τον τρόπο επιτυγχάνεται η επαναχρησιμοποίηση κώδικα που εξοικονομεί χρόνο, χρήμα και αυξάνει την αξιοπιστία του προγράμματος.

Ορισμός υποκλάσης στην Python

Για να ορίσουμε μια υποκλάση στην Python γράφουμε:

```
class ΌνομαΥποκλάσης(ΌνομαΥπερκλάσης):
```

όπου *ΌνομαΥποκλάσης* είναι το όνομα της υποκλάσης που θέλουμε να ορίσουμε και *ΌνομαΥπερκλάσης*, είναι το όνομα της υπερκλάσης από την οποία κληρονομεί.

Παράδειγμα 1

```
class BaseClass(object):  
    def my_print(self):  
        print "Hello "
```

```
class InheritingClass(BaseClass):  
    pass
```

Στο παραπάνω παράδειγμα ορίσαμε την υπερκλάση `BaseClass`, η οποία βλέπουμε ότι κληρονομεί από την έτοιμη (built in) κλάση `object` της Python. Στη συνέχεια, ορίσαμε την υποκλάση `InheritingClass`, η οποία κληρονομεί από την κλάση `BaseClass`.

Αν θέλουμε να δημιουργήσουμε ένα στιγμιότυπο της κλάσης `InheritingClass` και να καλέσουμε τη μέθοδο `my_print ()`, αυτό υλοποιείται ως εξής:

```
>>>x = InheritingClass ()  
>>>x.my_print()  
Hello
```

Βλέπουμε λοιπόν, ότι η υποκλάση μπορεί να χρησιμοποιεί όλες τις ιδιότητες και μεθόδους της υπερκλάσης.

Γιατί όμως είναι απαραίτητο η κλάση `BaseClass` να κληρονομεί από την `object`; Διότι με αυτόν τον τρόπο επιτρέπουμε στην υποκλάση `InheritingClass` να καλεί μεθόδους, όπως για παράδειγμα την `__init__` της κλάσης `BaseClass`. Αυτό θα γίνει καλύτερα κατανοητό με το παρακάτω παράδειγμα:

Παράδειγμα 2

Ορίζουμε την παρακάτω κλάση:

```
class Person(object):  
    def __init__(self, name):  
        self._name= name
```

Στη συνέχεια ορίζουμε την υποκλάση `Student` ως εξής:

```
class Student(Person):  
    def __init__(self, name, classatt):  
        super(Student, self).__init__(name)  
        self.classatt = classatt
```

Μέσα στον κατασκευαστή `__init__` της κλάσης `Student` χρησιμοποιούμε τη δήλωση:

`super(Student, self).__init__(name)`, για να καλέσουμε τον κατασκευαστή της υπερκλάσης `Person`. Αυτό μπορούμε να το κάνουμε ακριβώς, επειδή η κλάση `Person` κληρονομεί από την κλάση `object`.

Γι' αυτό είναι καλή πρακτική, όταν ορίζουμε μια κλάση που δεν είναι υποκλάση κάποιας άλλης, να κληρονομεί πάντα από την κλάση `object`.

Δραστηριότητα εμπέδωσης ενότητας

- Ποιες είναι οι ιδιότητες της κλάσης `Student`;
- Δημιουργήστε στιγμιότυπο της κλάσης `Student` με όνομα "Γιάννης" και τάξη "B".
- Πώς θα εμφανίσετε τις ιδιότητες του στιγμιότυπου που δημιουργήσατε στο προηγούμενο ερώτημα;
- Να ορίσετε υποκλάση `Teacher` της κλάσης `Person`, η οποία εκτός από το όνομα, θα έχει επιπλέον τις ιδιότητες κλάδος (field) και χρόνια υπηρεσίας (years).
- Δημιουργήστε τα παρακάτω στιγμιότυπα της κλάσης `Teacher`:
 - Στιγμιότυπο με όνομα "Αναστασίου", κλάδο "ΠΕ02" και χρόνια υπηρεσίας "11".
 - Στιγμιότυπο με όνομα "Παπαχρήστου", κλάδο "ΠΕ03" και χρόνια υπηρεσίας "16".

11.4.2 Πολυμορφισμός (polymorphism)

Στον αντικειμενοστρεφή προγραμματισμό, ο πολυμορφισμός αναφέρεται στη δυνατότητα των αντικειμενοστρεφών μεταγλωττιστών/διερμηνευτών να αποφασίζουν δυναμικά ποια, από της ομώνυμες μεθόδους, είναι κατάλληλη να κληθεί ανάλογα με το αντικείμενο που την καλεί ή τον τύπο και τον αριθμό των παραμέτρων της.

11.4.3 Ενθυλάκωση και Απόκρυψη δεδομένων

Ενθυλάκωση (encapsulation) καλείται η ιδιότητα που προσφέρουν οι κλάσεις, να «κρύβουν» τα ιδιωτικά δεδομένα τους από το υπόλοιπο πρόγραμμα και να εξασφαλίζουν ότι, μόνο μέσω των δημόσιων μεθόδων τους, θα μπορούν αυτά να προσπελαστούν.

Μπορούμε να πούμε ότι η ενθυλάκωση έχει δύο χαρακτηριστικά:

1. Συνδυάζει δεδομένα και μεθόδους σε ένα θύλακα (capsule).

Η κλάση ορίζεται ως ένας τύπος δεδομένων και περιλαμβάνει:

- Μεταβλητές ή πεδία τα οποία αντιπροσωπεύουν διάφορες ιδιότητες.
- Μεθόδους οι οποίες χρησιμοποιούνται για να κατασκευάσουν στιγμιότυπα της κλάσης (κατασκευαστές - constructors) και μεθόδους οι οποίες χρησιμοποιούνται για να αναπαραστήσουν τις λειτουργίες της κλάσης.

2. Ελέγχει την πρόσβαση στα δεδομένα (information hiding).

Η κλάση, όπως την ορίζουμε, εμφανίζει στους χρήστες και στις άλλες κλάσεις μια διεπαφή (interface), η οποία παρουσιάζει τις λειτουργίες της κλάσης. Με αυτό τον τρόπο, κρύβονται πληροφορίες χαμηλού επιπέδου για τον τρόπο υλοποίησης των λειτουργιών.

Έτσι, μπορούμε να χρησιμοποιούμε τα αντικείμενα, χωρίς να χρειάζεται να καταλαβαίνουμε τον τρόπο που έχουν υλοποιηθεί.

Παράδειγμα

Έχουμε έναν οδηγό αυτοκινήτου. Κάθε οδηγός μπορεί να οδηγεί το αυτοκίνητο, χωρίς να χρειάζεται να ξέρει πώς λειτουργεί η μηχανή του. Είναι αρκετό να γνωρίζει πώς ξεκινάει το αυτοκίνητο, πώς να βάζει βενζίνη, πώς να φρενάρει, πώς να επιταχύνει και πώς να σταματάει τη μηχανή. Αυτές οι βασικές λειτουργίες αποτελούν τη *διεπαφή* (interface) του αυτοκινήτου. Μπορούμε να σκεφτούμε τη διεπαφή ως το σύνολο των ενεργειών που έχουμε τη δυνατότητα να κάνουμε στο αυτοκίνητο, χωρίς να χρειάζεται να γνωρίσουμε πώς αυτά υλοποιούνται.

Κρύβοντας την πολυπλοκότητα υλοποίησης του αυτοκινήτου από το χρήστη επιτρέπουμε σε οποιονδήποτε, ανεξάρτητα από το αν είναι μηχανικός αυτοκινήτων ή όχι, να το οδηγήσει. Κατά τον ίδιο τρόπο, αποκρύπτοντας την υλοποίηση των λειτουργιών ενός αντικειμένου από το χρήστη, επιτρέπουμε σε οποιονδήποτε να το χρησιμοποιήσει ανεξαρτήτως από το αν γνωρίζει ή όχι τον τρόπο υλοποίησής του.

Ένα άλλο παράδειγμα ενθουλάκωσης, είναι η χρήση του τελεστή `in` στην Python. Θεωρήστε το παρακάτω τμήμα προγράμματος σε Python:

```
if c in "συμβολοσειρά":
```

```
    print "βρέθηκε"
```

```
else:
```

```
    print "δε βρέθηκε"
```

Ο παραπάνω κώδικας θα εμφανίσει το μήνυμα " βρέθηκε", αν ο χαρακτήρας `c` βρεθεί μέσα στη συμβολοσειρά "συμβολοσειρά", αλλιώς θα εμφανίσει το μήνυμα "δε βρέθηκε". Όμως, με ποιο τρόπο η Python υλοποιεί τη λειτουργία του τελεστή `in`; Ποιον αλγόριθμο αναζήτησης χρησιμοποιεί; Αυτό εμείς δε χρειάζεται να το ξέρουμε, γιατί η ακριβής υλοποίηση της λειτουργικότητας του τελεστή αυτού είναι κρυμμένη για τους χρήστες του τελεστή. Αυτή η απόκρυψη πληροφοριών είναι βασική στον αντικειμενοστρεφή προγραμματισμό και στη γλώσσα Python.

Πλεονεκτήματα

Η χρήση της ενθυλάκωσης έχει δύο βασικά πλεονεκτήματα:

1. Επιτρέπει Έλεγχο

- Η χρήση του αντικειμένου γίνεται μόνο μέσα από τις μεθόδους του.
- Δεν επιτρέπεται η άμεση αλλαγή των δεδομένων του αντικειμένου.

Με αυτόν τον τρόπο δεν επιτρέπεται σε άλλα αντικείμενα να αλλάξουν άμεσα τα δεδομένα ενός άλλου αντικειμένου.

2. Επιτρέπει αλλαγές

- Η χρήση του αντικειμένου δεν αλλάζει, αν αλλάξουν τα δεδομένα.
- Πρέπει όμως, να παραμείνουν τα ίδια πρότυπα μεθόδων.

Για να επανέλθουμε στο παράδειγμα του αυτοκινήτου που χρησιμοποιήσαμε παραπάνω, ο οδηγός μπορεί να οδηγήσει το αυτοκίνητο είτε είναι βενζινοκίνητο, είτε πετρελαιοκίνητο, είτε, αν κινείται με αέριο, χωρίς κανένα πρόβλημα. Αυτό συμβαίνει διότι τα αυτοκίνητα αυτά έχουν διαφορετικό εσωτερικό μηχανισμό λειτουργίας, κρυμμένο για τον οδηγό. Αυτό σημαίνει ότι, ακόμα κι αν ο μηχανισμός λειτουργίας του αυτοκινήτου αλλάξει, το αυτοκίνητο λειτουργεί με τον ίδιο τρόπο από την πλευρά του οδηγού.

Τροποποιητές πρόσβασης (Access Modifiers)

Η Ενθυλάκωση στον αντικειμενοστρεφή προγραμματισμό επιτυγχάνεται με τους **τροποποιητές πρόσβασης**.

Οι βασικοί τροποποιητές πρόσβασης είναι οι εξής:

- **public:** Η πρόσβαση στην κλάση, δεδομένα, μεθόδους είναι ανοικτή σε όλους
- **private:** Η πρόσβαση στα δεδομένα και τις μεθόδους είναι περιορισμένη μόνο στην κλάση
- **protected:** η πρόσβαση είναι ανοικτή σε κλάσεις που κληρονομούν από την ίδια κλάση.

Διαφορετικές γλώσσες προγραμματισμού μπορεί να προσφέρουν και άλλους τροποποιητές πρόσβασης πέρα από τους βασικούς. Για παράδειγμα η Java έχει 4 , η C++ 3 και η C# 5.

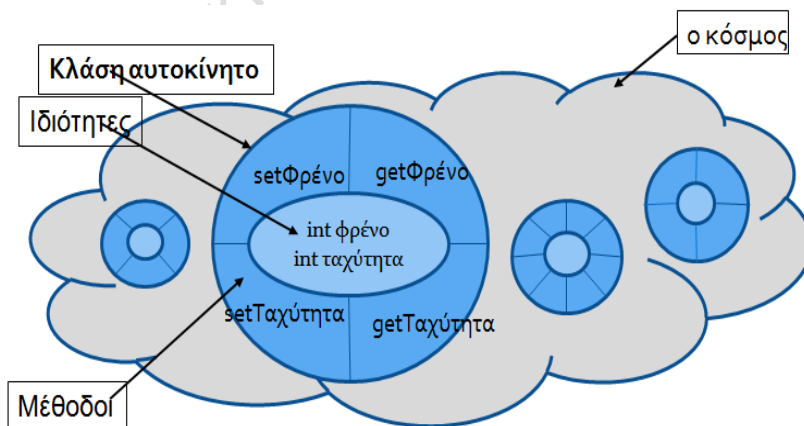
Μια κλάση ελέγχει την πρόσβαση στα διάφορα μέλη της (δεδομένα, μεθόδους) με τους τροποποιητές πρόσβασης.

Οι βασικοί κανόνες σύμφωνα με τους οποίους χρησιμοποιούνται οι παραπάνω τροποποιητές πρόσβασης σε μια κλάση είναι οι εξής:

- Τα μέλη μιας κλάσης, όπως οι υψηλού επιπέδου μέθοδοι, ορίζονται `public` για να εκθέσουν τις λειτουργίες τους σε χρήστες/κλάσεις.
- Τα μέλη μιας κλάσης που παρέχουν λεπτομέρειες υλοποίησης, ορίζονται `private` για να κρύψουν πληροφορίες από τους χρήστες/κλάσεις.

Πώς όμως αποκτούμε πρόσβαση στις `private` πληροφορίες μιας κλάσης; Συνήθως χρησιμοποιούμε ειδικές μεθόδους που τις ονομάζουμε `get()` και `set()`. Οι μέθοδοι αυτές επιστρέφουν τις τιμές των `private` ιδιοτήτων και τους δίνουν τιμή αντίστοιχα.

Αν πάρουμε πάλι το παράδειγμα του αυτοκινήτου, μπορούμε να πούμε ότι οι ιδιότητες του αυτοκινήτου, όπως για παράδειγμα φρένο ή ταχύτητα, ορίζονται ως `private` πληροφορίες και ότι χρησιμοποιούνται ειδικές μέθοδοι για την πρόσβαση στις πληροφορίες αυτές, όπως φαίνεται και από το παρακάτω σχήμα.



Οι περισσότερες αντικειμενοστρεφείς γλώσσες, όπως Java ή C++, χρησιμοποιούν τους παραπάνω τροποποιητές πρόσβασης άμεσα για να ορίσουν περιορισμούς πρόσβασης στις ιδιότητες και στις μεθόδους τους.

Στην Python όμως, αυτό μπορεί να οριστεί μόνο με σύμβαση. Για παράδειγμα τα ονόματα όλων των `private` μεθόδων και ιδιοτήτων ξεκινούν με διπλή κάτω παύλα.

11.5 Οδήγηση από Γεγονότα - Μηνύματα

Έχουμε ήδη αναφέρει τη σημασία που έχουν τα γεγονότα στον αντικειμενοστρεφή προγραμματισμό. Είναι ο τρόπος με τον οποίο καθορίζεται η συμπεριφορά των αντικειμένων: Μια συμπεριφορά καθορίζεται από ένα γεγονός, εξωτερικό του αντικειμένου, αλλά επίσης από ένα χειριστή του γεγονότος, δηλαδή από ένα κομμάτι κώδικα -στην Python συνάρτηση- μέσα στο αντικείμενο που υλοποιεί την ανταπόκριση του αντικειμένου στο γεγονός.

Υπάρχει μια πληθώρα πιθανών *γεννητριών* γεγονότων. Ξεκινά από το ίδιο το σύστημα, τα μέρη του και ιδιαίτερα τις μονάδες εισόδου εξόδου, από εφαρμογές που τρέχουν πάνω σε αυτό και επικοινωνούν με τη δική μας, από άλλα αντικείμενα της εφαρμογής μας. Ένα γεγονός προκαλεί ένα μήνυμα, ένα σύνολο συναφών με το γεγονός δεδομένων, που έχει σκοπό να προκαλέσει κάποια προγραμματισμένη αντίδραση. Φυσικά ένα μήνυμα -ειδικά όταν ανταλλάσσεται μεταξύ εφαρμογών ή *scripts* ή ακόμα και αντικειμένων- μπορεί να είναι το ίδιο ένα γεγονός.

Τα πιο γνωστά μας γεγονότα είναι αυτά που έχουν σχέση με το ποντίκι και το πληκτρολόγιο. Για παράδειγμα, το κλικ στο αριστερό κουμπί του ποντικιού, το πάτημα του πλήκτρου `return`, το πάτημα του πλήκτρου «G», αποτελούν, τουλάχιστον εν δυνάμει, γεγονότα. Για να έχουν ενδιαφέρον για την εφαρμογή μας βέβαια, θα πρέπει να υπάρχει κάποια προγραμματισμένη ανταπόκριση σε αυτά.

Ένα κρίσιμο ζήτημα είναι πώς εξασφαλίζουμε ότι το σωστό αντικείμενο θα λάβει το μήνυμα που σχετίζεται με ένα γεγονός που το αφορά. Υπάρχουν διάφοροι τρόποι διαχείρισης αυτού του θέματος, όπως:

Ένας τρόπος διαχείρισης των μηνυμάτων είναι να έχουμε χειριστές (κατάλληλες μεθόδους) σε διάφορα αντικείμενα, χωρίς κεντρικό έλεγχο, ή τουλάχιστον χωρίς εμφανή κεντρικό έλεγχο, με τα αντικείμενα να αποφασίζουν ποια μηνύματα τα αφορούν. Σε μια παραλλαγή της προηγούμενης περίπτωσης μπορεί να έχουμε ένα μοντέλο πελάτη/εξυπηρετητή, όπου ο πελάτης ζητά από τον εξυπηρετητή κάποια ενέργεια και αυτός ανταποκρίνεται, αν κρίνει ότι τον αφορά. Ένα καλό παράδειγμα είναι η περιήγηση στο Διαδίκτυο, όπου κάθε φορά που αλλάζουμε σελίδα ζητάμε μια εξυπηρέτηση από έναν διαδικτυακό εξυπηρετητή και αυτός ανταποκρίνεται στέλνοντάς μας τη σελίδα που ζητήσαμε, εφόσον κρίνει ότι έχουμε δικαίωμα πρόσβασης σε αυτή.

Ένας άλλος τρόπος διαχείρισης είναι, να έχουμε κεντρικό έλεγχο, ο οποίος δρομολογεί τα μηνύματα στους κατάλληλους χειριστές. Οι χειριστές αυτοί σε έναν καλό αντικειμενοστρεφή σχεδιασμό θα ανήκουν, βέβαια, σε διάφορα αντικείμενα. Η δρομολόγηση γίνεται μέσω ενός *διανεμητή* (dispatcher), στον οποίο κατευθύνονται τα μηνύματα που προκλήθηκαν από τα γεγονότα και ο οποίος γνωρίζει από πριν ποια αντικείμενα αφορά κάθε γεγονός και τα ενημερώνει κατάλληλα.

Μια ειδική περίπτωση κεντρικού ελέγχου είναι αυτή του μοντέλου *Παρατηρητή ή εγγραφής/ δημοσίευσης*, επίσης γνωστή και με το όνομα *μοντέλο του ηθοποιού*. Η γενική ιδέα παραμένει ίδια, όμως ο μοναδικός διανεμητής δε γνωρίζει προκαταβολικά, από τον αρχικό του προγραμματισμό, ποια αντικείμενα αφορά κάθε γεγονός. Τα αντικείμενα, που το γνωρίζουν, εγγράφονται σε μια κατάσταση -στην Python μπορεί να πρόκειται για ένα λεξικό- χειριστών μιας κατηγορίας γεγονότων, που τηρείται στον διανεμητή και του επιτρέπει, όταν προκύψει γεγονός αυτής της κατηγορίας, να γνωρίζει τα αντικείμενα που ενδιαφέρονται γι' αυτό.

Το όνομα *μοντέλο του ηθοποιού* προέκυψε από τον παραλληλισμό με την πρακτική του Hollywood όπου, ενώ οι ηθοποιοί εμφανίζονται σε πρακτορεία και εταιρίες παραγωγής εκδηλώνοντας το ενδιαφέρον τους για ρόλους, δε συνηθίζεται να

παίρνουν τηλέφωνα για να ενημερωθούν για την εξέλιξη. Οι εταιρίες συνήθως τους ενημερώνουν «μη μας καλέσετε, θα σας καλέσουμε εμείς», για να γλυτώσουν τα αναρίθμητα τηλέφωνα ηθοποιών που απορρίφθηκαν για το ρόλο. Ο διανεμητής σε ρόλο εταιρίας παραγωγής καλεί όσα αντικείμενα–ηθοποιούς κρίνει ότι πρέπει.

Το μοντέλο του ηθοποιού χρησιμοποιείται πολύ συχνά σε γραφικές διεπαφές χρήστη (GUIs) και ο λόγος είναι πολύ απλός: ότι όλα τα γραφικά στοιχεία ενός GUI, όπως κουμπιά, check-box, πεδία κειμένων, ετικέτες, μενού, μπάρες κύλισης και ό,τι άλλο υπάρχει σε ένα γραφικό περιβάλλον μπορούν, ειδικά σε συνδυασμό με μια είσοδο από το ποντίκι ή το πληκτρολόγιο, να παράγουν μια ποικιλία γεγονότων.

Είναι τόσα πολλά τα είδη γεγονότων για τα οποία απαιτείται να προγραμματιστούν χειριστές, που, αν ο έλεγχος έπρεπε να τα χειριστεί κατά περίπτωση, ο βρόχος της επανάληψης του διανεμητή θα γινόταν απίστευτα πολύπλοκος. Κατά συνέπεια, γίνεται η επιλογή να ενημερωθούν για κάθε κατηγορία γεγονότος όλα τα εγγεγραμμένα αντικείμενα και αυτά να κρίνουν αν τα αφορά, βάσει για παράδειγμα της θέσης στην οθόνη ενός κλικ του ποντικιού ή της πληροφορίας ποιο γραφικό στοιχείο ή παράθυρο είναι εκείνη τη στιγμή ενεργό.

Η εγγραφή χειριστών γεγονότων στην Python

Μια γραφική διεπαφή χρήστη που περιλαμβάνεται στις βιβλιοθήκες της Python και ήδη γνωρίσαμε, είναι η Tkinter. Θα τη χρησιμοποιήσουμε εδώ για να δώσουμε ένα παράδειγμα. Σε αυτήν όλα τα γεγονότα γραφικής διεπαφής ανήκουν σε μια κλάση γεγονότων. Όλοι οι χειριστές γεγονότων εγγράφονται σε GUI widgets, που περιλαμβάνουν τα γραφικά στοιχεία της διεπαφής, όπως κουμπιά, ετικέτες κλπ. Τα GUI widgets χρησιμοποιούνται το καθένα για το χειρισμό συγκεκριμένων τύπων γεγονότων, όπως τα κλικ στο ποντίκι ή η πίεση κάποιου πλήκτρου στο πληκτρολόγιο.

Ένα πρόγραμμα, που απλά, ανοίγει ένα frame που περιλαμβάνει ένα κουμπί εγκατάλειψης/κλεισίματος (quit button), θα δείχνει κάπως έτσι:

```
#!/usr/bin/env python
import Tkinter as tk
class Application(tk.Frame):
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self.grid()
        self.createWidgets()
    def createWidgets(self):
        self.quitButton = tk.Button(self, text='Quit',
        command=self.quit)
        self.quitButton.grid()
app = Application()
app.master.title('Sample application')
app.mainloop()
```

Τα widgets του Tkinter είναι υλοποιημένα ως κλάσεις. Έτσι, αν θέλουμε να υλοποιήσουμε ένα δικό μας κουμπί, αρκεί, για παράδειγμα, να δημιουργήσουμε το σχετικό αντικείμενο το οποίο όμως, θα πρέπει να συνδέεται με κάποιο παράθυρο ή πλαίσιο.

Στον παραπάνω κώδικα έχουμε υλοποιήσει έναν αυτόνομο εκτελούμενο κώδικα (script) σε Python, ο οποίος εισάγει το Tkinter και τα ονόματα του module, αφού προηγουμένως, για διευκόλυνση το μετονομάσει σε tk. Η κλάση Application κληρονομεί από την έτοιμη κλάση Frame του Tkinter, οπότε και η μέθοδος κατασκευαστής της συνάρτησης καλεί την αντίστοιχη της Frame. Κατόπιν ορίζονται τα widgets που θα χρησιμοποιηθούν και ορίζεται το κουμπί κλεισίματος (quitButton).

Οι γραμμές self.grid() και self.quitButton.grid(), απλώς ρυθμίζουν την εμφάνιση στην οθόνη των δύο αντικειμένων. Στις τρεις τελευταίες γραμμές ορίζεται ένα αντικείμενο κλάσης Application με όνομα app, του οποίου το κείμενο τίτλου ορίζεται σε 'Sample Application' και εκκινεί ο κύριος βρόχος της εφαρμογής σε αναμονή γεγονότων από το ποντίκι ή το πληκτρολόγιο.

Έχουμε καλέσει έτσι τον κατασκευαστή της κλάσης Button και έχουμε δημιουργήσει το αντικείμενο quitButton, ενώ ταυτόχρονα

το έχουμε συνδέσει με το γραφικό αντικείμενο πρώτου επιπέδου στο οποίο ανήκει. Δεν μπορούμε να έχουμε κουμπιά που «πλέουν» στη διεπαφή μας, οπότε τα τοποθετούμε μέσα σε παράθυρα ή frames. Για να γίνουν τα πράγματα χειρότερα, ο μοναδικός μας χειριστής είναι επίσης τετριμμένος: είναι η εντολή quit που ενεργοποιείται, όταν ενεργοποιηθεί το κουμπί quitButton.

Το Tkinter μας δίνει τη δυνατότητα να κάνουμε και πιο ενδιαφέροντα πράγματα το ίδιο εύκολα. Τα widgets προσφέρουν τη μέθοδο *bind* με την οποία μπορούμε να συνδέσουμε γεγονότα με συγκεκριμένα widgets, στην πραγματικότητα μας επιτρέπει να συνδέσουμε τρία πράγματα:

- έναν τύπο γεγονότος, για παράδειγμα το αριστερό κλικ του ποντικιού ή το πάτημα του κουμπιού ENTER στο πληκτρολόγιο
- ένα widget, όπως ας πούμε ένα συγκεκριμένο τύπο κουμπιού στο GUI μας
- και μια συνάρτηση χειριστή γεγονότων.

Δείτε για παράδειγμα τον κώδικα:

```
#!/usr/bin/env python
```

```
import Tkinter as tk
```

```
class Application(tk.Frame):
```

```
    def __init__(self, master=None):
```

```
        tk.Frame.__init__(self, master)
```

```
        self.grid()
```

```
        self.createWidgets()
```

```
    def createWidgets(self):
```

```
        self.quitButton = tk.Button(self, text='Quit')
```

```
        self.quitButton.bind('<Return>', quitButtonEventHandler)
```

```
        self.quitButton.bind('<Button-1>', quitButtonEventHandler)
```

```
        self.quitButton.grid()
```

```
    def quitButtonEventHandler(self):
```

```
        self.quit
```

```
app = Application()
app.master.title('Sample application')
app.mainloop()
```

Εδώ, στον ορισμό του `quitButton` δεν περιλαμβάνεται εντολή. Όμως, στις δύο επόμενες γραμμές τόσο το κουμπί ENTER όσο και το αριστερό κλικ του ποντικιού, μέσα από τα τυπικά ονόματα που τους δίνει το Tkinter, συνδέονται αφενός με το widget `quitButton` και αφετέρου με τη μέθοδο `quitButtonEventHandler`. Μέθοδο την οποία, για λόγους απλότητας του παραδείγματος, έχουμε αφήσει σε απλοϊκή μορφή, δηλαδή να κλείνει την εφαρμογή.

Αν συμβεί οποιοδήποτε από τα δύο γεγονότα, το πάτημα του ENTER στο πληκτρολόγιο ή το πάτημα του αριστερού κουμπιού του ποντικιού πάνω στο κουμπί `quitButton`, η εφαρμογή μας θα κλείσει.

Δραστηριότητες - Ερωτήσεις - Ασκήσεις κεφαλαίου

11.6 Δραστηριότητες

Δραστηριότητα 1

Να ορίσετε κλάση με όνομα `Akeraios`, η οποία θα έχει την ιδιότητα `timi` και τις παρακάτω μεθόδους:

- I. `Anathese_timi(timi)`, η οποία θα αναθέτει τιμή στην ιδιότητα του αντικειμένου.
- II. `Emfanise_timi()`, η οποία θα εμφανίζει την τιμή της ιδιότητας του αντικειμένου.

Στη συνέχεια να δημιουργήσετε στιγμιότυπο της κλάσης `Akeraios` με όνομα `Artios` και να χρησιμοποιήσετε τις παραπάνω μεθόδους για να δώσετε την τιμή 14 στην ιδιότητα του αντικειμένου και να την εμφανίσετε.

Δραστηριότητα 2

Να ορίσετε κλάση με όνομα Student η οποία θα έχει τρεις ιδιότητες για τον αριθμό μητρώου, το ονοματεπώνυμο και τους βαθμούς σε 8 μαθήματα (πίνακας).

Επίσης να ορίσετε τις παρακάτω μεθόδους της κλάσης:

- I. Μια μέθοδο για την ανάθεση τιμών στις ιδιότητες ενός αντικειμένου.
- II. Μια μέθοδο για την εμφάνιση των τιμών των ιδιοτήτων ενός αντικειμένου.
- III. Μια μέθοδο για τον υπολογισμό και την επιστροφή του μέσου όρου βαθμολογίας στα 8 μαθήματα.

Στη συνέχεια να ορίσετε ένα στιγμιότυπο της κλάσης Student με όνομα Chris και να χρησιμοποιήσετε τις παραπάνω μεθόδους για να δώσετε τιμή στις ιδιότητες του αντικειμένου, να τις εμφανίσετε και να υπολογίσετε το μέσο όρο βαθμολογίας του.

Δραστηριότητα 3

Συζητήστε, συμπληρώστε ή επεκτείνετε την παρακάτω σύνοψη βασικών όρων του κεφαλαίου.

Αντικειμενοστρεφής προγραμματισμός: ένας τρόπος προγραμματισμού στον οποίο τα δεδομένα και οι λειτουργίες (συναρτήσεις) είναι οργανωμένα σε αντικείμενα.

Κλάση: Ένας τύπος ορισμένος από το χρήστη. Μια συνταγή δημιουργίας αντικειμένων.

Αντικείμενο: μια δομή δεδομένων και επεξεργαστική ενότητα που έχει δημιουργηθεί βάσει κάποιας κλάσης.

Στιγμιότυπο μιας κλάσης: ένα αντικείμενο το οποίο δημιουργήθηκε χρησιμοποιώντας τη συγκεκριμένη κλάση.

Ιδιότητα αντικειμένου: μία από τις μεταβλητές που σχετίζονται με ένα αντικείμενο.

Μέθοδος: μια συνάρτηση η οποία ορίζεται μέσα σε έναν ορισμό κλάσης και χρησιμοποιείται, συνήθως, από τα στιγμιότυπά της.

Κληρονομικότητα: η μεταφορά χαρακτηριστικών μιας κλάσης σε άλλες κλάσεις (*υποκλάσεις*), που ρητά προέρχονται από την αρχική. Οι υποκλάσεις μπορούν να επαναχρησιμοποιήσουν τις μεταβιβάσιμες μεθόδους και ιδιότητες της γονικής τους κλάσης, αλλά και να προσθέτουν δικές τους.

Πολυμορφισμός: αναφέρεται στη δυνατότητα των αντικειμενοστρεφών μεταγλωττιστών / διερμηνευτών να αποφασίζουν δυναμικά ποια από τις ομώνυμες μεθόδους είναι κατάλληλη να κληθεί, ανάλογα με το αντικείμενο που την καλεί ή τον τύπο και τον αριθμό των παραμέτρων της.

Ενθυλάκωση: καλείται η ιδιότητα που προσφέρουν οι κλάσεις, να «κρύβουν» τα ιδιωτικά δεδομένα τους από το υπόλοιπο πρόγραμμα και να εξασφαλίζουν πως, μόνο μέσω των δημοσίων μεθόδων τους, θα μπορούν αυτά να προσπελαστούν.

11.7 Ερωτήσεις - Ασκήσεις

1. Ποια είναι τα βασικά χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού;
2. Σε τι διαφέρει ο αντικειμενοστρεφής από το Δομημένο προγραμματισμό;
3. Τι είναι η κλάση και τι το αντικείμενο (στιγμιότυπο) μιας κλάσης;
4. Πώς δηλώνουμε μια κλάση στην Python;
5. Τι σημαίνει κληρονομικότητα και τι πολυμορφισμός;
6. Δώστε ένα παράδειγμα κληρονομικότητας.
7. Τι είναι η ενθυλάκωση και ποια τα βασικά της πλεονεκτήματα.
8. Τι είναι ένα γεγονός; Δώστε παραδείγματα.

Σύνοψη - ανακεφαλαίωση

Στο κεφάλαιο αυτό παρουσιάστηκαν οι βασικές αρχές του Αντικειμενοστρεφούς Προγραμματισμού. Γνωρίσαμε τη βασική ορολογία και μάθαμε να δημιουργούμε κλάσεις και να χρησιμοποιούμε αντικείμενα. Τέλος παρουσιάστηκαν οι έννοιες της κληρονομικότητας και της ενθυλάκωσης και πώς τα γεγονότα καθορίζουν τη συμπεριφορά των αντικειμένων

ΕΣΩΤΕΡΙΚΗ ΕΓΚΕΚΡ. ΈΚΔΟΣΗ v201

12

Εισαγωγή
στην
υπολογιστική σκέψη

12. Εισαγωγή στην Υπολογιστική Σκέψη

Εισαγωγή

Η σύγχρονη εποχή χαρακτηρίζεται από ραγδαίες τεχνολογικές εξελίξεις, που επιφέρουν σημαντικές αλλαγές σε διάφορους τομείς της οικονομικής, εργασιακής και κοινωνικής ζωής. Καθημερινά, ακόμα και σε απλές ασχολίες, αυξάνεται η απαίτηση χρήσης πληροφοριακών συστημάτων, υπολογιστικών συσκευών και υπηρεσιών του Διαδικτύου. Σε ένα μεταβαλλόμενο τεχνολογικά περιβάλλον, ο σύγχρονος πολίτης καλείται να αναπτύξει ικανότητες και γνώσεις, που να του επιτρέπουν να αξιοποιεί κατά τον καλύτερο δυνατό τρόπο την τεχνολογία της εποχής του.

Στο κεφάλαιο αυτό θα αναπτυχθεί μια σημαντική -για τη σύγχρονη εποχή- νοητική διαδικασία οργάνωσης και επίλυσης προβλήματος: η *υπολογιστική σκέψη*, που ενισχύει το συνδυασμό των ψηφιακών τεχνολογιών με τις ανθρώπινες ιδέες. Μέσα από απλά παραδείγματα θα προσεγγίσουμε τα βασικά χαρακτηριστικά της *υπολογιστικής σκέψης* και θα εξοικειωθούμε με τις σχετικές έννοιες.

Διδακτικοί στόχοι

Μετά τη μελέτη του κεφαλαίου θα μπορούμε να:

- περιγράψουμε την έννοια της υπολογιστικής σκέψης (ΥΣ)
- αναφέρουμε τα χαρακτηριστικά της ΥΣ
- εξηγούμε τη σημασία της ΥΣ στην επίλυση προβλημάτων της καθημερινής ζωής
- αναλύουμε ένα υπολογιστικό πρόβλημα, περιγράφοντας τις βασικές διαδικασίες με τις οποίες αντιμετωπίζεται, όπως τη διάσπασή του σε απλούστερα, τη λογική οργάνωση των δεδομένων, την αναγνώριση προτύπων, την αναγνώριση πιθανών λύσεων, την περιγραφή της λύσης του με αλγοριθμικό τρόπο, τη γενίκευση της λύσης του.

Λέξεις κλειδιά

Υπολογιστική Σκέψη, επίλυση προβλήματος, διάσπαση προβλήματος, αφαίρεση, οπτική αναπαράσταση δεδομένων, αλγοριθμική λύση, γενίκευση λύσης.

Διδακτικές ενότητες

Η αύξηση της πολυπλοκότητας λόγω της αξιοποίησης προηγμένης τεχνολογίας σε μεγάλη κλίμακα και η αναγκαιότητα υπηρεσιών, που βασίζονται σε υπολογιστικά συστήματα και στο Διαδίκτυο, απαιτούν ολοένα και μεγαλύτερες ικανότητες ανάλυσης, σχεδίασης, υλοποίησης, ελέγχου, διάγνωσης και αντιμετώπισης λαθών. Για να μπορεί ο άνθρωπος να χρησιμοποιεί ανάλογα εργαλεία και να σχεδιάζει εφαρμογές, πρέπει να διαθέτει ικανότητες επίλυσης προβλήματος με τη βοήθεια υπολογιστικών συστημάτων, όπως είναι η Υπολογιστική Σκέψη.

Η **Υπολογιστική Σκέψη** (Computational Thinking) αποτελεί μια νοητική λειτουργία με χαρακτηριστικά, όπως είναι:

- Η αναπαράσταση δεδομένων, μέσω αφαιρέσεων, σε μεγάλα και πολύπλοκα προβλήματα. Για παράδειγμα, μοντέλα για την καθημερινή διανομή φρέσκου γάλατος, σε όλα τα σημεία πώλησης μιας μεγάλης πόλης, ή η προσομοίωση ενός φυσικού φαινομένου για την πρόγνυσή του.
- Η αυτοματοποίηση λύσεων μέσω της αλγοριθμικής σκέψης.
- Η βελτιστοποίηση λύσης.
- Η γενίκευση και μεταφορά μιας διαδικασίας επίλυσης προβλημάτων σε μια ποικιλία προβλημάτων, όπως για παράδειγμα η εύρεση εξόδου σε λαβυρίνθους με διαφορετικό σχήμα.

Όταν κάποιος προσπαθεί να λύσει ένα πρόβλημα με αλγοριθμικό τρόπο, ώστε στη συνέχεια να περιγράψει τη λύση του και να την υλοποιήσει, για να επιλύεται αυτόματα από κάποια υπολογιστική μηχανή, καλείται να απαντήσει σε ερωτήματα σχετικά με τη βέλτιστη λύση του. Για παράδειγμα, πόσο δύσκολο είναι στην επίλυσή του ή ποιος είναι ο ταχύτερος αλγόριθμος επίλυσής του (πολυπλοκότητα χρόνου). Η *πολυπλοκότητα αλγορίθμων* αποτελεί ένα σημαντικό τομέα της Επιστήμης των Υπολογιστών και μελετά πόσο αποδοτικά μπορεί να λυθεί ένα συγκεκριμένο πρόβλημα. Το μοντέλο της *μηχανής Turing* (θυμίζουμε ότι ο Alan Turing είναι από τους θεμελιωτές της Επιστήμης των Υπολογιστών) αξιοποιείται για τη διερεύνηση της δυνατότητας επίλυσης ενός προβλήματος με αλγοριθμικό τρόπο.

12.1 Η έννοια της Υπολογιστικής Σκέψης

Η Wing, που έκανε δημοφιλή την ιδέα της Υπολογιστικής Σκέψης στο χώρο της εκπαίδευσης, επεξηγεί ότι η υπολογιστική σκέψη περιγράφει τη νοητική δραστηριότητα μορφοποίησης ενός προβλήματος, ώστε να επιδέχεται υπολογιστικές λύσεις. Οι λύσεις αυτές μπορεί να υλοποιηθούν είτε από τον άνθρωπο, είτε από τη μηχανή ή από το συνδυασμό ανθρώπων και μηχανών. Η ΥΣ είναι μια προσέγγιση επίλυσης προβλήματος που ενισχύει το συνδυασμό των ψηφιακών τεχνολογιών με τις ανθρώπινες ιδέες. Δεν αντικαθιστά την έμφαση στη δημιουργία, στη λογική και στην κριτική σκέψη, αλλά αντίθετα, τονίζει αυτές τις δεξιότητες μέσα από την ανάδειξη τρόπων για την οργάνωση ενός προβλήματος, ώστε να μπορεί να υποστηριχθεί η επίλυσή τους από υπολογιστές.

Ας σκεφτούμε τα ακόλουθα παραδείγματα.

- Εύρεση ενός λήμματος σε μια πολυσέλιδη εγκυκλοπαίδεια. Επίλυση του προβλήματος με εφαρμογή των αλγορίθμων της σειριακής και της Δυναμικής αναζήτησης. Αναγνώριση και υλοποίηση της αποδοτικότερης και αποτελεσματικότερης λύσης. Γενίκευσή της για επίλυση παρόμοιων προβλημάτων αναζήτησης.
- Εύρεση της πιο σύντομης διαδρομής για τη μετάβαση από το σπίτι στο σχολείο. Μορφοποίηση του προβλήματος, αφαίρεση περιττών στοιχείων και επιλογή της απαραίτητης πληροφορίας. Επισήμανση των βασικών κόμβων (σημείων) των εναλλακτικών διαδρομών σε ένα χάρτη της πόλης. Χάραξη της κάθε διαδρομής ενώνοντας τους βασικούς κόμβους ως διαδικασία αφαίρεσης και μοντελοποίησης του προβλήματος. Αναπαράσταση της κάθε διαδρομής με χρήση κατάλληλης δομής δεδομένων.
- Ανάλυση της διαδικασίας της ουράς μπροστά σε ένα ταμείο supermarket για το τρόπο εξυπηρέτησης. Αναγνώριση των βασικών λειτουργιών της ώθησης και απώθησης σε μία ουρά. Μοντελοποίηση των διαδικασιών και περιγραφή τους με αλγοριθμικό τρόπο.
- Ο προγραμματισμός ενός εκπαιδευτικού ρομπότ για την εύρεση της εξόδου σε ένα λαβύρινθο. Αναγνώριση, ανάλυση και υλοποίηση πιθανών λύσεων του προβλήματος. Ανάλυση

και υλοποίηση του αλγορίθμου (κανόνες κατεύθυνσης, όπως δεξιού ή αριστερού χεριού).

- Σχεδίαση ενός σπιτιού με γεωμετρικά σχήματα πάνω σε χαρτί. Εννοιολογική αφαίρεση και επιλογή των βασικών σχημάτων από τα οποία αποτελείται το σχέδιο του σπιτιού (τετράγωνο- ισόπλευρο τρίγωνο). Αναγνώριση των βασικών ιδιοτήτων κάθε σχήματος που απαιτούνται για το σχεδιασμό τους. Εντοπισμός των διαδικασιών που επαναλαμβάνονται κάθε φορά για το σχεδιασμό του κάθε σχήματος. Αυτοματοποίηση της λύσης σχεδιασμού ενός σπιτιού από υπολογιστή με αλγοριθμικό τρόπο. Υλοποίηση της λύσης σε γλώσσα προγραμματισμού, για παράδειγμα σε γλώσσα Python με χρήση της βιβλιοθήκης turtle.

Η υπολογιστική σκέψη αποτελεί μια ικανότητα των ανθρώπων να είναι ικανοί να λύνουν προβλήματα και όχι σε μια προσπάθεια προσαρμογής τους, ώστε να σκέφτονται σαν υπολογιστές. Η υπολογιστική σκέψη συνδυάζει την επιστήμη, την τεχνολογία και την κοινωνία, που αλληλεπιδρούν. Η επιστημονική έρευνα τροφοδοτεί την τεχνολογική εξέλιξη, η οποία τροφοδοτεί νέες κοινωνικές εφαρμογές κ.ο.κ.

Όλα τα επιλύσιμα προβλήματα μπορούν να λυθούν με πολύ περισσότερους τρόπους από αυτούς που θεωρούμε στην αρχή ως τους μόνους δυνατούς. Συχνά, αρκεί να επανεξετάσουμε το ερώτημα, αναδιατυπώνοντάς το και να σκεφτούμε ελεύθερα. Αυτό συμβαίνει διότι, πολλές φορές, όταν αντιμετωπίζουμε ένα πρόβλημα ψάχνουμε τη λύση, που μας υποδεικνύει η αρχική του διατύπωση.

12.2 Χαρακτηριστικά της Υπολογιστικής Σκέψης

Η υπολογιστική σκέψη περιλαμβάνει, χωρίς να περιορίζεται, τα παρακάτω βασικά χαρακτηριστικά:

- Μορφοποίηση προβλημάτων με τρόπο που επιτρέπει τη χρήση υπολογιστή και άλλων εργαλείων για την επίλυσή τους.
- Εννοιολογική αφαίρεση και επιλογή της απαραίτητης πληροφορίας για την αντιμετώπιση της πολυπλοκότητας των σύνθετων προβλημάτων.
- Διάσπαση ενός προβλήματος σε απλούστερα υποπροβλήματα.

- Λογική οργάνωση και ανάλυση δεδομένων.
- Αναπαράσταση δεδομένων μέσα από μοντέλα και προσομοιώσεις.
- Αναγνώριση, ανάλυση και υλοποίηση πιθανών λύσεων με σκοπό την επίτευξη των αποτελεσματικότερων και αποδοτικότερων συνδυασμών, βημάτων και πόρων.
- Αυτοματοποίηση των λύσεων μέσα από την περιγραφή τους με αλγοριθμικό τρόπο.
- Αξιολόγηση του αλγορίθμου και της λύσης που δόθηκε.
- Γενίκευση και μεταφορά της διαδικασίας επίλυσης του συγκεκριμένου προβλήματος σε μια ευρύτερη ποικιλία από προβλήματα.
- Κατανόηση της ανθρώπινης συμπεριφοράς κατά την αντιμετώπιση ενός προβλήματος.

Για την ανάλυση του προβλήματος και τον εντοπισμό των δομικών ή κύριων στοιχείων που το αποτελούν, είναι απαραίτητο το πρόβλημα να μορφοποιηθεί, ώστε να είναι απαραίτητο να απαλλαγεί από περιττές λεπτομέρειες που αυξάνουν, χωρίς λόγο, τον όγκο των στοιχείων που πρέπει να διαχειριστούμε. Η λειτουργία αυτή είναι γνωστή ως *αφαίρεση* (abstraction) και αποτελεί μια νοητική ικανότητα εντοπισμού των βασικών χαρακτηριστικών ενός αντικείμενου ή γενικότερα μιας κατάστασης. Η αξία της αφαίρεσης έγκειται στη δυνατότητα κριτικής επεξεργασίας δεδομένων και στην ανακάλυψη σχέσεων μεταξύ αντικειμένων ή καταστάσεων. Η αφαιρετική ικανότητα είναι απαραίτητη για τη σωστή ανάλυση ενός προβλήματος σε απλούστερα υποπροβλήματα και τη σαφή διατύπωση της δομής του.

Η υπολογιστική σκέψη περιέχει τη χρήση της αφαίρεσης, τη διάσπαση ενός προβλήματος σε πιο απλά, το χρονοπρογραμματισμό ενός έργου, τη χρήση μεγάλου όγκου δεδομένων (big data), όπως είναι για παράδειγμα η διαχείριση petabytes δεδομένων στο Cern με χρήση της Python. Χαρακτηριστικό της Υπολογιστικής Σκέψης είναι η σύλληψη εννοιών, η οποία απαιτεί σκέψη σε πολλαπλά επίπεδα αφαίρεσης. Άλλο ένα χαρακτηριστικό της, είναι ότι συνδυάζει τη μαθηματική σκέψη με τη σκέψη του μηχανικού. Αυτό, διότι η Επιστήμη των Υπολογιστών στηρίζεται στα Μαθηματικά, αλλά παράλληλα

ελέγχει συστήματα τα οποία αλληλεπιδρούν με τον πραγματικό κόσμο, με αποτέλεσμα να χρειάζεται συνδυασμός μηχανικής και μαθηματικής σκέψης.

Παραδείγματα

Κατά τον υπολογισμό του χρόνου μετάβασης ενός οχήματος από μια τοποθεσία σε μια άλλη, το χρώμα του οχήματος δεν αποτελεί βασικό χαρακτηριστικό. Αντίθετα, η ταχύτητα που μπορεί να αναπτύξει το όχημα και οι εναλλακτικές διαδρομές που μπορεί να ακολουθήσει, είναι βασικά στοιχεία. Η ικανότητα της αφαίρεσης μας επιτρέπει να θεωρήσουμε ότι ένα όχημα της Φόρμουλα 1 και ένα τετρακίνητο τζιπ είναι και τα δύο αυτοκίνητα, παρόλο που έχουν τελείως διαφορετικά χαρακτηριστικά.

Στα Μαθηματικά, όταν λύνουμε ένα πρόβλημα, η απόδοση του αγνώστου στο χαρακτήρα X (έστω X), είναι μια αφαιρετική διαδικασία.

12.3 Υπολογιστική σκέψη και επίλυση προβλημάτων

Η επίλυση ενός προβλήματος με τον υπολογιστή περιλαμβάνει τα ακόλουθα σημαντικά στάδια:

- Τον ακριβή προσδιορισμό του προβλήματος.
- Την περιγραφή της λύσης του με την ανάπτυξη του αντίστοιχου αλγορίθμου.
- Τη διατύπωση του αλγορίθμου σε κατανοητή μορφή από τον υπολογιστή για την υλοποίηση και αυτοματοποίηση της λύσης.

Μέσα από τη διαδικασία επίλυσης ενός προβλήματος, καλλιεργείται η αυστηρότητα και η σαφήνεια της έκφρασης και της διατύπωσης, η απόκτηση δεξιοτήτων αλγοριθμικής προσέγγισης, η ανίχνευση και η δυνατότητα διάκρισης των μερών ενός προβλήματος και η ανάπτυξη ικανοτήτων αναζήτησης και εύρεσης εναλλακτικών λύσεων.

Πολλά από τα ανθρώπινα τεχνουργήματα, φυσικά ή τεχνητά, στηρίζονται στην ιδέα της επαναχρησιμοποίησης–συνδυασμού μικρότερων δομικών μονάδων, με αποτέλεσμα την απλότητα και την ποικιλία αντικειμένων και χαρακτηριστικών. Στην Python η

τεχνική αυτή υλοποιείται με τις συναρτήσεις, δίνοντας καλύτερο έλεγχο και υψηλό επίπεδο αφαίρεσης. Θυμίζουμε ότι οι συναρτήσεις είναι επαναχρησιμοποιήσιμα τμήματα προγραμμάτων, που τα χρησιμοποιούμε για να μην επαναλαμβάνουμε κώδικα μέσα στα προγράμματα μας, εξασφαλίζοντας μέσα από διαδικασίες αφαίρεσης την αυτοτέλειά τους, με κατάλληλη χρήση παραμέτρων.

Δραστηριότητες - Ερωτήσεις - Ασκήσεις

12.4 Δραστηριότητες κεφαλαίου

Δραστηριότητα 1. Διαχείριση προβλημάτων

Για τα τρία διαφορετικά προβλήματα, που παρουσιάζονται στη συνέχεια, να συμπληρωθεί η ακόλουθη πληροφορία:

Κατανόηση του προβλήματος (χώρος του προβλήματος):

- Ποια δεδομένα είναι γνωστά;
- Τι δεν είναι γνωστό;
- Ποιο είναι το ζητούμενο;
- Ποιες είναι οι συνθήκες;
- Ποιο είναι το πλάνο εργασίας για την επίλυση του προβλήματος;
- Να σχεδιαστεί η λύση.

Υλοποίηση της λύσης: Χρησιμοποιώντας το πλάνο εργασίας, να παρουσιαστεί η όλη εργασία και η λύση.

Να γίνει ανακεφαλαίωση και συζήτηση της λύσης.

Πρόβλημα 1. Εύρεση των κάλπικων νομισμάτων με μια ζύγιση

Υπάρχουν δέκα σακιά που περιέχουν 100 νομίσματα το καθένα. Το κάθε νόμισμα ζυγίζει 10 γραμμάρια. Το ένα από τα δέκα σακιά έχει μέσα μόνο κάλπικα νομίσματα, τα οποία ζυγίζουν εννέα (9) γραμμάρια το καθένα. Πώς μπορούμε με μία μόνο ζύγιση σε μία ηλεκτρονική ζυγαριά ακριβείας, να βρούμε ποιο σακί περιέχει τα κάλπικα νομίσματα;

Πρόβλημα 2. Το πρόβλημα της Χειραψίας

Ας υποθέσουμε ότι είκοσι άνθρωποι βρίσκονται μαζί με ένα μαθητή σε ένα δωμάτιο και πρέπει ο μαθητής να ανταλλάξει

χειραψία με κάθε έναν από αυτούς. Με πόσους ανθρώπους τελικά θα ανταλλάξει χειραψία; Εάν υπάρχουν N ($N > 0$) άνθρωποι μαζί με το μαθητή στο δωμάτιο, με πόσους τελικά θα έρθει αυτός σε επαφή;

Πρόβλημα 3. Βελτιστοποίηση

Να μελετηθεί το πρόβλημα που θα υπολογίζει τα ρέστα που πρέπει να δώσει ένα αυτόματο μηχάνημα έκδοσης εισιτηρίων. Τα εισιτήρια κοστίζουν 0.95€. Ο αλγόριθμος θα δέχεται το ποσό που πληρώνει ο πελάτης και θα επιστρέφει τον αριθμό των κερμάτων με αξία 2€, 1€, 0.50€, 0.20€, 0.10€. Ο αλγόριθμος πρέπει να λειτουργεί με τέτοιο τρόπο, ώστε τα ρέστα να δίνονται με το μικρότερο δυνατό (βελτιστοποίηση) αριθμό νομισμάτων.

Δραστηριότητα 2. Γενίκευση

Ο σημερινός αριθμός αυτοκινήτων που κυκλοφορούν σε μια πόλη είναι 90000. Αν ο αριθμός αυτός αυξάνεται με ετήσιο ρυθμό 5%, να γραφεί σενάριο σε Python που υπολογίζει σε πόσα χρόνια ο αριθμός των αυτοκινήτων θα ξεπεράσει τις 160000. Να εμφανίζει πόσα θα είναι τότε τα αυτοκίνητα. Γενικεύστε το πρόγραμμα για Arith_Aytok αυτοκίνητα, που αυξάνονται με ετήσιο ρυθμό (Rythmos) και ξεπερνούν ένα δοσμένο όριο (Orio). Τέλος, να οπτικοποιηθεί ο κώδικας στο <http://www.pythontutor.com/>

Δραστηριότητα 3. Γενίκευση

Μελετήστε το πρόβλημα των πύργων του Ανόι βλέποντας την οπτικοποίησή του από το Εθνικό Αποθετήριο Εκπαιδευτικού Περιεχομένου «Φωτόδεντρο»:

(<http://photodentro.edu.gr/aggregator/lo/photodentro-lor-8521-1010>).

Το πρόβλημα να επιλυθεί αρχικά για τρεις δίσκους, σε φυσική γλώσσα. Να αξιοποιηθεί η λύση αυτή για να λυθεί το πρόβλημα με τέσσερις δίσκους. Τέλος, να γενικευθεί η λύση για οποιοδήποτε πλήθος δίσκων.

12.5 Ερωτήσεις - Ασκήσεις

- Να αναφερθούν πέντε βασικά χαρακτηριστικά της υπολογιστικής σκέψης.
- Να αναφερθούν τρία καθημερινά παραδείγματα υπολογιστικής σκέψης.
- Να περιγραφεί σύντομα η λειτουργία της Αφαίρεσης για την κατανόηση σύνθετων προβλημάτων.
- Να περιγραφεί σύντομα η λειτουργία της Γενίκευσης της διαδικασίας επίλυσης προβλήματος.

Σύνοψη

Στο κεφάλαιο αυτό ασχοληθήκαμε με τα χαρακτηριστικά της υπολογιστικής σκέψης στην επίλυση προβλημάτων. Δόθηκε, με τη βοήθεια παραδειγμάτων, έμφαση στη σημασία της υπολογιστικής σκέψης στην επίλυση προβλημάτων της καθημερινής ζωής.

Αναφορές - Βιβλιογραφία

(Η τελευταία προσπέλαση στις ηλεκτρονικές πηγές έγινε τον Απρίλιο του 2016)

Alice, (2016), 3D περιβάλλον κινούμενων γραφικών,
<http://www.alice.org/>

Ben Shneiderman and Catherine Plaisant. Designing the User Interface. Pearson Education, Boston, 4th edition, 2005

Berkeley, (2015), <http://python.berkeley.edu/>.

Donald A. Norman. The Design of Everyday Things.
Doubleday/Currency, New York, NY, 1990. First published as
The Psychology of Everyday Things.

Downey, A. (2012) «Think Python, How to think like a computer scientist», O' Reilly.
<http://www.greenteapress.com/thinkPython/>.

Ferg, S. (2006) "Event-Driven Programming: Introduction, Tutorial, History". Διαθέσιμο στη θέση:
http://Tutorial_EventDrivenProgramming.sourceforge.net

Jeannette M. Wing, "Computational Thinking," Communications of the ACM, CACM vol. 49, no. 3, March 2006, pp. 33-35.

Levitin, Anany (2007), The Design & Analysis of Algorithms.
Pearson Education.

Miller, N., M., and Ranum, L., D. (2011). Problem Solving with Algorithms and Data Structures using Python. Franklin, Beedle & Associates; 2nd edition.

Object-Oriented Systems Analysis and Design Using UML (2nd edition). S. Bennett, S. McRobb, R. Farmer, McGraw Hill, 2002.

Python Tutorial http://python-tutorial-greek.readthedocs.org/en/latest/oop_general.html

Shipman, J, W. (2013) "Tkinter 8.5 reference: a GUI for Python"

Swaroop, C., H. (2013) "A byte of Python",
<http://www.swaroopch.com/notes/Python/> & και στα Ελληνικά
<http://ubuntu-gr.org/> Ελληνική κοινότητα του Ubuntu

Swaroop, C., H. (2013) "A byte of Python", Διαδικτυακή έκδοση ηλεκτρονικού βιβλίου <http://www.swaroopch.com/notes/Python/>

με άδεια Creative Commons Attributions –ShareAlike 4.0 International License. Το βιβλίο έχει μεταφραστεί και στα Ελληνικά από την Ελληνική κοινότητα του Ubuntu.

<http://ubuntu-gr.org/> και είναι διαθέσιμο στο Διαδίκτυο.

Thomas T. Cormen, Charles E. Leiserson, and Ronald L. Rivest. 1990. Introduction to Algorithms. MIT Press, Cambridge, MA, USA.

Tutorial από το δικτυακό τόπο της γλώσσας Python για την γνωριμία με τη γλώσσα Python – Δομή Ελέγχου και Δομές Επανάληψης <https://docs.Python.org/2/tutorial/controlflow.htm>

Αβούρης Ν. (2000) «Εισαγωγή στην επικοινωνία ανθρώπου υπολογιστή», εκδ. Δίαυλος.

Αβούρης Ν. Κουκιάς Μ., Παλιουράς Β, Σγάρμπας Κ. (2013) «Εισαγωγή στους υπολογιστές με τη γλώσσα Python», Εκδόσεις Πανεπιστημίου Πατρών, Πάτρα.

Ανάπτυξη Γραφικής Διεπαφής με τη βιβλιοθήκη Tkinter της Python

ΑΠΣ_Τομέα_Πληρ_ΕΠΑΛ (2015). Αναλυτικά Προγράμματα Σπουδών του μαθήματος Γενικής Παιδείας «Εισαγωγή στις Αρχές της Επιστήμης των Η/Υ» της Β΄ και Γ΄ τάξης Ημερήσιων και Γ΄ και Δ΄ τάξης Εσπερινών ΕΠΑ.Λ. και των μαθημάτων ειδικοτήτων του Τομέα Πληροφορικής της Ομάδας Προσανατολισμού Τεχνολογικών Εφαρμογών των τάξεων Β΄ και Γ΄ Ημερήσιων και Β΄, Γ και Δ΄ Εσπερινών ΕΠΑ.Λ., ΦΕΚ 2010. 16/9/2015.

Αρχές Προγραμματισμού Υπολογιστών, (2015). Διδακτικό υλικό των Αράπογλου Α., Βραχνός Ε., Κανίδης Ε., Μακρυγιάννης Π., Μπελεσιώτης Β., Τζήμας Δ, ISBN 978-960-06-5141-6

Βιβλιοθήκη οπτικοποίησης της Google με Python

ΒΙΚΙΠΑΙΔΕΙΑ Αντικειμενοστρεφής Προγραμματισμός

https://el.wikipedia.org/wiki/Αντικειμενοστρεφής_προγραμματισμός

Βικιπαίδεια, υλικό και χρήσιμοι σύνδεσμοι, με έλεγχο ποιότητας, <http://el.wikipedia.org/wiki/Python>

Δικτυακός κόμβος με παράδειγμα χρήσης της sqlite στην Python

Δικτυακός κόμβος υποστήριξης με πλούσιο υλικό πολυμέσων για τη διδασκαλία της γλώσσας Python,
<http://www.Pythonschool.net/>

Δικτυακός κόμβος υποστήριξης της γλώσσας Python,
<https://www.Python.org/>

Δικτυακός κόμβος υποστήριξης της γλώσσας Python,
<http://www.pythonschool.net> και ειδικότερα των Βάσεων Δεδομένων
<http://www.pythonschool.net/category/databases.html>

Δικτυακός τόπος για τη διδασκαλία της επιστήμης της Πληροφορικής χωρίς υπολογιστές Computer Science Unplugged: <http://www.csunplugged.org>

Δικτυακός τόπος εκπαίδευσης στη γλώσσα Python:
<https://www.codecademy.com/learn/python>

Δικτυακός τόπος με πλούσιο σχετικό υλικό <http://www.python-course.eu/course.php>

Δικτυακός τόπος με πλούσιο σχετικό υλικό
http://www.tutorialspoint.com/python/python_files_io.htm

Δικτυακός τόπος με σχετικό υλικό υποστήριξης
http://www.tutorialspoint.com/python/python_modules.htm

Δικτυακός τόπος με ψηφιακό υλικό για την εισαγωγή στην αλγοριθμική σκέψη: <http://www.teaching-materials.org/algorithms/>

Δικτυακός τόπος οπτικοποίησης κώδικα:
<http://www.pythontutor.com/>

Ενδεικτικές Βιβλιοθήκες προγραμματισμού για οπτικοποίηση δεδομένων

Εργαλείο on-line για τη κατασκευή διαγραμμάτων ροής
<https://www.glify.com>

Ευρετήριο διεπαφών προγραμματισμού εφαρμογών της Python,

Λεβεντέας, Δ., (2010), «Taspython. Εκμάθηση Python Βήμα, Βήμα. Οδηγός Python Μέσω Παραδειγμάτων», Ομάδα TasPython.

Οδηγός για τον Εκπαιδευτικό για το Πρόγραμμα Σπουδών του Μαθήματος «Πληροφορική» Γ' Τάξης Γενικού Λυκείου, στο πλαίσιο του έργου «ΝΕΟ ΣΧΟΛΕΙΟ (Σχολείο 21ου αιώνα) – Νέο Πρόγραμμα Σπουδών», Υποέργο 9: «Εκπόνηση Προγραμμάτων Σπουδών Γενικού Λυκείου, Μουσικών και Καλλιτεχνικών Λυκείων», Υ.ΠΟ.ΠΑΙ.Θ, Ινστιτούτο Εκπαιδευτικής Πολιτικής (Ι.Ε.Π), Ιανουάριος 2015.

Οδηγός εκπαιδευτικού - Κεφάλαιο 1, ΈΡΓΟ «Νέο Σχολείο (Σχολείο 21ου αιώνα) – Νέο Πρόγραμμα Σπουδών» (κωδ. ΟΠΣ: 295450/Υποέργο 9/ Δράση: Εκπόνηση των Προγραμμάτων Σπουδών και Οδηγού Εκπαιδευτικού Μαθήματος «Πληροφορική» Γ' τάξης Γενικού Λυκείου – ΦΕΚ 189/23-1-2015).

ΠΣ_Γ_ΓΕΛ (2015). Πρόγραμμα Σπουδών και Οδηγός Εκπαιδευτικού Μαθήματος «Πληροφορική» Γ' τάξης Γενικού Λυκείου (ΦΕΚ189/23-1-2015) (ΠΣ_Γ_ΓΕΛ, 2015). Έργο ΙΕΠ «Νέο Σχολείο (Σχολείο 21ου αιώνα) – Νέο Πρόγραμμα Σπουδών» (κωδ. ΟΠΣ: 295450/Οριζόντια Πράξη στις 8 Π.Σ., 3 Π.Σ.τ. Εξ., 2 Π.Σ.τ. Εισ./Υποέργο 9 Δράση: Εκπόνησης των Προγραμμάτων Σπουδών και Οδηγού Εκπαιδευτικού Μαθήματος «Πληροφορική» Γ' τάξης Γενικού Λυκείου. Μέλη (αξιολογητές, συντονιστής, εμπειρογνώμονες): (εδώ αλφ/κά-αναλυτικά στο κάθε υλικό/παραδοτέα): Αράπογλου Αριστείδης, εκπαιδευτικός ΠΕ19, υπ. ΚΕ.ΠΛΗ.ΝΕ.Τ - Βαρζάκας Παναγιώτης, μέλος Ε.Π. -Α.Τ.Ε.Ι. - Βραχνός Ευριπίδης, εκπαιδευτικός ΠΕ19 - Κανίδης Ευάγγελος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής - Λέκκα Δήμητρα, εκπαιδευτικός ΠΕ19 - Μαραγκός Κωνσταντίνος, εκπαιδευτικός ΠΕ19 - Μαυρίδης Ιωάννης, μέλος ΔΕΠ - Μπελεσιώτης Βασίλειος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής - Παπαδάκης Σπυρίδων, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής - Τζήμας Δημήτριος, εκπαιδευτικός ΠΕ19.

Σημειώσεις επικοινωνίας Ανθρώπου – Μηχανής (2005), Καθηγητής Γιάννης Ιωαννίδης, Δρ. Γιώργος Λέππουρας, διαθέσιμο στο http://cgi.di.uoa.gr/~ys08/EAM_simeiwseis.pdf (Πρόσβαση Ιούνιος 2015)

Συγγραφείς_Προγραμματισμός. (2016). Ομάδες συγγραφής διδακτικού υλικού μαθήματος Προγραμματισμός, Γ' ΕΠΑ.Λ.,

Τομέας Πληροφορικής, ΑΔΑ ΩΓΣΝΟΞΛΔ-Τ6Ψ και 6ΥΓΖΟΞΛΔ-ΛΨΥ, διαθέσιμες στο <https://et.diavgeia.gov.gr/>

Το διαδικτυακό μάθημα

<http://www.highschoollearning.com/freelessons/oop/ObjectOrientedProgramming.htm>

Υπολογιστική σκέψη και την επίλυση προβλήματος -

Εκπαιδευτικά παιχνίδια: <http://games.thinkingmyself.com/>

Ψηφιακό Σχολείο - Αποθετήριο Φωτόδεντρο. Περιέχει μεταξύ

άλλων μαθησιακά αντικείμενα για την προσέγγιση της

αλγοριθμικής σκέψης ["http://photodentro.edu.gr/lor/subject-search?locale=el"](http://photodentro.edu.gr/lor/subject-search?locale=el)

Ωρολόγιο Πρόγραμμα (2015). των μαθημάτων Γενικής Παιδείας

της Α΄ τάξης Εσπερινών ΕΠΑ.Λ. και των Β΄, Γ΄ τάξεων

Ημερησίων και Α΄, Β΄ και Γ΄ τάξεων Εσπερινών ΕΠΑ.Λ. ανά

Ειδικότητα Τομέα Ομάδας Προσανατολισμού, ΦΕΚ1053/2015.

Χρήσιμοι Διαδικτυακοί Τόποι

<http://blog.yhathq.com/posts/ggplot-for-Python.html>

http://matplotlib.org/users/pyplot_tutorial.html

http://sebastianraschka.com/Articles/2014_sqlite_in_python_tutorial.html

<http://stanford.edu/~mwaskom/software/seaborn/>

<http://www.blog.pythonlibrary.org/2012/07/18/python-a-simple-step-by-step-sqlite-tutorial/>

[http://www.greenteapress.com/thinkPython/.](http://www.greenteapress.com/thinkPython/)

[http://www.greenteapress.com/thinkPython/.](http://www.greenteapress.com/thinkPython/)

http://www.Python-course.eu/Python_tkinter.php

<http://www.swaroopch.com/notes/Python/>

<http://www.swaroopch.com/notes/Python/>

<https://code.google.com/p/google-visualization-Python/>

<https://pypi.Python.org/pypi>

<https://pypi.Python.org/pypi/ggplot>

<https://wiki.Python.org/moin/TkInter>

Ευρετήριο όρων

- API, 154
- Class, 197
- Computational Thinking, 227
- Data Base Management System, 172
- GUI, 154
- LIFO, 141
- Open, 93
- Read, 95
- Self, 199
- SQL, 175
- Tkinter, 159
- UML, 29
- Write, 95
- Αλγόριθμος ταξινόμησης ευθείας ανταλλαγής, 80
- Ανοιχτά προβλήματα, 14
- Αντικειμενοστρεφής σχεδίαση, 28
- Αποδομητής, 195
- Αρθρώματα, 118, 125
- Αρθρωτός ή Τμηματικός Προγραμματισμός, 24
- Αρχείο κειμένου, 93
- Αρχές σχεδίασης διεπαφής, 157
- Αρχιτεκτονική τριών επιπέδων ΒΔ., 174
- Βάση δεδομένων, 172
- Γραφικών διεπαφή, 154
- Γράφοι, 148
- Δέντρα, 148
- Δηλωτικός προγραμματισμός, 22
- Διαγραμματική αναπαράσταση προβλήματος, 16
- Διάσχιση λίστας, 135
- Διαχείριση καταλόγων, 101
- Δομή Ακολουθίας, 47
- Δομή δεδομένων, 128
- Δομή επανάληψης, 53
- Δομή επιλογής, 48
- Διαδική αναζήτηση, 68
- Εμβέλεια, 115
- Ενθυλάκωση, 212
- Ενσωματωμένες συναρτήσεις Python, 41
- Εξωτερικές βιβλιοθήκες, 42
- Επικοινωνία Ανθρώπου-Υπολογιστή, 155
- Επιλύσιμα προβλήματα, 14
- Ιδιότητες αντικειμένου, 194
- Κατασκευαστής, 194
- Κατηγορίες συναρτήσεων, 111
- Κλάσεις, 195
- Κληρονομικότητα, 196, 207

- Κύκλος ανάπτυξης προγράμματος/λογισμικού, 19
- Λεξικά, 147
- Λίστα, 131
- Μέθοδοι, 194
- Μεθοδολογίες εργασίας ανάλυση δομής προβλήματος, 15
- Μοντέλο καταρράκτη, 19
- Μοντέλο σπείρας, 20
- Ολοκληρωμένο Περιβάλλον Ανάπτυξης Προγραμμάτων IDLE, 30
- Ορίσματα τρόπου προσπέλασης αρχείου, 94
- Ουρά, 144
- Πακέτα, 123, 125
- Παράμετροι, 112
- Παράμετροι συναρτήσεων, 60
- Πολυμορφισμός, 196
- Πρόβλημα*, 14
- Προστακτικός προγραμματισμός, 21
- Πρότυπη βιβλιοθήκη της Python, 121
- Πρωτεύον κλειδί πίνακα ΒΔ, 179
- Σειριακή αναζήτηση, 67
- Στάδια επίλυσης προβλήματος, 14
- Στατικές μέθοδοι, 205
- Στιγμιότυπο, 198
- Στοίβα, 140
- Συγχώνευση δυο ταξινομημένων λιστών, 137
- Συναρτήσεις, 125
- Συνάρτηση range, 138
- Σύνθετα προβλήματα, 14
- Σύστημα Διαχείρισης Βάσης Δεδομένων, 172
- Ταξινόμηση Ευθείας ανταλλαγής, 76
- Ταξινόμηση με Εισαγωγή, 81
- Ταξινόμησης με Εισαγωγή, 84
- Τρόποι αναπαράστασης αλγορίθμου, 17
- Τροποποιητές πρόσβασης, 214
- Τύποι και δομές δεδομένων στις Γλ. Προγρ/σμού, 44
- Υπογραφή, 204
- Υπολογιστικά προβλήματα, 14
- Υπολογιστική Σκέψη, 227
- Χαρακτηριστικά αρχείου, 99
- Χαρακτηριστικά υπολογιστικής σκέψης, 229
- Χαρακτηριστικά υποπρογράμματος, 107

