Υπουργείο Παιδείας, Έρευνας και Θρησκευμάτων
Ινστιτούτο Εκπαιδευτικής Πολιτικής

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

Προγραμματισμός Υπολογιστών

Οδηγός Εκπαιδευτικού

Γ' Τάξη ΕΠΑ.Λ.
Τομέας Πληροφορικής

Ινστιτούτο Τεχνολογίας Υπολογιστών & Εκδόσεων
«ΔΙΟΦΑΝΤΟΣ»

ΙΝΣΤΙΤΟΥΤΟ ΕΚΠΑΙΔΕΥΤΙΚΗΣ ΠΟΛΙΤΙΚΗΣ

Πρόεδρος: **Κουζέλης Γεράσιμος**, Καθηγητής ΕΚΠΑ

Γραφείο έρευνας, σχεδιασμού και εφαρμογών Β΄

Επιστημονικά Υπεύθυνος:
Δρ. **Τσαπέλας Θεοδόσιος,** Σύμβουλος Β΄ Πληροφορικής

Συγγραφική ομάδα

Αράπογλου Αριστείδης, Εκπαιδευτικός Πληροφορικής
Βραχνός Ευριπίδης, Εκπαιδευτικός Πληροφορικής
Κανίδης Ευάγγελος, Σχολικός Σύμβουλος Πληροφορικής
Λέκκα Δήμητρα, Εκπαιδευτικός Πληροφορικής
Μακρυγιάννης Παναγιώτης, Εκπαιδευτικός Πληροφορικής
Μπελεσιώτης Βασίλειος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής
Παπαδάκης Σπυρίδων, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής
Τζήμας Δημήτριος, Εκπαιδευτικός Πληροφορικής

Επιμέλεια - συντονισμός ομάδας

Κανίδης Ευάγγελος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής
Μπελεσιώτης Βασίλειος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής

Επιτροπή κρίσης

Βογιατζής Ιωάννης, Επίκουρος Καθηγητής, ΑΤΕΙ Αθήνας
Εφόπουλος Βασίλειος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής
Κωτσάκης Σταύρος, Σχολικός Σύμβουλος ΠΕ19-Πληροφορικής

Πίνακας περιεχομένων

Εισαγωγικό σημείωμα συγγραφέων

Το παρόν σύγγραμμα έχει ακολουθήσει το Α.Π.Σ. του μαθήματος (ΦΕΚ 2010 τ.Β 16-9-2015) και τη διδακτέα - εξεταστέα ύλη, όπως αυτή καθορίζεται στην ΥΑ Φ6/160716/Δ4 (2016) και αποτελεί μέρος του όλου διδακτικού υλικού του μαθήματος, δηλαδή του Βιβλίου, Τετραδίου Εργασιών και Λύσεων μαθητή.

Η υλοποίησή του έγινε αμισθί από το σύνολο των εμπλεκομένων, βάσει της απόφασης του ΔΣ του ΙΕΠ (ΑΔΑ: ΩΓΣΝΟΞΛΔ-Τ6Ψ & 6ΥΓΖΟΞΛΔ-ΛΨΥ).

Οφείλουμε να ευχαριστήσουμε τη φιλόλογο κα Ευφροσύνη Δεληγιάννη, Σχολική Σύμβουλο ΠΕ02 για την άτυπη μεν αλλά καθοριστική βοήθειά της.

Στα κείμενα του παρόντος υλικού και για λόγους απλοποίησης και μη διάσπασης της προσοχής, χρησιμοποιείται το δεύτερο πληθυντικό πρόσωπο, καλύπτοντας και τα δύο γένη.

Γενικές οδηγίες μαθήματος

Η φιλοσοφία του μαθήματος

Οι περισσότερες από τις δραστηριότητες που δίνονται τόσο στο Βιβλίο Μαθητή (ΒΜ) όσο και στο Τετράδιο Μαθητή (ΤΕΜ) ακολουθούν μια διερευνητική/ανακαλυπτική προσέγγιση, σύμφωνα με την οποία οι μαθητές κατακτούν τους διδακτικούς στόχους του μαθήματος, μέσα από συνεχείς πειραματισμούς με το προγραμματιστικό περιβάλλον. Ένας τρόπος για να το πετύχουμε αυτό είναι να δίνουμε τμήματα κώδικα στους μαθητές για να πειραματιστούν με αυτά και να εξάγουν συμπεράσματα που θα τους βοηθήσουν να σχηματίσουν το θεωρητικό μοντέλο (αλγόριθμο) που θέλουμε, τον οποίο στη συνέχεια υλοποιούν στη γλώσσα προγραμματισμού Python. Η μελέτη των αλγορίθμων με διερευνητικό τρόπο χρειάζεται υπομονή, επιμονή και χρόνο, διαδικασία που σε κάποιες περιπτώσεις είναι δύσκολο να μπει στα στενά πλαίσια του χρονοπρογραμματισμού ενός σχεδίου μαθήματος.

Σε όλες τις ασκήσεις-δραστηριότητες οι αλγόριθμοι διατυπώνονται στη γλώσσα Python, ώστε να μπορούν να εκτελεστούν άμεσα. Η ψευδογλώσσα και τα διαγράμματα ροής θεωρούνται γνωστά από τη Β' τάξη και μπορούν να χρησιμοποιηθούν από τον εκπαιδευτικό, κατά την κρίση του, για την παρουσίαση και την επεξήγηση των δομών ελέγχου (επιλογής, επανάληψης). Η υλοποίηση όμως ενός αλγορίθμου ή τμήματος του, θα πρέπει να ζητείται μόνο σε γλώσσα Pyhton.

Η γλώσσα προγραμματισμού Python χαρακτηρίζεται από την πληθώρα εντολών/συναρτήσεων και βιβλιοθηκών που διαθέτει για την αντιμετώπιση ποικίλων προβλημάτων. Οι ασκήσεις και δραστηριότητες που υπάρχουν στο ΒΜ, στο ΤΕΜ και στον παρόντα Οδηγό Εκπαιδευτικού, μπορούν να λυθούν με χρήση των εντολών που περιέχουν τα σχολικά βιβλία της Β' και Γ' τάξης. Έτσι, ο μαθητής πρέπει να γνωρίζει και να μπορεί να χρησιμοποιήσει τις παρακάτω συναρτήσεις και μεθόδους οι οποίες αναφέρονται στα σχολικά βιβλία.

Γενικές συναρτήσεις	len, range, type, id, input, raw_input
Μετατροπές τύπων	str, int, float, list, bool
Μαθηματικές Συναρτήσεις	abs, pow, divmod, math.sqrt
Μέθοδοι λίστας	List.append, List.insert, List.pop
Μέθοδοι αρχείου	open, File.close, File.read, File.readline
Λειτουργίες λεξικού	dict, del

** Στον πίνακα, όπου **file** είναι ένα αρχείο και **list** μια λίστα.*

Σκοπός του μαθήματος είναι η ανάπτυξη της αλγοριθμικής και υπολογιστικής σκέψης του μαθητή με όχημα τη γλώσσα Python και όχι η σε βάθος εκμάθηση της γλώσσας Python με όλες τις δυνατότητες που είναι ενσωματωμένες σε αυτήν ή παρέχονται μέσω των βιβλιοθηκών, που μπορεί να βρει και χρησιμοποιήσει κάποιος. Σε εκπαιδευτικό περιβάλλον η επίλυση μιας άσκησης, ο τρόπος με τον οποίο ο μαθητής φτάνει στη λύση, έχει πολύ συχνά περισσότερη σημασία από το τελικό αποτέλεσμα της λύσης.

Παράλληλα δεν πρέπει να αγνοηθεί ότι κάθε επιστημονική ορθή λύση είναι αποδεκτή εφόσον συνάδει με το πλαίσιο της εκφώνησης.

Για το λόγο αυτό σε κάθε άσκηση θα πρέπει ο εκπαιδευτικός να οριοθετεί τον τρόπο επίλυσης, προφορικά ή μέσα από την εκφώνηση της άσκησης, ώστε να γίνεται η όσο το δυνατόν καλύτερη επίτευξη των διδακτικών στόχων. Ενδεικτικά ο εκπαιδευτικός μπορεί να επεκταθεί στη διδασκαλία του και να επι-

τρέψει τη χρήση συναρτήσεων και μεθόδων που δεν αναφέρονται στα σχολικά βιβλία της Β' και Γ' τάξης, όπως οι sorted, reverse, max, min, sum, avg, μέθοδος join κ.ά, ή αντίθετα να απαγορεύσει τη χρήση άλλων συναρτήσεων (εκτός σχολικών βιβλίων) ή ακόμα και κάποιου τελεστή, όπως για παράδειγμα του τελεστή in, ο οποίος θεωρείται γνωστός.

Επισημαίνεται ότι οι εκτός των σχολικών βιβλίων βιβλιοθήκες, συναρτήσεις και μέθοδοι δεν αποτελούν μέρος της θεωρίας και δεν μπορούν να εξεταστούν.

Το υλικό του παρόντος **απευθύνεται στον εκπαιδευτικό** και για το λόγο αυτό περιέχει αναφορές σε έννοιες και πρακτικές, όπως για παράδειγμα βελτιστοποίησης, που δεν αναφέρονται στο βιβλίο μαθητή. Οι έννοιες αυτές δίνονται για πληρέστερη ενημέρωση του εκπαιδευτικού, μπορούν να αξιοποιηθούν στην εκπαιδευτική διαδικασία κατά την κρίση του, αλλά δεν εξετάζονται.

Στην αρχή του μαθήματος, προτείνεται ο εκπαιδευτικός να εκμεταλλευτεί και όσο αυτό είναι δυνατόν, το διερμηνευτή της Python IDLE, ώστε οι μαθητές του να επαναλάβουν εντολές και δομές που γνώρισαν στη Β' τάξη. Μέσα από διευρευνητικές δραστηριότητες οι μαθητές θα ανακαλύψουν περαιτέρω χαρακτηριστικά της γλώσσας. Αφού διαπιστώσει ότι έχουν εξοικειωθεί στον επιθυμητό βαθμό με το περιβάλλον και με τις βασικές εντολές της Python, τότε μπορούν προχωρήσουν στη συγγραφή ολοκληρωμένων προγραμμάτων στο συντάκτη του περιβάλλοντος προγραμματισμού. Προτείνεται επίσης η χρήση του online Python tutor (pythontutor.com) για την οπτικοποίηση των δομών δεδομένων κατά την εκτέλεση του κώδικα.

Σημαντικές Παρατηρήσεις

- Η έκδοση της Python που χρησιμοποιείται στο μάθημα είναι η 2 και πιο συγκεκριμένα οποιαδήποτε μεταγενέστερη έκδοση της 2.7.10.
- Αν θέλετε να γράψετε ελληνικούς χαρακτήρες μέσα σε ένα πρόγραμμα, είτε ως σχόλια, είτε ως αλφαριθμητικά, θα πρέπει η πρώτη γραμμή του αρχείου python να είναι η παρακάτω:

  # -*- coding: utf-8 -*-
- Το περιβάλλον προγραμματισμού που προτείνεται είναι το IDLE που συνοδεύει την Python 2. Μπορείτε να χρησιμοποιήσετε όμως και άλλα περιβάλλοντα όπως είναι για παράδειγμα το PyScripter.

- Αν χρησιμοποιείτε Linux, η Python που διαθέτει, δεν περιέχει το IDLE, οπότε θα πρέπει να αναζητήσετε και εγκαταστήσετε το αρχείο idle-python2.6 (2016).

Παρατηρήσεις για τις λύσεις των δραστηριοτήτων

Σε όλες τις δραστηριότητες/ασκήσεις που ζητείται να εισαχθεί από το πληκτρολόγιο μια λίστα αντικειμένων και δεν αναφέρεται το πλήθος των στοιχείων που θα εισαχθούν αλλά και ούτε κάποιο κριτήριο διακοπής της εισαγωγής, μπορείτε να θεωρήσετε ότι η εισαγωγή των δεδομένων σταματάει όταν ο χρήστης δώσει την τιμή None.

Σε ορισμένες περιπτώσεις χρησιμοποιούνται συναρτήσεις χωρίς την ανάλυσή τους, που έχουν οριστεί ήδη σε προηγούμενες δραστηριότητες και απλά αντιγράφονται ξανά. Οπότε, για λεπτομέρειες, θα πρέπει να ανατρέξετε στην αντίστοιχη δραστηριότητα, διαδικασία που άλλωστε χαρακτηρίζει την *επαναχρησιμοποίηση κώδικα*.

Η χρήση της input, για την εισαγωγή δεδομένων, δεν αποκλείει την εισαγωγή αλφαριθμητικών. Για να γίνει όμως σωστά θα πρέπει αυτά να περικλείονται σε εισαγωγικά (quotes), όπως ακριβώς αναφέρονται και μέσα στον κώδικα. Έτσι, για παράδειγμα αν θέλουμε να δώσουμε το όνομα *Τομέας Πληροφορικής* θα το δώσουμε ως εξής: "*Τομέας Πληροφορικής*".

Αν θέλουμε να γράψουμε μια πολύ μεγάλη εντολή σε περισσότερες από μια γραμμές, τότε χρησιμοποιούμε το σύμβολο **** .

Παρατήρηση για το Χρονοπρογραμματισμό

Στον παρακάτω πίνακα δίνεται ενδεικτικός χρονοπρογραμματισμός για τη διδασκαλία του μαθήματος, σύμφωνα με το Πρόγραμμα Σπουδών. Σε περίπτωση που η διδακτέα - εξεταστέα ύλη είναι μειωμένη, η κατανομή των ωρών των ενοτήτων που αφαιρούνται στις υπόλοιπες ενότητες, γίνεται κατά την κρίση του διδάσκοντος. Ο χρονοπρογραμματισμός που βρίσκεται στην τελευταία στήλη του πίνακα, αποτελεί πρόταση. Οι συγγραφείς θεωρούν τη Θεωρία και το Εργαστήριο του μαθήματος ως ενιαίο σύνολο, το οποίο διδακτικά δεν πρέπει να διαχωριστεί.

Ειδικά, για κεφάλαια ή τμήματα της ύλης που σχετίζονται με τη Β΄ Τάξη, προτείνεται να ακολουθηθεί μια διαδικασία αναγνώρισης των πρότερων γνώσεων, με ανάλογη προσαρμογή του χρονοπρογραμματισμού και των σχετικών σεναρίων διδασκαλίας. Το σενάριο διδασκαλίας πρέπει να διαθέτει χα-

ρακτηριστικά *διαφοροποιημένης διδασκαλίας,* προσφέροντας ποικιλία προτάσεων, με στόχο την επίτευξη, ανά μαθητή, υψηλού αποτελέσματος μάθησης.

Ενδεικτικός χρονοπρογραμματισμός *Οι δύο πρώτες στήλες είναι από το αναλυτικό πρόγραμμα*			
	Θ	Ε	Πρόταση
1. Από το πρόβλημα στην ανάπτυξη αλγόριθμου	1	-	Ε.Δ.Ε.Υ*
2. Ανάπτυξη προγράμματος	2	1	Ε.Δ.Ε.Υ*
3. Βασικά στοιχεία γλώσσας προγραμματισμού	3	1	6
4. Αλγοριθμικές Δομές	9	3	14
5 Κλασικοί αλγόριθμοι ΙΙ	9	3	15
6. Διαχείριση Αρχείων	3	1	8
7. Προηγμένα στοιχεία γλώσσας προγραμματισμού	12	3	20
8. Δομές δεδομένων ΙΙ	12	4	20
9. Εφαρμογές σε Γλώσσα Προγραμματισμού με χρήση API	4	4	Ε.Δ.Ε.Υ*
10 Βάσεις δεδομένων	6	2	Ε.Δ.Ε.Υ*
11. Αντικειμενοστρεφής Προγραμματισμός	9	3	17
12. Εισαγωγή στην Υπολογιστική Σκέψη	5	-	Ε.Δ.Ε.Υ*
Σύνολο ωρών Θ/Ε	**75**	**25**	**100**

Ε.Δ.Ε.Υ* Εκτός Διδακτέας Εξεταστέας Ύλης σύμφωνα με την ΥΑ Φ6/160716/Δ4 (2016).

Μέρος Ι

Κεφάλαια

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

Οδηγίες

Τα κεφάλαια 1 και 2, ως αναφέρθηκε σχετικά, είναι εκτός διδακτέας - εξεταστέας ύλης. Αναφορά στις έννοιες που περιέχονται σε αυτά προτείνεται να γίνει κατά τη διάρκεια επίλυσης ασκήσεων, για την καλύτερη κατανόησή τους, χωρίς αυτό να αποτελεί αντικείμενο αυτοτελούς εξέτασης.

Το κεφάλαιο 3 περιλαμβάνεται στη διδακτέα-εξεταστέα ύλη. Οι μαθητές έχουν διδαχθεί τις περισσότερες από τις έννοιες που περιέχει στη Β' τάξη, επιβάλλεται όμως να γίνει επανάληψη των εντολών, των συναρτήσεων, καθώς και των λογικών εκφράσεων που χρησιμοποιούνται στη δομή επιλογής και στις επαναληπτικές δομές, στη λογική της σπειροειδούς προσέγγισης και εμβάθυνσης.

Σε ότι αφορά στο κεφάλαιο 4 προτείνεται να δοθεί ιδιαίτερη έμφαση στα θεματικά αντικείμενα: **Δομές ακολουθίας, επιλογής και επανάληψης.** Τις δομές αυτές τις έχουν γνωρίσει οι μαθητές στη Β' τάξη, οπότε προτείνεται να γίνει επανάληψη-επικαιροποίηση και εμβάθυνση στις γνώσεις αυτές με αρκετά παραδείγματα για κάθε μία, ανάλογα με τις πρότερες γνώσεις των μαθητών. Προτείνεται να ακολουθηθεί προς τούτο διαδικασία διερεύνησης των γνώσεων αυτών, με ανάλογη προσαρμογή των σεναρίων διδασκαλίας. Προτείνεται να αναλυθεί και να εξηγηθεί με χρήση παραδειγμάτων, τόσο η χρήση εμφωλευμένων δομών, όσο και η χρησιμότητα του συνδυασμού των αλγοριθμικών δομών για την επίλυση προβλημάτων. Ειδικότερα θα πρέπει να δοθεί έμφαση στο συνδυασμό δομών επιλογής και επανάληψης.
Προτείνεται να γίνουν παραδείγματα της συνάρτησης range, ώστε οι μαθητές να μπορούν να υπολογίζουν τη λίστα που παράγει. Επίσης, προτείνεται η χρήση δομών επανάληψης, που να περιέχουν την εντολή print, με τις παραμέτρους συνάρτησης range να λείπουν. Οι μαθητές να συμπληρώνουν τις παραμέτρους, έτσι ώστε να εμφανίζονται συγκεκριμένες ακολουθίες αριθμών.

Χρονοπρογραμματισμός κεφαλαίου

Ενότητα	Ώρες
4.1 Αλγοριθμικές δομές - Ροές εκτέλεσης προγράμματος	8

Μέρος ΙΙ

Κεφάλαια

5. Κλασικοί Αλγόριθμοι ΙΙ
6. Διαχείριση Αρχείων
7. Προηγμένα στοιχεία γλώσσας προγραμματισμού
8. Δομές Δεδομένων ΙΙ
11. Αντικειμενοστρεφής Προγραμματισμός

5. Κλασικοί Αλγόριθμοι II

5.1 Χρονοπρογραμματισμός κεφαλαίου

Ενότητα	Ώρες
5.1 Δυαδική Αναζήτηση	8
5.2 Ταξινόμηση ευθείας ανταλλαγής	7

5.2 Δυαδική Αναζήτηση

Διδακτικές οδηγίες

1η ώρα – 2η ώρα

Η πρώτη ώρα ξεκινάει με τη δραστηριότητα “Βρες τον αριθμό”, όπως περιγράφεται στο σχολικό βιβλίο, όπου το σχέδιο δραστηριότητας παρουσιάζεται αναλυτικά στο ΤΕΜ. Την πρώτη ώρα προτείνεται οι μαθητές να εξοικειωθούν με τη συνάρτηση random.randint για την παραγωγή τυχαίων αριθμών. Στη συνέχεια οι μαθητές θα αναπτύξουν ένα πρόγραμμα-παιχνίδι στο οποίο θα έχουν απεριόριστες προσπάθειες να μαντέψουν τον αριθμό, που «σκέφτεται» ο υπολογιστής. Θα ήταν καλό να πειραματιστούν με το πρόγραμμα, για διάφορες τιμές των ορίων της συνάρτησης randint.

Αν κρίνετε ότι οι μαθητές σας θα δυσκολευτούν, μπορείτε να τους δώσετε το φύλλο εργασίας και να τους ζητήσετε να αντιγράψουν τον κώδικα στο IDLE και να συμπληρώσουν τα κενά.

Στο τέλος οι μαθητές μπορούν να προσθέσουν ένα μετρητή και μια συνθήκη, έτσι ώστε, αν ξεπεράσουν έναν αριθμό προσπαθειών χωρίς να βρουν τον αριθμό, να εμφανίζεται κατάλληλο μήνυμα, όπως φαίνεται στο παρακάτω πρόγραμμα:

```
# Ο παίκτης πρέπει να βρει έναν αριθμό από 1 έως και 20
#  τον οποίο έχει “σκεφτεί” ο  υπολογιστής.
# Αν δεν το βρει με το πολύ 10 προσπάθειες, χάνει.
import random
secret_number  =  random.randint(1,20)
guesses  =  0
found =  False
while  not  found   and   guesses < 10:
```

```
    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"
if   not  found :
    print "Χρησιμοποίησες και τις 10 προσπάθειές σου."
    print  "Ο αριθμός που ψάχνεις είναι ο ", secret_number
```

3η ώρα

Την επόμενη ώρα οι μαθητές επεκτείνουν/βελτιώνουν το πρόγραμμα που έχουν αναπτύξει, έτσι ώστε να τους παρέχει μια υπόδειξη μετά από κάθε δοκιμή. Εδώ εμφανίζεται για πρώτη φορά η ιδέα της δυαδικής αναζήτησης. Ο εκπαιδευτικός, θα μπορούσε αρχικά να παίξει ένα παιχνίδι *βρες τον αριθμό* με τους μαθητές, πριν ξεκινήσουν να υλοποιούν πρακτικά την ιδέα της δυαδικής αναζήτησης.

```
# Τώρα ο παίκτης λαμβάνει και μια υπόδειξη
# από τον υπολογιστή, αν ο αριθμός που  διάλεξε
# είναι μικρότερος ή μεγαλύτερος από το ζητούμενο.
import random
secret_number  =  random.randint(1,20)
guesses  =  0
found =  False
while  not  found   and   guesses < 10:
    guess = input("Μάντεψε τον αριθμό : ")
    guesses = guesses + 1
    if   guess == secret_number :
```

```
        print "Μπράβο το βρήκες με ", guesses, " προσπάθειες"
        found  = True
    elif   guess < secret_number :
        print "Είσαι χαμηλότερα "
    else :
        print "Είσαι υψηλότερα"
if   not  found :
    print "Χρησιμοποίησες και τις 10 προσπάθειές σου."
    print  "Ο αριθμός που ψάχνεις είναι ο ", secret_number
```

Αφού οι μαθητές υλοποιήσουν τον παραπάνω αλγόριθμο, προτείνεται να τον εκτελέσουν πολλές φορές παίζοντας το παιχνίδι με τον υπολογιστή, μέχρι να καταλάβουν με ποιον τρόπο θα βρουν το ζητούμενο αριθμό, με τις λιγότερες δυνατές προσπάθειες. Εδώ είναι σημαντικό να διαπιστώσουν ότι η επιλογή του μέσου του διαστήματος, είναι η καλύτερη στρατηγική. Προτείνεται ο εκπαιδευτικός να βοηθήσει και να διευκολύνει τους μαθητές με κατάλληλες ερωτήσεις καθοδήγησης. Επίσης πρέπει να τους τονιστεί ότι ο λόγος που μπορούν να αξιοποιήσουν τη στρατηγική αυτή, είναι η αξιοποίηση της διάταξης των αριθμών σε αύξουσα σειρά.

4η – 5η ώρα

Το επόμενο βήμα είναι να αλλάξει ο ρόλος του παίκτη. Τώρα ο υπολογιστής θα παίζει με τον εαυτό του. Εδώ καλείται ο μαθητής να υλοποιήσει σε Python τη στρατηγική που χρησιμοποίησε στο προηγούμενο βήμα για να βρει τον αριθμό. Τώρα θα προγραμματίσει τον υπολογιστή για να κάνει και αυτός το ίδιο. Αν οι μαθητές σας δυσκολεύονται να ξεκινήσουν, μπορείτε να τους δώσετε τον ημιτελή αλγόριθμο του φύλλου εργασίας, ώστε να συμπληρώσουν τα κενά. Στη συνέχεια θα "παίξουν" με τον υπολογιστή πολλούς γύρους, στους οποίους θα πρέπει να καταγράψουν σε κάθε περίπτωση, πόσες δοκιμές έκανε ο υπολογιστής μέχρι να βρει τον αριθμό.

Ακολούθως, μπορεί να γίνει μια "φιλοσοφική" συζήτηση για το κατά πόσο οι μαθητές έχουν αναπτύξει ένα πρόγραμμα που σκέφτεται

και που μπορεί πολύ γρήγορα να μαντέψει όποιον αριθμό σκεφτούν.

Το βήμα αυτό είναι το σημαντικότερο και το δυσκολότερο σε αυτήν τη δραστηριότητα, οπότε, αν ο καθηγητής το κρίνει αναγκαίο, μπορεί να διαθέσει άλλη μία ώρα για αυτό το βήμα της δραστηριότητας.

```
# 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 = raw_input("Είναι ο αριθμός ο   " + str(mid)
                                              + " ? (N/O)" )
   guesses = guesses + 1
   if   answer == "N"  :
     found = True
   else :
      answer = raw_input("Είναι μικρότερος του " + str(mid)
                                                  + " ? (N/O)" )
      if answer =="N"  :
        last = mid – 1
      else :
        first = mid + 1
if   found == True  :
   print  "Κέρδισα!!!  Το  βρήκα  με  ", guesses, "  προσπάθειες"
else :
   print "Κέρδισες"
```

6η ώρα

Ήρθε η ώρα να διατυπωθεί ο αλγόριθμος της δυαδικής αναζήτησης σε λίστα. Προτείνεται τα δεδομένα της λίστας να είναι ονόματα και

όχι αριθμοί, ώστε να μην υπάρξει σύγχυση με τους δείκτες του πίνακα που είναι αριθμοί. Με αυτόν τον τρόπο, οι μαθητές θα διακρίνουν καλύτερα την έννοια της θέσης/δείκτη ενός στοιχείου της λίστας από την τιμή του στοιχείου στη θέση αυτή.

Αρχικά μπορεί να γίνει μια συζήτηση με τους μαθητές για το πώς μπορούμε να αναζητήσουμε σε μια λίστα με ονόματα ένα συγκεκριμένο όνομα, ακολουθώντας την στρατηγική της δυαδικής αναζήτησης. Τώρα οι μαθητές καλούνται να υλοποιήσουν μια συνάρτηση *binary_search* που θα ψάχνει σε μια λίστα ένα στοιχείο. Αν δυσκολεύονται, μπορεί να τους δοθεί ο αλγόριθμος με μορφή άσκηση συμπλήρωσης κενών, πάντα με αναφορά σε αυτά που έχουν κάνει τις προηγούμενες ώρες.

7η – 8η ώρα

Στη συνέχεια μπορούν να γίνουν και κάποιες ασκήσεις στις οποίες ο μαθητής θα χρησιμοποιήσει τη συνάρτηση *binary_search* που έχει υλοποιήσει.

Είναι σημαντικό οι μαθητές να συγκρίνουν τον αλγόριθμο της δυαδικής αναζήτησης με αυτόν της σειριακής που έχουν διδαχθεί στη Β' τάξη όσον αφορά την απόδοση, έστω και εμπειρικά. Επίσης, θα πρέπει να τονιστεί ότι η δυαδική αναζήτηση μπορεί να χρησιμοποιηθεί μόνον, όταν τα στοιχεία της λίστας είναι διατεταγμένα, σε κάποια σειρά, φθίνουσα ή αύξουσα. Άρα από αυτό φαίνεται η ανάγκη της ταξινόμησης-οργάνωσης των δεδομένων μας σε κάποια σειρά. Αυτό είναι και το κίνητρο για την εισαγωγή των αλγορίθμων ταξινόμησης των επόμενων ενοτήτων: ότι δηλαδή χρειάζεται να ταξινομήσουμε τα δεδομένα μας, ώστε να μπορούμε να κάνουμε αποδοτική αναζήτηση.

5.3 Ταξινόμηση Ευθείας Ανταλλαγής

Διδακτικές οδηγίες

1η – 2η ώρα

Προτείνεται να παρουσιαστεί στους μαθητές η εισαγωγική δραστηριότητα του σχολικού βιβλίου μέσα από μικρές ασκήσεις, όπως είναι η περίπτωση του μαθητή που θέλει να μπει στη σωστή σειρά ενώ βρίσκεται στο τέλος της ουράς.

Ο αλγόριθμος της ευθείας ανταλλαγής πρέπει να παρουσιαστεί στους μαθητές σταδιακά και πιο συγκεκριμένα ξεκινώντας από την εσωτερική επανάληψη και εφαρμόζοντάς την πολλές φορές, έτσι ώστε να διαπιστώσουν ότι με κάθε εκτέλεση ανεβαίνει μία ακόμα “φυσαλίδα”. Για το σκοπό αυτόν μπορούν να αξιοποιηθούν διάφορες εφαρμογές οπτικοποίησης αλγορίθμων που υπάρχουν στο Διαδίκτυο, όπως:

- http://www.cs.armstrong.edu/liang/animation/web/BinarySearch.html
- https://www.cs.usfca.edu/~galles/visualization/Search.html

Μια σημαντική παρατήρηση εδώ είναι πως, αν για παράδειγμα θέλουμε μόνο τους 4 καλύτερους, δε χρειάζεται να ταξινομήσουμε όλη τη λίστα, αλλά αρκεί να γίνουν μόνο 4 περάσματα, κάτι το οποίο είναι ένα από τα πλεονεκτήματα της ταξινόμησης ευθείας ανταλλαγής.

Άσκηση που θα μπορούσε να δοθεί στους μαθητές, είναι να μετρήσουν το πλήθος των αντιμεταθέσεων και των συγκρίσεων για διάφορα είδη δεδομένων εισόδου.

Τα δεδομένα προς ταξινόμηση δε χρειάζεται να είναι μόνον αριθμητικά, αλλά μπορεί να είναι οποιουδήποτε είδους, όπως αλφαριθμητικά ή ακόμα και λογικές τιμές.

3η – 5η ώρα

Εδώ, ο εκπαιδευτικός μπορεί να χρησιμοποιήσει και διάφορες εφαρμογές οπτικοποίησης αλγορίθμων, όπως για παράδειγμα είναι το visualgo (2016), για να δείξει τη λειτουργία του αλγορίθμου.

Να λάβουμε υπόψη ότι η οπτικοποίηση μπορεί να βοηθήσει στα παρακάτω:

- Να κατανοήσουν οι μαθητές ότι αν έχουμε N αριθμούς, χρειαζόμαστε N–1 περάσματα.
- Σε κάθε πέρασμα δε χρειάζεται να φτάσει η φυσαλίδα μέχρι πάνω, αλλά μόνο μέχρι την αρχή του ταξινομημένου μέρους του πίνακα.

Οι μαθητές είναι καλό να εξασκηθούν αναπτύσσοντας ένα πρόγραμμα το οποίο εμφανίζει τη λίστα που ταξινομείται σε κάθε πέρασμα, όπως φαίνεται παρακάτω, όπου δίνουμε και έναν τρόπο γραφής του αλγορίθμου λίγο διαφορετικό από αυτόν του βιβλίου:

```
#Ταξινομεί, με τον αλγόριθμο της ευθείας ανταλλαγής, τη λίστα A
def bubbleSort( A ):
    N = len( A )
    for i in range( N ):                # από i=0 μέχρι i=N-1
        for j in range(N-1, i, -1):     # από j=N-1 μέχρι i+1
            if A[ j ] < A[ j-1 ] :
                A[ j ], A[ j-1 ] = A[ j-1 ], A[ j ]
        print A             # Εμφάνιση της λίστας σε κάθε πέρασμα
```

2ος τρόπος

```
def bubbleSort( A ):
    N = len( A )
    for i in range( 1, N ):
        for j in range(N-1, i-1, -1):
            if A[ j ] < A[ j-1 ] :
                A[ j ], A[ j-1 ] = A[ j-1 ], A[ j ]
        print A
```

Εκτός από την οπτικοποίηση του αλγορίθμου θα μπορούσε να γίνει και διερεύνηση, έτσι ώστε οι μαθητές να βλέπουν όλες τις τιμές i, j

σε κάθε βήμα, για να διαπιστώσουν μόνοι τους τι πρέπει να μπει στα άκρα των επαναλήψεων. Αυτό μπορεί να γίνει με κατάλληλες εντολές εκτύπωσης των i,j, όπως φαίνεται στην παρακάτω συνάρτηση:

```
def bubbleSort( A ):
    N = len( A )
    for i in range( N ):
        print  " i = " , i
        print  " j = " , j
        for j in range(N-1, i, -1):
            print  " j = " , j, " j-1 = " ,   j–1,  ",  " ,
            if A[ j ] < A[ j-1 ] :
                A[ j ], A[ j-1 ] = A[ j-1 ], A[ j ]
        print
```

Υπενθυμίζεται ότι αν θέλουμε η print μετά την εμφάνιση του αποτελέσματος να μην αλλάξει γραμμή, αλλά να παραμείνει στην ίδια, βάζουμε ένα κόμμα στο τέλος της εντολής.

Δοκιμάστε να τρέξετε την παραπάνω συνάρτηση για διάφορες τιμές που βρίσκονται στα όρια του πεδίου των επαναλήψεων. Σε κάποιες περιπτώσεις μπορεί να δοθούν σκόπιμα λάθος αριθμοί και να ζητηθεί από τους μαθητές η διόρθωση των πεδίων.

Προσοχή! Υπενθυμίζεται ότι στην Python οι λίστες μπορούν να έχουν και αρνητική δεικτοδότηση, οπότε το A[–1], δεν είναι λάθος, αλλά αναφέρεται στο τελευταίο στοιχείο της λίστας!

6η – 7η ώρα

Τις τελευταίες δύο ώρες μπορούν να δοθούν στους μαθητές πραγματικά προβλήματα στα οποία να χρειαστεί να χρησιμοποιήσουν τον αλγόριθμο ταξινόμησης της ευθείας ανταλλαγής σε συνδυασμό με τον αλγόριθμο της δυαδικής αναζήτησης. Εδώ οι μαθητές θα πρέπει να αξιοποιήσουν τα πλεονεκτήματα του τμηματικού προγραμματισμού, έτσι ώστε να μη χρειαστεί να ξαναγράφουν για κάθε

άσκηση τον ίδιο αλγόριθμο, παρά μόνο αν χρειάζεται κάποια παραλλαγή του, όπως φαίνεται στο επόμενο πρόβλημα.

5.4 Απαντήσεις Δραστηριοτήτων κεφ. 5 ΒΜ

Δραστηριότητα 1

Να αντιστοιχίσετε τους παρακάτω αλγόριθμους με τις κατάλληλες λειτουργίες:

Αλγόριθμος	Λειτουργία
1.Ταξινόμηση με επιλογή	Α. Αντιμετάθεση ζευγών
2. Ταξινόμηση ευθείας ανταλλαγής	Β. Τοποθέτηση στοιχείου σε ταξινομημένη λίστα
3. Ταξινόμηση με εισαγωγή	Γ. Εύρεση ελαχίστου

Ενδεικτική Λύση

Η αντιστοίχιση έχει ως εξής:

1Γ. (Ταξινόμηση με επιλογή - Εύρεση Ελαχίστου)

2Α. (Ταξινόμηση ευθείας ανταλλαγής - Αντιμετάθεση ζευγών)

1Α. (Ταξινόμηση με εισαγωγή - Τοποθέτηση στοιχείου σε ταξινομημένη λίστα)

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;

Ενδεικτική Λύση

Αν αλλάξουμε μόνο τον τελεστή σύγκρισης στην findMinPosition από "**<**" σε "**>**", ώστε να βρίσκει το μεγαλύτερο αντί το μικρότερο, τότε και η συνάρτηση selectionSortAscending θα ταξινομεί σε φθίνουσα αντί σε αύξουσα σειρά.

```
def findMinPosition(start, end, List):
    position = start
    for i in range(start, end):
        if List[ i ] > List[ position ] :
            position = i
     return position

def selectionSortAscending(List):
    position = None
    n = len(List)
    for i in range(0,n):
        position = findMinPosition(i, n, List)
        List[ i ], List[ position ] = List[ position ], List [ i ]
    return List
```

Εδώ, αλλάξαμε τη λειτουργία της selectionSortAscending χωρίς να πειράξουμε κάτι στον κώδικά της. Αυτό είναι ένα παράδειγμα της ανεξαρτησίας και της ευελιξίας του *τμηματικού προγραμματισμού*.

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.

Ενδεικτική Λύση

Η λογική μεταβλητή ascending παραμένει True, όσο δε βρίσκουμε ένα ζευγάρι στοιχείων για το οποίο δεν ισχύει η σχέση που θέλουμε. Δηλαδή ψάχνουμε ένα ζευγάρι που να είναι σε φθίνουσα σειρά. Αν βρούμε ένα τουλάχιστον, δεν έχει πλέον νόημα να συνεχίσουμε.

```
# Με τη χρήση λογικής μεταβλητής
```

```
def isAscending(myList):
    ascending = True
    i = 0
    N = len(myList)
    while ascending and i<N-1 :
        if (myList[ i ] > myList[ i+1 ]) :
            ascending = False
        i = i +1
    return ascending
```

```
# Χωρίς τη χρήση λογικής μεταβλητής
def isAscending2(myList):
    i = 0
    pred = myList[ 0 ]
    for item in myList :
        if item < pred:
            return False
        else:
            pred = item
    return True
```

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 δε βρίσκονται στην επιθυμητή σειρά, καθώς η “φυσαλίδα ανεβαίνει στην επιφάνεια”.

Ενδεικτική Λύση

Χρησιμοποιούμε την ιδέα, όπου η λογική μεταβλητή isSorted μας δίνει την πληροφορία, αν η λίστα είναι ταξινομημένη στο τέλος κάθε περάσματος. Αν γίνει έστω και μία αντιμετάθεση, η isSorted θα γίνει False. Αν δε γίνει αντιμετάθεση, σημαίνει ότι όλα τα στοιχεία είναι στη σωστή σειρά, άρα ταξινομημένα, οπότε ο αλγόριθμος δεν έχει λόγο να συνεχίσει.

```
# 1ος τρόπος, με λογική μεταβλητή
def optimizedBubbleSort( A ):
    N = len( A )
    isSorted = False
    i=1
    while i < N and not isSorted :
        isSorted = True
        for j in range(N-1, i-1, -1):
            if A[ j ] < A[ j-1 ]:
                A[ j ], A[ j-1 ] = A[ j-1 ], A[ j ]
                isSorted = False
        i = i + 1
```

```
# 2ος τρόπος, με βίαιη διακοπή της επανάληψης for...in..
def optimizedBubbleSort2( A ):
    N = len( A )
    for i in range( N ):
        isSorted = True
        for j in range(N-1, i, -1):
            if A[ j ] < A[ j-1 ] :
                A[ j ], A[ j-1 ] = A[ j-1 ], A[ j ]
                isSorted = False
        if isSorted:
```

```
        return
```

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.

Ενδεικτική Λύση

Ο πρώτος, πιο απλός και γρήγορος τρόπος, είναι να μετρήσουμε πόσα είναι τα True. Αν είναι έστω k, οπότε γνωρίζουμε το μέγεθος (len) της λίστας, μπορούμε να βρούμε και τα False (μέγεθος – k). Στη συνέχεια, θέτουμε True στα πρώτα k στοιχεία και False στα υπόλοιπα.

```
#1ος τρόπος: Μετράμε πόσα είναι τα True :
def count_true_values( booleanList ) :
    true_values = 0
    for item in booleanList :
        if item :                          # ή if item == True :
            true_values = true_values + 1
    return true_values

# 2ος τρόπος (True=1,False=0). Μετράμε πόσα είναι τα True
def count_true_values2( booleanList ) :
    true_values = 0
    for item in booleanList :
        true_values = true_values + item
    return true_values

# 1ος τρόπος: Μετράμε πόσα είναι τα True και στη συνέχεια
# βάζουμε τόσα True στα πρώτα στοιχεία του πίνακα και στα
```

```
 # υπόλοιπα False.
def swapBooleanbyCounting( List ):
    N = len( List )
    true_values = count_true_values( List )
    for i in range(true_values):
        List[ i ] = True
    for i in range(true_values, N):
        List[ i ] = False
    return List
```

Στο δεύτερο τρόπο ξεκινάμε με δύο δείκτες στα άκρα της λίστας και αντιμεταθέτουμε τα αντιδιαμετρικά στοιχεία (True, False), μέχρι να συναντήσουμε από αριστερά False ή από δεξιά True. Οι δείκτες κινούνται ταυτόχρονα και αντίθετα. Όταν συναντηθούν, έχουμε αντιμεταθέσει όλες τις τιμές που ήταν στη λάθος πλευρά.

```
def swapBooleanbyComparison( List ):
    N = len( List )
    left = 0
    right = N-1
    while left < right:
        if (List[ right ]==True and List[ left ]==False ) :
            List[ left ],List[ right ] = List[ right ],List[ left ]
            left = left+1
            right = right-1
        elif (List[ right ]==False):
            right = right-1
        else :
            left = left+1
    return List
```

Δραστηριότητα 6

Να γράψετε ένα πρόγραμμα σε Python το οποίο θα δέχεται μια λίστα με λογικές τιμές True/False και στη συνέχεια θα καλεί τη 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.

Ενδεικτική Λύση

Αρχικά σχεδιάζουμε μια συνάρτηση για να διαβάζουμε μια λίστα, η οποία θα μας χρειαστεί και σε επόμενες δραστηριότητες. Η συνάρτηση διαβάζει από το πληκτρολόγιο μια λίστα από αντικείμενα, ενώ η ανάγνωση της λίστας σταματάει, όταν δοθεί η τιμή None και ακολούθως επιστρέφει τη λίστα ως τιμή της συνάρτησης. Αν θέλουμε να εισάγουμε αλφαριθμητικά, πρέπει να τα εισάγουμε ανάμεσα σε εισαγωγικά (quotes), για παράδειγμα "Γυμνάσιο". Μπορούμε να διαπιστώσουμε ότι η συνάρτηση δουλεύει για κάθε τύπο, κάτι που αποτελεί ένα από τα ισχυρά πλεονεκτήματα της Python.

```
def readList( ) :
    print "*****************************************"
    print "* Για το τέλος δώσε την τιμή None *"
    print "*****************************************"
    index = 0
    L = [ ]
    value = input("L[" + str(index) + "] = ")
    while value != None :
        L.append(value)
        index += 1
        value = input("L[" + str(index) + "] = ")
    return L
```

Μετράμε πόσα είναι τα True, διπλασιάζουμε το ποσό και το συγκρίνουμε με το πλήθος των στοιχείων της λίστας. Έτσι βρίσκουμε

ποια τιμή εμφανίζεται τις λιγότερες φορές. Αυτό θα είναι και το πλήθος των ζευγών (True, False).

```
def program6_Counting(  ):
    List = readList( )
    N = len( List )

    # μετράει πόσα είναι τα True (ορίστηκε στη δραστηριότητα 5)
    true_values = count_true_values( List )

   # είναι τα True περισσότερα από τα False;
    if (2*true_values < N):
        minValue = True
        couples = true_values
    else:
        minValue = False
        couples = N – true_values
    for i in range(0, 2*couples, 2):
        List[ i ] = True
        List[ i+1 ] = False
    for i in range(2*couples, N):
        List[ i ] = not minValue
    return List
```

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.

Ενδεικτική Λύση

1ος τρόπος: Αφού οι τιμές True, False είναι μισές-μισές, τότε θέτουμε στις πρώτες μισές θέσεις True και στις υπόλοιπες False.

```
  # 1ος τρόπος: θέτουμε τα πρώτα N/2 True
  # και τα επόμενα N/2 False.
def splitBoolean_byCounting( List ):
    mid = len(List) / 2
    for index in range(mid):
        List[ index ] = True
        List[ index + mid ] = False
    return List
```

2ος τρόπος: Αρκούν N/2 περάσματα του αλγορίθμου ταξινόμησης.

```
def splitBoolean_bySorting( List ):
    N = len( List )
    mid = N / 2
    for i in range(mid):
        for j in range(N-1, i, -1) :
            if   List[ j ]  >  List[ j-1 ] :          # True > False
                List[ j-1 ], List[ j ] = List[ j ],List[ j-1 ]
    return List
```

Δραστηριότητα 8

Το πρόβλημα της ολλανδικής σημαίας αναφέρεται στην αναδιάταξη μιας λίστας γραμμάτων, η οποία περιέχει μόνον 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 αλγόριθμους ταξινόμησης που παρουσιάστηκαν σε αυτήν την ενότητα, ώστε να επιλύει αυτό το πρόβλημα.

Ενδεικτική Λύση

1ος τρόπος: Αρκεί να αλλάξουμε τη συνθήκη του αλγόριθμου ταξινόμησης ευθείας ανταλλαγής, έτσι ώστε να ισχύει η διάταξη RWB. Έτσι σχηματίζουμε συνθήκη με όλες τις περιπτώσεις. Το σκεπτικό είναι ότι η αντιμετάθεση μεταξύ δυο στοιχείων θα γίνει, αν δεν ισχύει η σειρά R W B.

```
def dutchFlag( L ):
    N  = len( L )
    for i in range( N ):
        for j in range(N-1, i, -1) :
            if  (L[ j ]=='W' and L[ j-1 ]=='B')          \
                or (L[ j ]=='R' and L[ j-1 ]=='W')      \
                or (L[ j ]=='R' and L[ j-1 ]=='B') :
                  L[ j-1 ], L[ j ] = L[ j ], L[ j-1 ]
    return L
```

Σημείωση: Αν μια εντολή δε χωράει σε μια γραμμή και θέλουμε να συνεχιστεί στην επόμενη, χρησιμοποιούμε το σύμβολο **\ .**

2ος τρόπος (Μόνο για ενημέρωση του εκπαιδευτικού): Όμοια με πριν, μόνο που τώρα ορίζουμε μια αντιστοίχιση των γραμμάτων R, W, B με τους αριθμούς 1, 2, 3, έτσι ώστε να ισχύει η διάταξη που θέλουμε, δηλαδή code(R) < code(W) < code(B). Θα ήταν πιο απλό, αν χρησιμοποιούσαμε λεξικό.

Το λεξικό θα μπορούσε να οριστεί με τις παρακάτω εντολές:

```
code =  dict( )
code['R'] = 1  ;  code['W'] = 1  ;  code['B'] = 1
```

Όμως, εμείς θα χρησιμοποιήσουμε την παρακάτω συνάρτηση:

```
# Μόνο μία από τις συνθήκες μπορεί να είναι True (1),
# οι άλλες θα είναι False (0)
def code(letter):
    return (letter=='R') + (letter=='W')*2 + (letter=='B')*3

def dutchFlag2( L ):
    N  = len( L )
    for i in range( N ):
        for j in range(N-1, i, -1) :
            if code( L[ j ] ) < code( L[ j-1 ] ) :
                L[ j-1 ], L[ j ] = L[ j ], L[ j-1 ]
    return L
```

Αν χρησιμοποιούσαμε το λεξικό, η συνθήκη θα γινόταν:

```
code( L[ j ] ) < code( L[ j-1 ] )
```

3ος τρόπος: Ο πιο γρήγορος αλγόριθμος που διατρέχει μία φορά τη λίστα. Χρησιμοποιεί τρεις δείκτες lo, mid, hi, έναν για κάθε χρώμα. Ο **lo** βρίσκεται πάντα αριστερά, ο **hi** δεξιά και ο **mid** στη μέση. Έτσι, όλα εξαρτώνται από το στοιχείο που είναι στη μέση. Για παράδειγμα, αν βρούμε R στη μεσαία θέση (mid), εκτελούμε αντι-μετάθεση με τον αριστερό δείκτη (lo), σε κάθε περίπτωση.

```
def dutchFlag( L ):
    hi  = len( L ) – 1
    lo = mid = 0
```

```
while mid <= hi :
    if  L[ mid ] == 'R' :
        L[ lo ], L[ mid ] = L[ mid ], L[ lo ]
        lo = lo + 1
        mid = mid + 1
    elif L[ mid ] == 'B' :
        L[ hi ], L[ mid ] = L[ mid ], L[ hi ]
        hi = hi - 1
    else :
        mid = mid + 1
    return L
```

Ερώτηση για συζήτηση: Αν ο χρήστης δώσει αλφαριθμητικό, αντί για λίστα, λειτουργεί ο αλγόριθμος; Γιατί; Μπορείτε να το διορθώσετε;

Δραστηριότητα 9

Να γράψετε μια συνάρτηση σε Python η οποία διαβάζει αριθμούς από το χρήστη και στη συνέχεια τους τοποθετεί σε μια λίστα σε φθίνουσα σειρά, την οποία και επιστρέφει. Κάθε φορά που διαβάζει ένα νέο αριθμό, τον τοποθετεί στη σωστή θέση στην ήδη ταξινομημένη λίστα, ώστε να διατηρείται η φθίνουσα διάταξη των στοιχείων της λίστας. Η εισαγωγή των αριθμών σταματάει, όταν δοθεί η τιμή None. Ποιον αλγόριθμο ταξινόμησης σας θυμίζει η παραπάνω λειτουργία; Σε τι διαφέρει η συνάρτηση που θα αναπτύξετε από τον αλγόριθμο αυτόν;

Ενδεικτική Λύση

Θέλουμε έναν αλγόριθμο ταξινόμησης ο οποίος να ταξινομεί σταδιακά ένα μέρος του πίνακα (incremental). Δε θέλουμε κάθε φορά που έρχεται ένα νέο στοιχείο, να εκτελούμε πάλι ταξινόμηση όλου του πίνακα, αλλά να τοποθετούμε το νέο στοιχείο στη σωστή θέση, έτσι ώστε ο πίνακας να παραμένει ταξινομημένος. Αυτή είναι η βασική ιδέα του αλγορίθμου ταξινόμησης με εισαγωγή. Κάθε φορά

που διαβάζουμε ένα νέο αριθμό, εκτελούμε αναζήτηση για να βρούμε τη θέση που πρέπει να εισαχθεί και παράλληλα μετακινούμε μια θέση δεξιά, όσα στοιχεία πρέπει να βρίσκονται μετά από αυτόν.

```
def insertionSort( ) :
    print "*Δώσε την τιμή None για να σταματήσεις*"
    number = input("number = ")
    L = [ ]
    while (number is not None) :    # number != None
        L.append(number)
        value = L[ len(L) - 1 ]
        j = len(L) – 1
        while j > 0 and L[ j-1 ] < value :
            L[ j ] = L[ j-1 ]
            j = j-1
        L[ j ] = value    # έξω από τη while
        number = input("number = ")

    return L
```

Δραστηριότητα 10

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

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

Ενδεικτική Λύση

Αρχικά, το πρόγραμμα διαβάζει δύο λίστες με τη συνάρτηση readList που έχουμε ορίσει σε προηγούμενη δραστηριότητα. Στη

συνέχεια, εφαρμόζουμε έναν αλγόριθμο ταξινόμησης, ώστε να μπορεί να χρησιμοποιηθεί η δυαδική αναζήτηση μετά, για τον έλεγχο της ύπαρξης ενός αριθμού της λίστας Β στη λίστα Α. Για την καλύτερη οργάνωση του προγράμματος, αναπτύξαμε κάποιες συναρτήσεις.

Αρχικά ορίζουμε την ταξινόμηση ευθείας ανταλλαγής:

```
# Ταξινομεί με τον αλγόριθμο της ευθείας ανταλλαγής τη λίστα A
def bubbleSort( A ):
    N = len( A )
    for i in range( N ):
        for j in range(N-1, i, -1):
            if A[ j ] < A[ j-1 ] :
                A[ j ], A[ j-1 ] = A[ j-1 ], A[ j ]
```

Ορίζουμε τη συνάρτηση που υλοποιεί τη δυαδική αναζήτηση:

```
# Ελέγχει την ύπαρξη ενός στοιχείου key στην ταξινομημένη
# λίστα L υλοποιώντας τον αλγόριθμο της δυαδικής αναζήτησης
def binarySearch( A, key ):
    last = len( A ) - 1
    first = 0
    found = False
    while first <= last and not found:
        mid = (last + first) // 2
        if A[ mid ] == key :
            found = True
        elif A[ mid ] < key :
            first = mid + 1
        else:
            last = mid - 1
```

```
    return found
```

Στη συνέχεια παραθέτουμε το πρόγραμμα που χρησιμοποιεί τις παραπάνω συναρτήσεις. Παρατηρήστε ότι αυτό δε δουλεύει μόνο για αριθμούς. Δουλεύει επίσης για αλφαριθμητικά, αλλά και για λογικές τιμές. Δοκιμάστε να εκτελέσετε το παραπάνω πρόγραμμα και με άλλους τύπους δεδομένων, όπως με αλφαριθμητικά. Αυτό για να εξοικειωθούν οι μαθητές με την ιδέα ότι στην Python έχουμε τη δυνατότητα να ορίζουμε τη λειτουργία του αλγόριθμου ανεξάρτητα από το είδος των δεδομένων. Δηλαδή δε χρειάζεται να ορίσουμε διαφορετικό αλγόριθμο για αλφαριθμητικά, διαφορετικό για αριθμούς κ.ο.κ.

```
def program_10( ):

    A = readList( )
    B = readList( )
    bubbleSort( A )

    count = 0
    for item in B :
        if binarySearch( A, item ) :
            count = count + 1

    print "Πλήθος κοινών στοιχείων = ", count
```

6. Διαχείριση Αρχείων

6.1 Οδηγίες

Μέσα από τη διδασκαλία του κεφαλαίου, οι μαθητές θα πρέπει να κατανοήσουν ότι τα δεδομένα που επεξεργάζεται ένα πρόγραμμα, πολύ σπάνια τα δίνει ο χρήστης την ώρα που αυτό εκτελείται. Το πρόγραμμα διαβάζει τα δεδομένα από κάπου, με την πιο συνηθισμένη πηγή να είναι ένα αρχείο.

Για το λόγο αυτό, ιδιαίτερη έμφαση θα πρέπει να δοθεί στα παρακάτω θεματικά αντικείμενα:

- Άνοιγμα ενός αρχείου κειμένου και ανάγνωση δεδομένων από αυτό.
- Άνοιγμα ενός αρχείου κειμένου και εγγραφή δεδομένων σε αυτό.
- Δημιουργία ενός αρχείου για εγγραφή ή και ανάγνωση.
- Εμφάνιση των περιεχομένων ενός αρχείου κειμένου.

Θα πρέπει να γίνουν αρκετές δραστηριότητες, όπου οι μαθητές θα εισάγουν δεδομένα από ένα αρχείο και θα αποθηκεύουν αποτελέσματα σε ένα άλλο.

Σκοπός του κεφαλαίου δεν είναι η εφαρμογή προηγμένων επεξεργασιών πάνω σε ένα αρχείο.

6.2 Λύση Δραστηριοτήτων

Η απάντηση στις δραστηριότητες 1 και 2 αναφέρεται στο Βιβλίο Μαθητή.

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.

Ενδεικτική Λύση

```
f = open ('demoFile.txt', 'r')
for line in (open('demoFile.txt').readlines()):
    print line
final = [ ]
for line in f :
    final.append(line)
final.reverse()
for line in final :
    print line
fout = open ('output.txt', 'w')
for line in final :
    fout.write (line)
f.close()
fout.close()
```

Δραστηριότητα 4

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

Ενδεικτική Λύση

```
# Κατασκευή λίστας με τα τετράγωνα των αριθμών από 1 έως 10
my_list = [i**2 for i in range(1,11)]

# Άνοιγμα αρχείου κειμένου για εγγραφή
f = open('output.txt', 'w')

# Εγγραφή  των στοιχείων της λίστας στο αρχείο
for item in my_list:
    f.write(str(item) + '\n') # δέχεται string όρισμα η write
f.close()
```

```
# Άνοιγμα του αρχείου για ανάγνωση και εκτύπωση των
# περιεχομένων του
f = open('output.txt', 'r')
print f.readline()
print f.read()
f.close()
```

7. Προηγμένα στοιχεία γλώσσας προγραμματισμού

7.1 Οδηγίες

Μέσα από τη διδασκαλία του κεφαλαίου οι μαθητές θα πρέπει να κατανοήσουν τη χρησιμότητα και τον τρόπο χρήσης των συναρτήσεων στη δημιουργία ενός σύνθετου προγράμματος στην Pyhton.

Θα πρέπει όμως, να τονιστεί στους μαθητές ότι δεν μπορούν να χρησιμοποιούν οποιαδήποτε βιβλιοθήκη ή οποιαδήποτε συνάρτηση της Python που υπάρχει, αλλά να ακολουθούν τις οδηγίες της εκφώνησης της άσκησης. Οι συναρτήσεις που υπάρχουν στα σχολικά βιβλία του μαθήματος, δίνονται στις γενικές οδηγίες του παρόντος βιβλίου. Επίσης, πρέπει να δίνεται ιδιαίτερη προσοχή στην κατανόηση και στην ορθή χρήση του τρόπου που οι τιμές των μεταβλητών του προγράμματος αναγνωρίζονται ή τροποποιούνται μέσα σε μια συνάρτηση, καθώς και η χρήση των παραμέτρων της.

7.2 **Δραστηριότητες** (Πρόσθετες του ΒΜ)

Δραστηριότητα 1

Ανοίγουμε ένα νέο αρχείο στην Python στο οποίο θα προσθέσουμε τους ορισμούς των συναρτήσεων που θα δώσουμε παρακάτω:

Ορίζουμε τη συνάρτηση python3 η οποία εμφανίζει τη λέξη python 3 φορές. Επίσης, ορίζουμε και τη συνάρτηση python9 που εμφανίζει τη λέξη python 9 φορές.

```
def printPython3( ):
    print 'python'
    print 'python'
    print 'python'
>>> printPython3()
python
python
python
```

```
def printPython9( ):
    print 'python'
    print 'python'
    print 'python'
    print 'python'
    print 'python'
    print 'python'
    print 'python'
    print 'python'
    print 'python'
```

Μπορείτε να ξαναγράψετε τη συνάρτηση printPython9 χρησιμοποιώντας λιγότερες εντολές;

Να ορίσετε μια συνάρτηση η οποία να εμφανίζει 21 φορές τη λέξη python με αποκλειστική χρήση των δύο παραπάνω συναρτήσεων, χρησιμοποιώντας όσο το δυνατόν λιγότερες εντολές.

Ενδεικτική λύση

```
def printPython9( ):          def printPython21( ) :
    printPython3( )               printPython9( )
    printPython3( )               printPython9( )
    printPython3( )               printPython3( )
```

Δραστηριότητα 2

Δίνονται οι παρακάτω συναρτήσεις σε python

```
def printPython3( ):          def printPython12():
   for i in range(3):            for i in range(12):
      print 'python'                  print 'python'
```

Συμπληρώστε τον ορισμό της παρακάτω συνάρτησης, ώστε να αποτελεί γενίκευση των προηγουμένων και στη συνέχεια, δώστε τις διπλανές κλήσεις στο διερμηνευτή:

```
def printPython( _______ ):        >>> printPython(3)
   for i in range(_____):          >>> printPython(9)
      print 'python'              >>> printPython(21)
                                  >>> printPython(100)
```

Σε τι διαφέρει η τελευταία συνάρτηση από όλες τις προηγούμενες; Ποια είναι η σχέση της με αυτές;

Ενδεικτική λύση

```
def printPython( N ):
   for i in range( N ):
      print 'python'
```

Η τελευταία συνάρτηση αποτελεί γενίκευση όλων των προηγούμενων. Για N=3 το αποτέλεσμα είναι ίδιο με αυτό της printPython3, για N=9 ίδιο με αυτό της printPython9 κ.ο.κ. Το N είναι η παράμετρος

που ορίζει κάθε φορά πόσα μηνύματα θέλουμε να εμφανίσουμε στην οθόνη.

Δραστηριότητα 3

Δίνεται το παρακάτω πρόγραμμα. Να εντοπίσετε τα τμήματα κώδικα που πρέπει να γίνουν υποπρογράμματα, έτσι ώστε να αποφευχθεί η επανάληψη τμημάτων κώδικα που επιτελούν την ίδια λειτουργία. Στη συνέχεια να ξαναγράψετε το παρακάτω πρόγραμμα με τη χρήση υποπρογράμματος.

```
sum1 = 0
for i in range(100):
  sum1 = sum1 + i
print sum1
sum2 = 0
for j in range(200):
  sum2 = sum2 + j
print sum2 + sum1
sum3 = 0
for k in range(sum1):
  sum3 = sum3 + k
print sum3 + sum2 + sum1
```

Ενδεικτική λύση

Παρατηρούμε ότι ένα τμήμα κώδικα επαναλαμβάνεται τρεις φορές:

sum1 = 0
for i in range(100):
  sum1 = sum1 + i
	sum2 = 0
	for j in range(200):
	  sum2 = sum2 + j
		sum3 = 0
		for k in range(sum1):
		  sum3 = sum3 + k

Η λειτουργία που επιτελείται είναι, ακριβώς, ο υπολογισμός του αθροίσματος μιας σειράς αριθμών. Το μόνο που αλλάζει είναι τα ονόματα των μεταβλητών και το πλήθος των αριθμών. Μπορούμε

όμως, να αλλάξουμε τα ονόματα των μεταβλητών και να πάρουμε σε κάθε περίπτωση το ίδιο αποτέλεσμα:

```
s = 0
for i in range(100):
  s = s + i
```

```
s = 0
for i in range(200):
  s = s + i
```

```
s = 0
for i in range(sum1):
  s = s + i
```

Το μόνο που αλλάζει τώρα, είναι το πλήθος των αριθμών που θέλουμε να προσθέσουμε. Αυτό θα είναι η παράμετρος στη συνάρτηση που θα σχεδιάσουμε:

```
def sum( N ):
   s = 0
   for i in range( N ):
      s = s + i
   return s
```

Τώρα το αρχικό μας πρόγραμμα παίρνει την παρακάτω μορφή:

```
sum1 = sum( 100 )
print sum1
sum2 = sum( 200 )
print sum2 + sum1
sum3 = sum( sum1 )
print sum3 + sum2 + sum1
```

8. Δομές Δεδομένων ΙΙ

8.1 Ενδεικτικός χρονοπρογραμματισμός κεφαλαίου

Ενότητα	Ώρες
Συμβολοσειρές (strings)	6
Λίστες	8
Στοίβα	3
Ουρά	3

8.2 Διδακτικές οδηγίες

Συμβολοσειρές. Διδακτική προσέγγιση

Υπενθυμίζουμε στους μαθητές τα βασικά χαρακτηριστικά των αλφαριθμητικών / συμβολοσειρών που έχουν διδαχθεί στη Β' τάξη στο μάθημα Αρχές Προγραμματισμού Υπολογιστών. Μέσα από παραδείγματα στο διερμηνευτή της Python, προτείνεται οι μαθητές να εξοικειωθούν με τις παρακάτω βασικές ιδιότητες των αλφαριθμητικών:

- Τα αλφαριθμητικά έχουν σταθερό μέγεθος. Δεν μπορούμε ούτε να προσθέσουμε ούτε να αφαιρέσουμε χαρακτήρες.
- Δεν μπορούμε να τροποποιήσουμε έναν χαρακτήρα σε μία συγκεκριμένη θέση του αλφαριθμητικού.
- Η αρίθμηση του δείκτη των στοιχείων ενός αλφαριθμητικού ξεκινάει από το 0 (και όχι από το 1), δηλαδή αν p = "Pyhton" το p[0] είναι το γράμμα 'P'.
- Για την αρίθμηση των στοιχείων ενός αλφαριθμητικού υπάρχει και αρνητική δεικτοδότηση, όπου το τελευταίο στοιχείο αντιστοιχεί στο -1 , το προτελευταίο στο -2 κλπ.

Οι βασικές λειτουργίες των συμβολοσειρών, στις οποίες πρέπει να επικεντρωθούμε, είναι οι εξής:

- Συνένωση συμβολοσειρών σε μια νέα με τον τελεστή '**+**'
- Μετατροπή μιας τιμής σε συμβολοσειρά με τη συνάρτηση **str**.

- Υπολογισμός του μεγέθους μιας συμβολοσειράς με τη συνάρτηση **len**.
- Αντίστροφα με τη συνάρτηση **int**, αν έχουμε έναν αριθμό με τη μορφή συμβολοσειράς, μπορούμε να τον μετατρέψουμε σε αριθμητική τιμή, ώστε να μπορεί να συμμετέχει σε αριθμητικές πράξεις.

Για παράδειγμα, μεταξύ δύο συμβολοσειρών ο τελεστής «+» συνενώνει τις δύο συμβολοσειρές σε μία, ενώ, αν μετατραπούν σε αριθμητικές τιμές με το τελεστή «+» γίνεται, πλέον, πρόσθεση των δύο τιμών επιστρέφοντας το άθροισμα τους.

Όλα τα παραπάνω αναδεικνύονται, αν δώσουμε στους μαθητές να εκτελέσουν τις παρακάτω εντολές στο διερμηνευτή και τους ζητήσουμε να εξηγήσουν τα αποτελέσματα:

```
>>> sxoleio =  "Ζάννειο Πειραματικό Γυμνάσιο"
>>> tmima =  "Γ4"
>>> print sxoleio[0] + sxoleio[1] + sxoleio[8]  \
          + sxoleio[7] + sxoleio[8] + sxoleio[4] \
          + sxoleio[5] + sxoleio[6]
>>> len( tmima )
>>> len( sxoleio )
>>> print  "Τμήμα " + tmima + " στο "  + sxoleio
>>> 12 + 48
>>> "12" + "48"
>>> str( 12 ) + str( 48 )
>>> int("12") + int("48")
```

Επίσης, πολύ σημαντικός είναι και ο τελεστής in για τον έλεγχο ύπαρξης ενός συμβόλου σε μια συμβολοσειρά. Μπορεί να χρησιμοποιηθεί το παράδειγμα 2 στη σελίδα 131 του ΒΜ με την καταμέτρηση των φωνηέντων μιας λέξης.

Στο παράδειγμα αυτό οι μαθητές έρχονται σε επαφή με την επαναληπτική διαδικασία **for** letter **in** word, με την οποία μπορούμε να διατρέξουμε όλα τα στοιχεία μιας συμβολοσειράς.

Με παρόμοια διερευνητική προσέγγιση και αξιοποιώντας τις δυνατότητες του διερμηνευτή, μπορούμε να συζητήσουμε με τους μαθητές και τις έννοιες των λιστών.

Λίστες

Οι λίστες έχουν αρκετές παρόμοιες λειτουργίες με τις συμβολοσειρές, όπως είναι η συνένωση (+), η συνάρτηση len, η προσπέλαση σε στοιχείο, ο έλεγχος ύπαρξης με τον τελεστή **in** και η διάσχιση με το ιδίωμα **for** item **in** List. Γι' αυτό προτείνεται να ξεκινήσουμε με ασκήσεις που πραγματεύονται αυτές τις έννοιες με τις οποίες είναι ήδη εξοικειωμένοι οι μαθητές.

Θα πρέπει όμως, να τονίσουμε τη βασική εννοιολογική διαφορά της λίστας από αυτή της συμβολοσειράς. Ενώ και οι δύο είναι ακολουθίες (sequences) δεδομένων, η λίστα, σε αντίθεση με την συμβολοσειρά, μπορεί και να αυξομειώσει το μέγεθός της και να τροποποιήσει τα στοιχεία της. Επίσης, η συμβολοσειρά περιέχει μόνο χαρακτήρες, ενώ η λίστα μπορεί να περιέχει διάφορους τύπους δεδομένων.

Οι λειτουργίες που πρέπει να γνωρίζουν οι μαθητές και τις οποίες μπορούν να χρησιμοποιήσουν στις ασκήσεις, είναι οι παρακάτω:

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.

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 είναι *αντικείμενα* και ως αντικείμενα διαθέτουν διάφορες *μεθόδους*. Από αυτές, στο μάθημα αυτό, θα χρησιμοποιούνται μόνο οι **append**, **insert** και **pop**, όπως περιγράφονται στο βιβλίο μαθητή. Ο εκπαιδευτικός, αν θεωρεί ότι η χρήση άλλων μεθόδων εκτός βιβλίου βοηθάει στην επίτευξη των διδακτικών στόχων, μπορεί να τις χρησιμοποιήσει, χωρίς αυτό να αποτελεί εξεταστέα ύλη. Ωστόσο, οι τρεις που αναφέρθηκαν είναι αρκετές για την επίτευξη των διδακτικών στόχων του μαθήματος.

Για παράδειγμα, αν ζητήσουμε να βρεθεί ένα στοιχείο σε μια λίστα και στη συνέχεια να διαγραφεί, τότε οι μαθητές θα πρέπει πρώτα να εκτελέσουν αναζήτηση στη λίστα και στη συνέχεια διαγραφή με την pop. Μπορούν όμως, με τη χρήση της remove, να κάνουν και τα δύο με μία εντολή. Επίσης, μπορούν, με μία μόνο εντολή, να βρουν τη μέγιστη τιμή μιας λίστας ή ακόμα και το πλήθος των στοιχείων μιας λίστας με τη μέγιστη τιμή:

```
>>> grades = [15, 15, 16, 17, 18, 9, 12, 18, 17,18]
>>> print max(grades)
>>> print grades.count(max(grades))
```

κάτι που αλλοιώνει εντελώς, τους διδακτικούς στόχους που θέλαμε να πετύχουμε με αυτήν την άσκηση. Πρέπει, λοιπόν να οριοθετούμε προσεκτικά ποιες είναι οι επιτρεπτές λειτουργίες / συναρτήσεις / μέθοδοι της γλώσσας που μπορούν να χρησιμοποιήσουν οι μαθητές, ώστε να έχουν νόημα οι ασκήσεις που τους θέτουμε στο πλαίσιο των στόχων του μαθήματος.

Στοίβα – Ουρά

Η στοίβα και η ουρά αποτελούν θεμελιώδεις δομές δεδομένων. Αν και οι εφαρμογές τους είναι απεριόριστες, στο πλαίσιο αυτού του μαθήματος, θα περιοριστούμε σε απλές εφαρμογές. Ο σκοπός μας είναι να κατανοήσουν οι μαθητές τις βασικές λειτουργίες των δομών αυτών και να μπορούν να τις χρησιμοποιήσουν στην επίλυση απλών προβλημάτων, όπως αυτά που δίνονται στο τετράδιο του μαθητή.

Σημαντική παρανόηση στην πρόσθεση στοιχείων

Εδώ, υπάρχει κάτι το οποίο θα πρέπει να προσέξουμε ιδιαιτέρως: η παρακάτω λανθασμένη υλοποίηση της push της στοίβας (και αντίστοιχα της enqueue της ουράς):

```
# Λάθος στην Ώθηση
def push(stack, item):
    stack = stack + [ item ]
```

Η παράμετρος stack δεν πρόκειται να αλλάξει, διότι **η εντολή απόδοσης τιμής δεν προσθέτει στη λίστα stack το στοιχείο item, αλλά δημιουργεί μια νέα λίστα**. Η παλιά λίστα μένει ανεπηρέαστη, αφού η αλλαγή δεν περνάει στο πρόγραμμα, αλλά μένει τοπικά.

Αναγκαστικά, πρέπει να χρησιμοποιήσουμε τη μέθοδο append, όπως φαίνεται παρακάτω.

```
# Ώθηση
def push(stack, item):
    stack.append(item)
```

Τώρα, η stack έξω από την push θα τροποποιηθεί.

8.3 Ενδεικτικές απαντήσεις Δραστηριοτήτων ΒΜ

Δραστηριότητα 1

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

Ενδεικτική Λύση

Διασχίζουμε τη λέξη, γράμμα – γράμμα, με την εντολή for … in …

```
# Εμφανίζει κάθε γράμμα της λέξης σε ξεχωριστή γραμμή
def splitLetters( ):
    word = raw_input("Δώσε μια λέξη : ")
    for letter in word:
        print letter
```

Δραστηριότητα 2

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

Ενδεικτική Λύση

Ελέγχει, αν ένα γράμμα είναι κεφαλαίο με χρήση του τελεστή in, διασχίζοντας τη λέξη γράμμα – γράμμα, με την εντολή for … in … , αφού πρώτα κατασκευάσει ένα string/σύνολο με όλα τα κεφαλαία αγγλικά γράμματα.

```
def countCapitals( ):
    enCapSet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    word = raw_input("Δώσε μια λέξη: ")
    countCapitals = 0
    for letter in word:
        if letter in enCapSet  :
            countCapitals += 1
    return countCapitals
print countCapitals( )
```

Δραστηριότητα 3

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

```
# αρχικά δίνουμε την υλοποίηση μιας στοίβας
# που θα χρησιμοποιήσουμε και σε επόμενες δραστηριότητες
# Ώθηση
def push(stack, item):
    stack.append(item)

# Απώθηση
def pop(stack):
    return stack.pop()

# Έλεγχος άδειας στοίβας
def isEmpty(stack):
    return len(stack)==0
```

```
# Δημιουργία νέας στοίβας
def createStack():
    return [ ]

# Αντιστρέφει μια λέξη με τη χρήση στοίβας. Το τελευταίο γράμμα της λέξης
#  είναι το τελευταίο που θα  εισέλθει στη στοίβα και θα βρίσκεται στην
# κορυφή της. Όταν ξεκινήσουμε να απωθούμε γράμματα από τη στοίβα θα
# βγει πρώτα το τελευταίο, μετά το προτελευταίο κ.ο.κ. Άρα, τα γράμματα θα
# εμφανιστούν σε αντίστροφη σειρά από αυτή με την οποία μπήκαν στην
# στοίβα.

def reverseWord( ):
    word = raw_input("Δώσε μια λέξη : ")
    reverse = ""
    stack = createStack( )
    for letter in word:
        push( stack, letter )

    while not isEmpty(stack):
        reverse = reverse + pop(stack)
        print reverse
```

Δραστηριότητα 4

Να γράψετε μια συνάρτηση is*Substring(string, substring)* η οποία θα ελέγχει, αν η συμβολοσειρά substring περιέχεται στην συμβολοσειρά string και αν ναι, θα επιστρέφει True. Στη συνέχεια, να γράψετε συνάρτηση η οποία να υπολογίζει πόσες φορές εμφανίζεται μια συμβολοσειρά μέσα σε μια άλλη.

```
# Αυτό μπορεί να γίνει με χρήση του τελεστή in πολύ απλά
def isSubstring( string, substring):
    return substring in string

# Ελέγχει αν το substring εμφανίζεται στο string ξεκινώντας από  τη θέση start
def isPrefix(string, substring, start):
    i = start
    j = 0
    count = 0
```

```
    while j < len(substring) and i < len(string) :
        if string[ i ] == substring[ j ] :
            count = count + 1
        i = i + 1
        j = j + 1
    return  count == len(substring)

# εκτελούμε για κάθε θέση του string την προηγουμένη συνάρτηση
# και έτσι υπολογίζουμε πόσες φορές εμφανίζεται το substring στο String
def findSubstring( string, substring):
    pos = 0
    count = 0

    while pos < len(string) :
        if isPrefix(string, substring, pos)== True :
            count = count + 1
        pos = pos + 1
        return count
```

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 απόλυτή τιμή του αριθμού.

```
# γίνεται χρήση της στοίβας που έχει οριστεί προηγουμένως
def push(stack, item):
    stack.append(item)
def pop(stack):
    return stack.pop()
def isEmpty(stack):
    return len(stack)==0
def createStack():
    return [ ]
```

```
def program85( ):
    stack = createStack()
    number = input("δώσε έναν αριθμό : ")
    while number != 0 :
        if number>0 :
            push(stack, number)
        else:
            i = 0
            while i <= number and not isEmpty(stack) :
                print stack.pop() ,
                i = i + 1
            print
        number = input("δώσε έναν αριθμό : ")
program85()
```

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, αν επιθυμείτε, ένα λεξικό.

```
# 1ος  τρόπος: χωρίς λεξικό
# Επιστρέφει τη θέση ενός γράμματος letter στο
# αλφάβητο alphabet
def indexStr(letter):
    capSet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    lowSet   = "abcdefghijklmnopqrstuvwxyz"
    for index in range(26):
        if letter == capSet[index] or letter == lowSet[index]:
            return index
    return -1;

def program86( ):
    enCapitalSet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    text = raw_input("Δώσε ένα κείμενο με αγγλικά γράμματα : ")
```

```
# δημιουργία και μηδενισμός μιας λίστας μετρητών
counter = [ ]
for i in range(26):
    counter.append(0)
# αυξάνουμε το μετρητή στην αντίστοιχη θέση του γράμματος
for letter in text:
    index = indexStr(letter)
    if index >= 0 :
        counter[ index ] += 1
print " Letter    Frequency"
print "--------  -----------"
for i in range(26):
    print "  ", enCapitalSet[i],"         ", counter[i]
```

```
# 2ος τρόπος: με λεξικό (μόνο για τον εκπαιδευτικό)
def program86_2( ):
    messg = ("Δώσε ένα κείμενο με αγγλικά μικρά γράμματα: ")
    text = raw_input( messg )
    alphabet = dict( )
    for letter in text:
        if letter in alphabet:
            alphabet[ letter ] += 1
        else:
            alphabet[ letter ] = 1
    print " Letter    Frequency"
    print "--------  -----------"
    for letter in alphabet:
        print "  ", letter,"         ", alphabet[letter]
    print "*********************************************************"
```

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.

```
def maxLength( wordList ) :
    maxLen = 0
    maxWord = ""
    for word in wordList:
        if len(word) > maxLen :
            maxLen = len(word)
            maxWord = word
    return maxWord
```

Δραστηριότητα 8

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

```
def num_of_Vowels( word ):
    vowels = "AEIOUYaeiouy"
    count = 0
    for letter in word:
        if letter in vowels:
            count += 1
    return count
def maxVowels( wordList ):
    maxV = 0
    maxWord = ""
    for word in wordList:
        if num_of_Vowels(word) > maxV :
            maxV = num_of_Vowels(word)
            maxWord = word
    return maxWord
```

11. Αντικειμενοστρεφής Προγραμματισμός

11.1 Οδηγίες

Το κεφάλαιο 11 αποτελείται από πέντε ενότητες, εκ των οποίων μόνο οι τρεις πρώτες είναι, όπως αναφέρθηκε στην αρχή, στη διδακτέα και στην εξεταστέα ύλη. Αυτό σημαίνει ότι μόνο οι τρεις από τους πέντε διδακτικούς στόχους είναι συναφείς. Συγκεκριμένα να μπορούν οι μαθητές να:

- περιγράφουν τις διαφορές του αντικειμενοστρεφούς προγραμματισμού σε σχέση με το δομημένο προγραμματισμό
- δημιουργούν απλά αντικείμενα και κλάσεις
- διακρίνουν την έννοια της κλάσης από εκείνη του αντικειμένου.

Οι στόχοι αυτοί δεν έχουν χαρακτήρα στείρας αποστήθισης με τη μορφή ορισμών, που άλλωστε δεν υπάρχουν ούτε και σε αυτό το κεφάλαιο. Αφορούν αφενός στην εξοικείωση με τα βασικά δομικά στοιχεία του αντικειμενοστρεφούς προγραμματισμού και αφετέρου στη λειτουργική τους χρήση στο πλαίσιο του συγκεκριμένου *προγραμματιστικού παραδείγματος*.

Η έμφαση, επομένως, είναι στο *αντικείμενο* ως βασική εννοιολογική και ταυτόχρονα προγραμματιστική μονάδα. Αναπόφευκτα, έμφαση δίνεται επίσης, σε όσα καθιστούν το αντικείμενο εύκολο στη δημιουργία και με τη δυνατότητα να έχει πολλές διαφορετικές παρουσίες μέσα στο πρόγραμμά μας (*κλάσεις*) καθώς και σε όσα του επιτρέπουν να είναι ενεργό, να έχει μεταβλητές ιδιότητες και να ανταποκρίνεται σε εξωτερικές επιδράσεις (*μέθοδοι*). Έτσι, έμφαση δίνεται στην κλάση, ως τρόπο δημιουργίας ή/και απαραίτητη γενίκευση των αντικειμένων και στη μέθοδο, ως τρόπο διαχείρισης των ιδιοτήτων και επομένως, έκφραση των ενεργειών ή, αλλιώς, των συμπεριφορών του αντικειμένου.

Η διάκριση της έννοιας της κλάσης από αυτήν του αντικειμένου είναι βασική σε πολλά επίπεδα, αλλά δεν μπορεί να προκύψει με ευκρίνεια, παρά μόνο μετά από εμπειρική προσέγγιση. Η δημιουργία αντικειμένων ως στιγμιότυπων μιας κλάσης (στην ενότητα 11.2) στην πράξη διευκολύνει πολύ την ανάδειξη των διαφορών αλλά και τη διαμόρφωση από το μαθητή, μιας λειτουργικής πρώτης θεωρίας για

τη σχέση των εννοιών. Η προσέγγιση της κλάσης ως μιας συνταγής δημιουργίας αντικειμένων, ήδη από την ενότητα 11.1 και ταυτόχρονα ως ενός τύπου δεδομένων ορισμένων από το χρήστη (δες και τα παραδείγματα εδώ) είναι επίσης βοηθητική.

Η όλη εμπειρία άλλωστε του μαθητή από την Python υποστηρίζει αυτήν την προσέγγιση. Η γλώσσα προγραμματισμού Python, τόσο στην έκδοση που χρησιμοποιούμε όσο και σε επόμενες, όλο και περισσότερο αντιμετωπίζει τύπους δεδομένων και κλάσεις ως έννοιες περίπου ταυτόσημες. Το γεγονός ότι το κεφάλαιο για τον αντικειμενοστρεφή προγραμματισμό βρίσκεται σε τόσο προχωρημένη θέση στην ύλη, επιτρέπει την αξιοποίηση αυτής της εμπειρίας.

Αντίστοιχα, χρήσιμη προσέγγιση είναι, αυτή της μεθόδου ως περιγραφής συμπεριφοράς είτε της κλάσης είτε του αντικειμένου. Παραδείγματα ορισμού πράξεων, όπως στο παράδειγμα 1 εδώ, λειτουργούν θετικά σε αυτήν την κατεύθυνση. Παράλληλα, η προσέγγιση της συμπεριφοράς, μαζί με το γεγονός ότι οι μέθοδοι ανήκουν στις κλάσεις και τελικά στα αντικείμενα, απαντά πειστικά και με απλό τρόπο στη συχνή απορία των μαθητών: "ποια είναι η διαφορά μιας μεθόδου από μια συνάρτηση και γιατί έχουν διαφορετικό όνομα, αν πρόκειται απλά για συναρτήσεις;". Η συζήτηση για την εμβέλεια των μεν και των δε είναι απαραίτητη, αλλά μπορεί να αναβληθεί, μέχρι να αποκτήσουν οι μαθητές μια εργαλειακή ευχέρεια στην υλοποίηση απλών προγραμμάτων.

Όπως είναι φανερό, η αφαίρεση από την ύλη της ενότητας 11.4 αφαιρεί σημαντικές έννοιες του αντικειμενοστρεφούς προγραμματισμού, όπως η *κληρονομικότητα*, ο *πολυμορφισμός* και η *ενθυλάκωση*. Τυχόν υπάρχουσες δραστηριότητες οι οποίες περιέχουν στοιχεία αυτών των εννοιών έχουν συμπεριληφθεί μόνο για ενημέρωση των εκπαιδευτικών.

Το κεφάλαιο αυτό, όπως και ο ίδιος ο αντικειμενοστρεφής προγραμματισμός, αντιμετωπίζεται συχνά από ορισμένους διδάσκοντες ως δύσκολο θέμα. Όμως, η προσέγγιση του συγκεκριμένου βιβλίου πιστεύουμε ότι αποφεύγει συχνές παγίδες που συντείνουν στις δυσκολίες. Για παράδειγμα, αποφεύγεται τόσο η έμφαση στην *οντολογία,* όσο και η πρόωρη εστίαση στα γραφικά περιβάλλοντα, που,

παρά το γεγονός ότι αποτελούν τις πιο διαδεδομένες εφαρμογές, δεν είναι ταυτόχρονα από τις πιο εύκολες.

Θεωρούμε λοιπόν ότι το συγκεκριμένο κεφάλαιο, ακόμα και χωρίς τις περικοπές στη διδακτέα και εξεταστέα ύλη, μπορεί να αποδειχτεί από τα «εύκολα» και πάντως ευχάριστα κεφάλαια του βιβλίου. Επιπλέον, το γεγονός ότι η Python είναι μια γλώσσα που στη βάση της είναι ιδιαίτερα αντικειμενοστρεφής, σε συνδυασμό με την τουλάχιστον διετή πλέον εμπειρία των μαθητών μας σε αυτή, θα βοηθήσει σε αυτήν την κατεύθυνση.

Παραδείγματα για τον εκπαιδευτικό

Στο Διαδικτυακό τόπο (cs.hmc.edu, 2016), υπάρχουν προηγμένα παραδείγματα με χρήση αντικειμενοστρεφούς προγραμματισμού σε Python. Τα παραδείγματα αυτά, λόγω της δυσκολίας τους, προσφέρονται μόνο για την ενημέρωση του διδάσκοντα και όχι για διδασκαλία τους μαθητές.

Ακολουθεί ένα παράδειγμα δημιουργίας νέων τύπων δεδομένων.

Δημιουργία νέων τύπων δεδομένων

Θέλουμε να φτιάξουμε ένα πρόγραμμα που να συγκρίνει και να κάνει πράξεις με κλάσματα. Πρέπει λοιπόν για κάθε κλάσμα να κρατάμε τον αριθμητή και τον παρονομαστή του. Η Python μας προσφέρει ακέραιους και πραγματικούς αριθμητικούς τύπους, αλλά δεν προσφέρει ένα ξεχωριστό τύπο που να αναφέρεται σε κλάσματα. Στην πραγματικότητα, αν πληκτρολογήσουμε στο περιβάλλον της Python

```
>>> 3/4
```

επιστρέφει 0, οπότε, σε αυτήν την περίπτωση, είναι ακεραίου τύπου, ενώ αν πληκτρολογήσουμε

```
>>> 3.0/4
```

επιστρέφει 0.75, οπότε, σε αυτήν την περίπτωση, είναι πραγματικού τύπου.

Για το πρόγραμμα που θέλουμε να υλοποιήσουμε εμείς όμως, θα ήταν χρήσιμο, να μας έδινε η Python έναν τρόπο να χειριστούμε τα κλάσματα σαν ένα ζευγάρι ακεραίων. Δηλαδή, θα θέλαμε να έχουμε ένα νέο τύπο δεδομένων με το όνομα «κλάσμα», όπως ακριβώς, η Python έχει τους αριθμητικούς τύπους δεδομένων: ακέραιος και

πραγματικός. Επιπλέον, θα ήταν βολικό, αν μπορούσαμε να κάνουμε αριθμητικές πράξεις και συγκρίσεις κλασμάτων, όπως ακριβώς κάνουμε με τους ακεραίους.

Αυτή η δυνατότητα, να μπορούμε να δηλώσουμε νέους τύπους δεδομένων και να ορίσουμε τη συμπεριφορά τους, είναι η βάση αυτού που ονομάζεται αντικειμενοστρεφής προγραμματισμός και στην Python υλοποιείται με τη δημιουργία κλάσεων. Μέσα σε αυτές τις κλάσεις, οι μέθοδοι προσδιορίζουν τη συμπεριφορά.

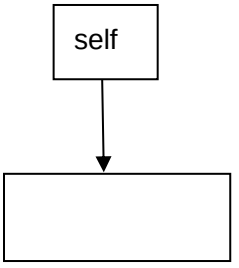
Έτσι, για το παράδειγμά μας ορίζουμε την κλάση *κλάσμα* ως εξής:

```
>>> class  Klasma:
        def __init__(self, arith, paron):
                self.arithmitis = arith
                self.paronomastis = paron
```

Από τη στιγμή που ορίσουμε την παραπάνω κλάση, μπορούμε να δημιουργήσουμε νέα κλάσματα, δηλαδή στιγμιότυπα της κλάσης Klasma. Για παράδειγμα παρακάτω δημιουργούμε δύο κλάσματα, το 5/100 και το 3/100:

```
>>> k1 = Klasma (5, 100)
>>> k2 = Klasma (3, 100)
```

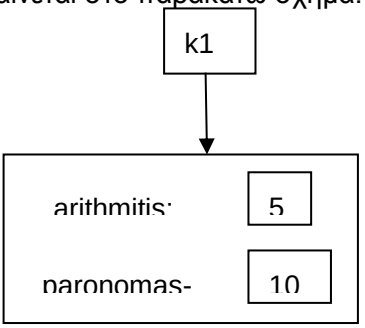
Όταν η Python αναγνωρίσει την εντολή k1 = Klasma (5, 100), κάνει δύο πράγματα. Αρχικά, δημιουργεί ένα κενό αντικείμενο που ονομάζουμε self. Στην πραγματικότητα το self είναι ένας δείκτης σε ένα κενό αντικείμενο, όπως φαίνεται στο παρακάτω σχήμα:



Στη συνέχεια, η Python ψάχνει μέσα στην κλάση Klasma να βρει μια συνάρτηση με το όνομα __init__. Η συνάρτηση αυτή χρησιμοποιεί το δείκτη self για να δώσει ως αρχικές τιμές στις ιδιότητες του

αντικειμένου arithmitis και paronomastis, τις τιμές 5 και 100 αντίστοιχα. Η συνάρτηση __init__ ονομάζεται μέθοδος και συγκεκριμένα **μέθοδος αρχικοποίησης τιμών** ή αλλιώς **κατασκευαστής**.

Το τελευταίο πράγμα που συμβαίνει, είναι ότι η μεταβλητή k1 είναι πλέον ένας δείκτης στο αντικείμενο που η Python μόλις δημιούργησε, όπως φαίνεται στο παρακάτω σχήμα:



Ορίσαμε λοιπόν, ένα νέο τύπο δεδομένων με όνομα “Klasma” και μπορούμε να δημιουργούμε αντικείμενα αυτού του τύπου.

Στόχος μας όμως, είναι να μπορούμε να κάνουμε πράξεις και συγκρίσεις κλασμάτων, όπως ακριβώς κάνουμε με τους ακεραίους. Αυτό γίνεται προσθέτοντας νέες συναρτήσεις μέσα στην κλάση. Θα προσθέσουμε λοιπόν μια καινούρια συνάρτηση στην κλάση “Klasma” που θα μας επιτρέπει να προσθέτουμε ένα κλάσμα με ένα άλλο και να επιστρέφουμε ένα νέο κλάσμα που θα είναι το άθροισμα των προηγούμενων. Αυτή η συνάρτηση, που θα την ονομάσουμε “add”, είναι μια **μέθοδος** της κλάσης. Έτσι, η κλάση μας γίνεται ως εξής:

```
>>> class  Klasma:
        def __init__(self, arith, paron):
            self.arithmitis = arith
            self.paronomastis = paron
        def add(self, other):
            neos_arithmitis = self.arithmitis * other.paronomastis
+ self.paronomastis * other.arithmitis
            neos_paronomastis = self.paronomastis * other.paro-
nomastis
            return Klasma(neos_arithmitis, neos_paronomastis)
```

Ας δώσουμε λοιπόν τώρα, τις παρακάτω εντολές στο περιβάλλον της Python:

```
>>> k1 = Klasma(1,2)
>>> k2 = Klasma (2, 3)
>>> k3 = k1.add(k2)
>>> k3.arithmitis
7
>>> k3.paronomastis
6
>>>
```

Βλέπουμε ότι δημιουργούνται τρία κλάσματα, με το κλάσμα k3 να είναι αυτό που προκύπτει από την πρόσθεση των κλασμάτων k1 και k2.

Με τον ίδιο τρόπο μπορούμε να προσθέσουμε και άλλες μεθόδους στην κλάση μας.

11.2 Λύσεις Δραστηριοτήτων ΒΜ

Δραστηριότητα εμπέδωσης (παραγράφου 11.2)

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 και 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 αντίστοιχα του αυτοκινήτου και να φροντίσετε ώστε 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.

Ενδεικτικές Απαντήσεις

1. Ο κατασκευαστής (constructor) της κλάσης είναι η μέθοδος __init____(self, make), ο οποίος δημιουργεί το αντικείμενο και αρχικοποιεί τις ιδιότητές του.

2. Ιδιότητες της κλάσης **Car:** make, speed
   Μέθοδοι: 1) speed_up(self,speed): δίνει τιμή στην ιδιότητα speed του αντικειμένου και εμφανίζει στην οθόνη το μήνυμα «I am driving at a speed» και στη συνέχεια εμφανίζει την τιμή της ιδιότητας του αντικειμένου ακολουθούμενη από το μήνυμα «km/h»
   2) turn(self): εμφανίζει στην οθόνη το μήνυμα «I am turning …»

3. Ο κατασκευαστής της κλάσης αλλάζει ως εξής:

```
def __init__(self, make, color,year):
        self.make=make
        self.speed = 60
        self.color = color
        self.year  = year
```

```
4. def turn(self, direction):
        print "I am turning … ", direction

5. I. convertible = Car("bmw","black",2013)
   II. sedan = Car("Toyota","red",2009)
6. convertible.turn("right")
7. sedan.speed_up(90)
```

Δραστηριότητα εμπέδωσης παραγράφου 11.4

- Ποιες είναι οι ιδιότητες της κλάσης Student;
- Δημιουργήστε στιγμιότυπο της κλάσης Student με όνομα “Γιάννης” και τάξη “Β”.
- 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”.

Ενδεικτική λύση

- Ιδιότητες της κλάσης Student: _name (που την κληρονομεί από την κλάση Person) και classatt
- student1 = Student(“Γιάννης”,”Β”)
- print student1._name

  print student1.classatt
- ```
  class Teacher (Person):
          def __init__(self, name,field,years):
                  super (Teacher, self).__init__(name)
  ```
  ```

```
                    self.field = field
                    self.years = years
```

- teacher1= Teacher("Αναστασίου ","ΠΕ02",11)
  teacher2 = Teacher("Παπαχρήστου ", "ΠΕ03", 16)

Δραστηριότητες (Παραγράφου 11.5)

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. καθώς και να την εμφανίσετε.

Ενδεικτική λύση

```
class Akeraios :
        def __init__(self):
                self.timi = 0
        def  Anethese_timi(self, timi):
                self.timi = timi
        def  Emfanise_timi(self):
                print self.timi


Artios = Akeraios()

Artios. Anethese_timi(14)
Artios. Emfanise_timi()
```

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:

- Μια μέθοδο για την ανάθεση τιμών στις ιδιότητες ενός αντικειμένου.
- Μια μέθοδο για την εμφάνιση των τιμών των ιδιοτήτων ενός αντικειμένου.
- 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.

Ενδεικτική λύση

```
class Student:
        def __init__(self, AM, name,grades):
            self.AM=AM
            self.name = name
            self.grades = grades[:8]
        def Emfanise_times(self):
            print "AM: ", self.AM
            print "Name: ", self.name
            print "grades: ", self.grades

        def Ypol_MO(self):
            sum=0
            for i in range(len(self.grades)):
                sum = sum + self.grades[i]
            return sum/len(self.grades)
    chris = Student( 209,“Chris”,[14,16,18,19,12,20,17,20])
    chris. Emfanise_times()
    print chris.Ypol_MO()
```

12. Βασικές Αναφορές

(Η τελευταία προσπέλαση στις ηλεκτρονικές πηγές έγινε τον Απρίλιο του 2017)

- cs.hmc.edu, 2016, Παιχνίδια και Αντικειμενοστρεφής Προγραμματισμός. Διαθέσιμο στο: https://www.cs.hmc.edu/csforall/OOPs/oops.html,
- idle-python2.6 (2016), Εγκατάσταση του IDLE της Python σε περιβάλλον Linux. Διαθέσιμο στο: https://www.cyberciti.biz/faq/rhel-centos-debian-ubuntu-python-idle-linux-installation/)
- visualgo (2016). Οπτικοποίηση λειτουργίας αλγορίθμων ταξινόμησης. Διαθέσιμο στο: https://visualgo.net/sortin
- 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
- 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.