Πληροφορική Γυμνασίου

Αρχές Προγραμματισμού με τη γλώσσα Python

Ευριπίδης Βραχνός

http://evripides.mysch.gr/

2016 – 2017

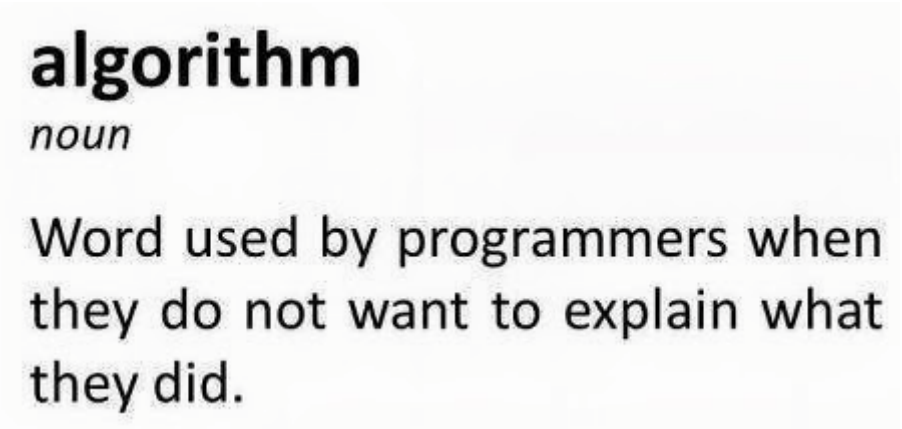
ZEN OF PYTHON

Keep in mind Python's philosophy as you code.

```
>>> import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
...
```



1

Εισαγωγή στον Προγραμματισμό

Εισαγωγή

Στο κεφάλαιο αυτό θα έχουμε την ευκαιρία να γνωρίσουμε και να εξοικειωθούμε με τα βασικά χαρακτηριστικά της γλώσσας προγραμματισμού Python. Μιας σύγχρονης γλώσσας προγραμματισμού, που χρησιμοποιείται για την ανάπτυξη πολλών προγραμμάτων, όπως απλά προγράμματα, εκπαιδευτικά παιχνίδια, σύνθετες εμπορικές εφαρμογές, πλατφόρμες κοινωνικής δικτύωσης (youtube, dropbox, instagram, google). Είναι μια διερμηνευόμενη, υψηλού επιπέδου γλώσσα, απλή και εύκολη στην εκμάθησή της. Άλλες διαδεδομένες σύγχρονες γλώσσες υψηλού επιπέδου είναι η C++, η Java, η Perl, η Ruby, η Processing.

Οι γλώσσες υψηλού επιπέδου βασίζονται γενικά σε λέξεις κλειδιά, συνήθως της Αγγλικής γλώσσας και ακολουθούν μια τυπική αυστηρή γραμματική και συντακτικό που καθορίζονται κατά τη φάση σχεδιασμού της γλώσσας. Έχουν το πλεονέκτημα ότι είναι εύκολα κατανοητές από τον άνθρωπο-προγραμματιστή και χαρακτηρίζονται από μεγάλη *φορητότητα*, δηλαδή τα προγράμματα που φτιάχνονται σε αυτές τις γλώσσες μπορούν να εκτελεστούν σε διάφορα είδη υπολογιστών χωρίς καθόλου ή με μικρές τροποποιήσεις.

Το ολοκληρωμένο προγραμματιστικό περιβάλλον της γλώσσας Python, που θα χρησιμοποιείται στο μάθημα αυτό, διατίθεται μέσω Διαδικτύου και μπορούμε να το εγκαταστήσουμε στον υπολογιστή μας, καθώς αποτελεί Ελεύθερο Λογισμικό Ανοικτού Κώδικα. Η γλώσσα προγραμματισμού Python είναι πολύ διαδεδομένη, με μια μεγάλη κοινότητα προγραμματιστών να την υποστηρίζει, δίνοντας συμβουλές και υλικό ελεύθερα στο Διαδίκτυο. Επιπρόσθετα περιέχει πλούσιες βιβλιοθήκες από έτοιμο κώδικα προγραμματισμού, που μπορούμε να τον χρησιμοποιούμε στα προγράμματά μας, μεθοδολογία που μας επιτρέπει την εύκολη και γρήγορη συγγραφή σύνθετων προγραμμάτων.

Για να επικοινωνήσουμε με τον υπολογιστή με τη γλώσσα Python, δίνοντας τις κατάλληλες εντολές για να τις εκτελέσει, χρειαζόμαστε να μάθουμε ένα πολύ μικρό σύνολο λέξεων-ρημάτων σαν της αγγλικής γλώσσας, που αντιστοιχούν στις εντολές, καθώς και τον τρόπο σύνταξής τους. Το πλεονέκτημα είναι ότι μόλις γράψουμε μια εντολή μπορούμε να ελέγξουμε το αποτέλεσμα, καθώς το προγραμματιστικό περιβάλλον διερμηνεύει τις εντολές άμεσα και μας ειδοποιεί αν έχουμε κάνει κάποιο λάθος στη σύνταξή τους. Αυτή η αμεσότητα μας δίνει τη δυνατότητα να μπορούμε να μαθαίνουμε μόνοι μας, χωρίς να διστάζουμε να πειραματιστούμε.

Ξεκινήστε έχοντας λίγη υπομονή στην αρχή. Προσέξτε ιδιαίτερα στη σύνταξη των εντολών καθώς είναι ιδιαίτερα αυστηρή. Δώστε την ανάλογη προσοχή στη λεπτομέρεια, χρησιμοποιείστε τον υπολογιστή. Μόλις μπείτε στο συναρπαστικό κόσμο του προγραμματισμού και αρχίζετε να μαθαίνετε να προγραμματίζετε δείξτε και σε άλλους, συνεργαζόμενοι σε ομάδες. Όταν προσπαθείτε να επεξηγήσετε κάτι σε κάποιο συμμαθητή σας, θα βοηθήσει και σας να κατανοήσετε σε μεγαλύτερο βάθος κάποια σημεία που μπορεί να μην είχατε αρχικά προσέξει.

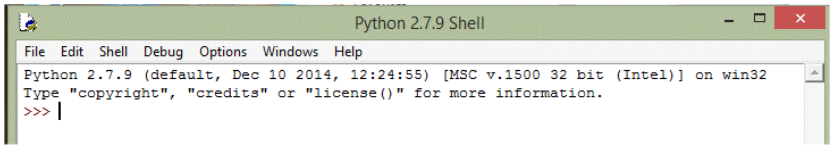
Γνωριμία με το ολοκληρωμένο περιβάλλον ανάπτυξης της γλώσσας προγραμματισμού

Πριν προχωρήσουμε να μαθαίνουμε πώς να προγραμματίζουμε στη γλώσσα προγραμματισμού Python, είναι χρήσιμο να εξοικειωθούμε με το πώς αυτή δουλεύει. Έτσι, θα ασχοληθούμε στη συνέχεια με το:

- πώς να εγκαθιστάμε το ολοκληρωμένο προγραμματιστικό περιβάλλον της Python
- πώς να το χρησιμοποιούμε για να γράψουμε το πρώτο μας απλό πρόγραμμα και στη συνέχεια να το αποθηκεύσουμε.

Για να ξεκινήσουμε να χρησιμοποιούμε τη γλώσσα προγραμματισμού Python χρειάζεται πρώτα να μεταφορτώσουμε (κατεβάσουμε) από το Διαδίκτυο και να εγκαταστήσουμε στον υπολογιστή μας το προγραμματιστικό περιβάλλον της. Για τις ανάγκες του μαθήματος θα χρησιμοποιήσουμε την έκδοση 2.7.13 της Python και ο χρωματισμός του κώδικα ακολουθεί το IDLE της Python.

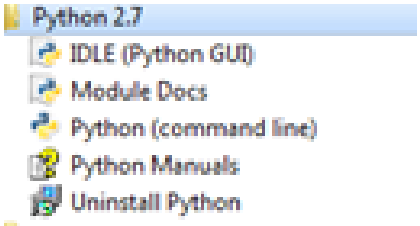
Το **Ολοκληρωμένο Περιβάλλον Ανάπτυξης Προγραμμάτων IDLE** (Integrated DeveLopment Environment) της Python, είναι ένα Ελεύθερο Λογισμικό/ Λογισμικό Ανοικτού Κώδικα (ΕΛ/ΛΑΚ), που, εύκολα, μπορούμε να το εγκαταστήσουμε στον υπολογιστή μας. Αρκεί να κατεβάσουμε από το Διαδίκτυο -και στη συνέχεια να εκτελέσουμε- το κατάλληλο αρχείο, ανάλογα με το λειτουργικό σύστημα του υπολογιστή μας, εφ' όσον είναι διαθέσιμο για διάφορα λειτουργικά συστήματα, όπως Microsoft Windows, Linux / Ubuntu, MAC OS X.



Εικόνα 3-1. Το περιβάλλον IDLE της Python

Εγκατάσταση IDLE Python για Microsoft Windows

Το προγραμματιστικό περιβάλλον IDLE είναι διαθέσιμο στον επίσημο δικτυακό τόπο υποστήριξης της γλώσσας Python https://www.python.org/. Αρχικά επιλέγουμε την έκδοση της γλώσσας (2.7.13), μεταφορτώνουμε το κατάλληλο αρχείο ανάλογα με την έκδοση των Windows και το εγκαθιστάμε εκτελώντας το στον υπολογιστή, αποδεχόμενοι τους προτεινόμενους όρους. Στη συνέχεια ελέγχουμε αν έγινε σωστά η εγκατάσταση. Τέλος ανοίγουμε το μενού επιλογών ως εξής: Έναρξη→Όλα τα προγράμματα→ Python→ και επιλέγουμε IDLE (εικόνα 3.1).



Εικόνα 3-1. *Επιλογή IDLE για να ξεκινήσει η εκτέλεση του προγραμματιστικού περιβάλλοντος*

Επιλέγοντας IDLE με το ποντίκι, ξεκινάει η εκτέλεση του προγραμματιστικού περιβάλλοντος και ανοίγει ένα παράθυρο στο μέσο της οθόνης. Τώρα, μπορούμε άμεσα να ξεκινήσουμε να πληκτρολογούμε το πρώτο μας πρόγραμμα, αρκεί να γράψουμε την κατάλληλη εντολή δίπλα στα τρία σύμβολα **>>>**.

Το Python Shell, είναι μέρος του ολοκληρωμένου περιβάλλοντος προγραμματισμού και ουσιαστικά επιτρέπει τη συγγραφή κάθε εντολής ξεχωριστά (η εντολή ακολουθεί μετά τα τρία σύμβολα **>>>**) και την άμεση εκτέλεσή της με τη βοήθεια του διερμηνευτή της γλώσσας.

Δραστηριότητα. Το πρώτο μας πρόγραμμα

Ας δοκιμάσουμε να δημιουργήσουμε το πρώτο μας πρόγραμμα με στόχο να εμφανιστεί στην οθόνη του υπολογιστή το μήνυμα χαιρετισμού: «Χαίρε, κόσμε!». Για το πρόγραμμά μας θα χρησιμοποιήσουμε την εντολή εμφάνισης **print**, μια εντολή εξόδου με τη δυνατότητα να εμφανιστεί στην έξοδο του υπολογιστή (οθόνη) ένα μήνυμα ή το αποτέλεσμα μιας πράξης.

```
>>> print "Ζάννειο Πειραματικό Γυμνάσιο"
Ζάννειο Πειραματικό Γυμνάσιο
```

Θα δουλέψουμε αρκετό χρόνο στον διερμηνευτή ώστε να εξοικειωθείτε με τις βασικές εντολές και στη συνέχεια θα χρησιμοποιήσουμε τον συντάκτη κώδικα του περιβάλλοντος IDLE για να δημιουργήσουμε και να αποθηκεύσουμε ολοκληρωμένα προγράμματα.

Η διαδικασία έχει ως εξής:

- από το μενού επιλογών File→ Άνοιγμα νέου αρχείου (**New File**)
- σύνταξη του κώδικα του προγράμματος
- αποθήκευση του προγράμματος (**Save**, όνομα αρχείου με κατάληξη .py)
- εκτέλεση του προγράμματος από την επιλογή Run→**Run Module** (ή πατάμε το πλήκτρο **F5**), εμφάνιση των αποτελεσμάτων στο Python Shell.

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 τύπων:

- ακέραιοι (Integer) (**int**)
- αριθμοί κινητής υποδιαστολής (floating point) (**float**)

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 * 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 ().

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

Αριθμητικές και λογικές 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 ο υπολογισμός της ακολουθεί 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 (Ή): πράξη διάζευξης.

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

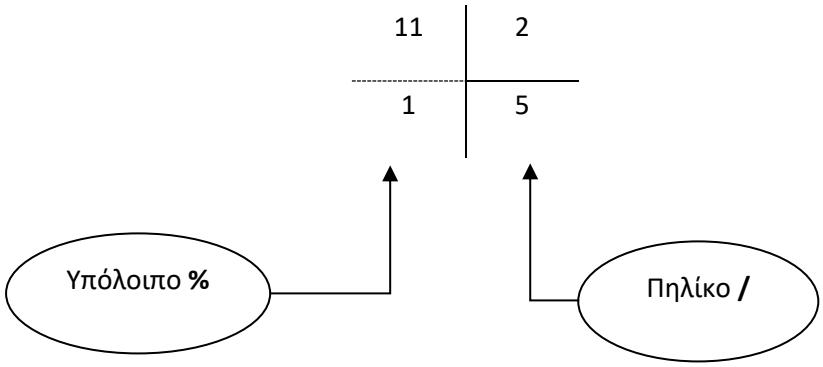
P	Q	P and Q	P or Q	Not P
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Η προτεραιότητα των λογικών τελεστών είναι not, and, or με αυτή τη σειρά.

Αριθμητικές πράξεις	**Αποτελέσματα**	**Επεξήγηση**
>>> 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

Θα πρέπει να προσέξετε τους τελεστές της διαίρεσης / και %. Το αποτέλεσμα του πρώτου τελεστή όπως φαίνεται παραπάνω εξαρτάται από τον τύπο των αριθμών που συμμετέχουν στην έκφραση. Αν ένας τουλάχιστον από τους αριθμούς δεν είναι ακέραιος τότε η διαίρεση είναι η γνωστή πραγματική διαίρεση που ξέρουμε. Όμως αν και οι δυο αριθμοί είναι ακέραιοι τότε το αποτέλεσμα είναι το πήλικό της ευκλείδειας διαίρεσης.

Παρακάτω δίνεται ένα παράδειγμα ώστε να γίνουν όλα πιο κατανοητά:



Όλα προκύπτουν από την ταυτότητα της διαίρεσης του Ευκλείδη

$$\Delta = \delta \cdot \pi + \upsilon$$

με $\pi = \Delta$ **div** δ και $\upsilon = \Delta$ **mod** δ

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

Μεταβλητές

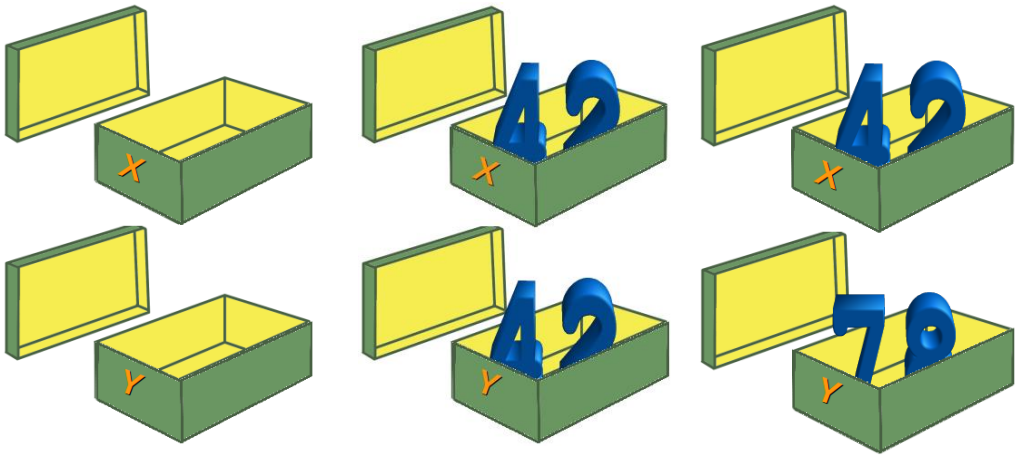
Αρκετές φορές σε ένα πρόγραμμα απαιτείται να αποθηκεύσουμε προσωρινά κάποια δεδομένα στη μνήμη του υπολογιστή. Για το σκοπό αυτό χρησιμοποιούμε τις **μεταβλητές**.

Οι μεταβλητές στον προγραμματισμό αντιστοιχούν σε μία θέση μνήμης του υπολογιστή. Κάθε φορά στη θέση αυτή μπορεί να αποθηκευτεί μόνο μία τρέχουσα τιμή. Οι μεταβλητές μπορούν να παίρνουν τιμές από διάφορους τύπους δεδομένων. Με τον όρο τιμή εννοούμε μια ακολουθία από bit (0,1) η οποία ερμηνεύεται σύμφωνα με κάποιον τύπο δεδομένων. Είναι δυνατό η ίδια ακολουθία από bits να έχει διαφορετική ερμηνεία ανάλογα με τον τύπο δεδομένων του οποίου ερμηνεύεται. Οι μεταβλητές χρησιμεύουν, ώστε εύκολα να μπορούμε να έχουμε πρόσβαση στο περιεχόμενό τους, που βρίσκεται προσωρινά αποθηκευμένο στη θέση μνήμης του υπολογιστή, που έχει δεσμευτεί για τη μεταβλητή αυτή.

Μπορείτε να φανταστείτε τις μεταβλητές ως κουτιά μέσα στα οποία μπορούμε να βάλουμε διάφορα αντικείμενα όπως αριθμούς ή λέξεις, αλλά κάθε κουτί μπορεί να δεχτεί

μόνο έναν αριθμό ή μια λέξη. Για παράδειγμα οι μεταβλητές x και y παρακάτω δεν έχουν ακόμα πάρει τιμή.

```
>>> x = y = 42
>>> y = 78
```



Ανάθεση τιμής σε μεταβλητές συμβατικών γλωσσών προγραμματισμού (όχι στην python ☺)

Στις περισσότερες γλώσσες προγραμματισμού οι δυο μεταβλητές αναφέρονται σε διαφορετικές θέσεις στη μνήμη όπως φαίνεται παραπάνω.

Η Python όμως διαφέρει πολύ στην διαχείριση των μεταβλητών. Η γλώσσα Python παρέχει εντυπωσιακές εναλλακτικές δυνατότητες έκφρασης 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, όπως ακέραιες, κινητής υποδιαστολής, 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: name, onoma, grade, area κ.ά.

Για να δώσουμε 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.

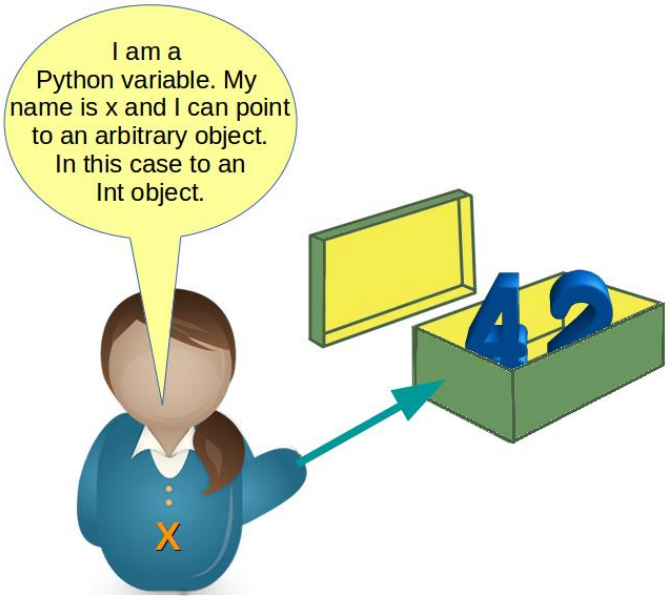
Προσοχή! ο τελεστής "=" δεν έχει τη σημασία που έχει το σύμβολο της ισότητας, όπως το χρησιμοποιούμε στα μαθηματικά. Στις περισσότερες γλώσσες προγραμματισμού, ο τελεστής εκχώρησης τιμής " = " σημαίνει ότι εκχωρούμε στη μεταβλητή μια τιμή ενός τύπου δεδομένου και ταυτόχρονα την εισάγουμε στη θέση μνήμης που αντιστοιχεί στη μεταβλητή με αυτό το όνομα. Για παράδειγμα, η εντολή x = 42 σημαίνει ότι η μεταβλητή με όνομα x αντιστοιχεί σε μια θέση μνήμης του υπολογιστή και η ακέραια τιμή 42 εισάγεται στη θέση

αυτή. Αν στη συνέχεια θέσουμε x = 250, στη θέση μνήμης που αναφέρεται η μεταβλητή a εισάγεται η νέα ακέραια τιμή 250 και η παλαιά τιμή 42 παύει πλέον να υπάρχει, αφού έχει αντικατασταθεί από τη 250.

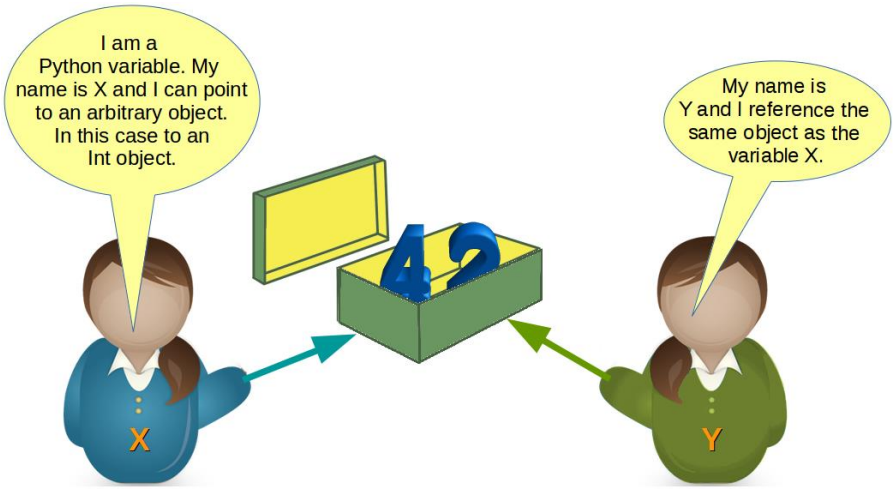
Όλα τα δεδομένα σε ένα πρόγραμμα Python αναπαρίστανται με αντικείμενα ή με σχέσεις μεταξύ των αντικειμένων, με κάθε αντικείμενο να έχει μια ταυτότητα (identity), έναν τύπο και μία τιμή. Για παράδειγμα, το 12 είναι ένα αντικείμενο με τιμή 12, τύπου int (ακέραιος).

a=1 a=2 a=b=2

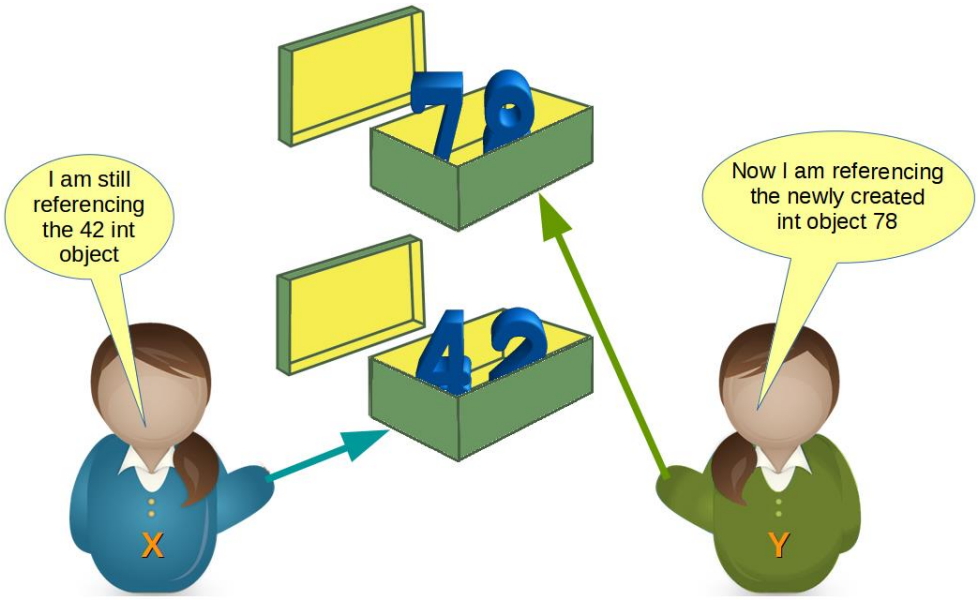
Οι μεταβλητές στην Python λειτουργούν σαν ετικέτες με κάποιο όνομα, που χρησιμεύουν για να αναφερόμαστε (ή αλλιώς για να δείχνουμε) σε κάποια αντικείμενα. Ο τύπος της μεταβλητής είναι ο εκάστοτε τύπος του αντικειμένου στην οποία αναφέρεται. Στο παράδειγμα **x = 42**, δημιουργείται η μεταβλητή με όνομα x και αναφέρεται στο αντικείμενο, με τιμή 42, με ακέραιο τύπο δεδομένων. Το σύμβολο "=" δημιουργεί ένα είδος δεσίματος μεταξύ του αντικειμένου 42 και της μεταβλητής με όνομα x. Ο τύπος της μεταβλητής είναι και αυτός ακέραιος, αφού αναφέρεται σε αντικείμενο με ακέραιο τύπο δεδομένων.



Όταν θέσουμε στη συνέχεια **y = x**, η μεταβλητή y αναφέρεται και αυτή στο 42. Δηλαδή και οι δυο μεταβλητές x, y έχουν συσχετιστεί με το 42. Στην πληροφορική αργκό λέμε ότι έχουν ένα δέσιμο (binding) με την τιμή 42.



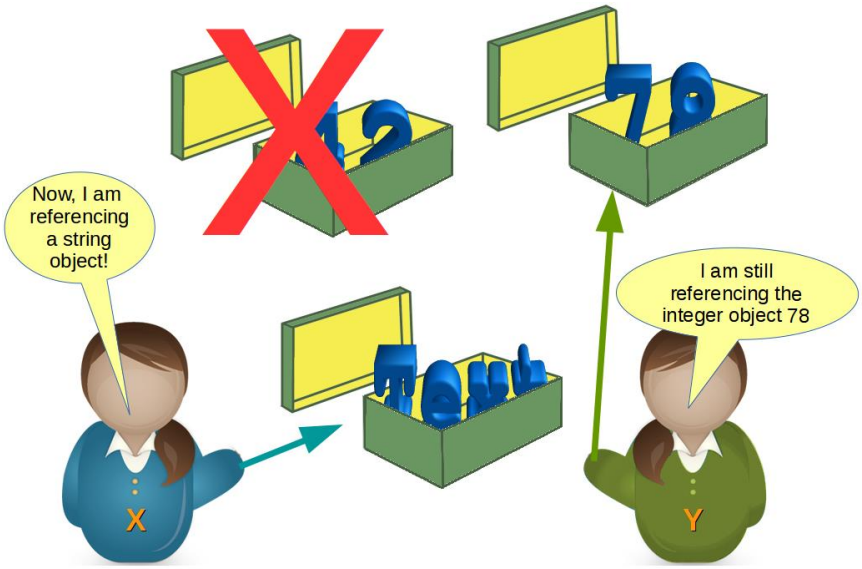
Όταν θέσουμε στη συνέχεια **y = 78**, η μεταβλητή y παύει πλέον να δείχνει το αντικείμενο με τιμή 42 και δημιουργείται ένα νέο "δέσιμο", ώστε να αναφέρεται στο αντικείμενο με τιμή 78.



Μπορούμε να θέσουμε μια μεταβλητή και σε αντικείμενο άλλου τύπου. Για παράδειγμα αν δώσουμε στη συνέχεια την εντολή:

```
>>> x = "Text"
```

η μεταβλητή x δείχνει πλέον σε ένα αντικείμενο τύπου string, δηλαδή σε μια λέξη. Δεν είναι πλέον ακέραιου τύπου (int) αλλά string (str), όπως φαίνεται στην παρακάτω εικόνα:



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).

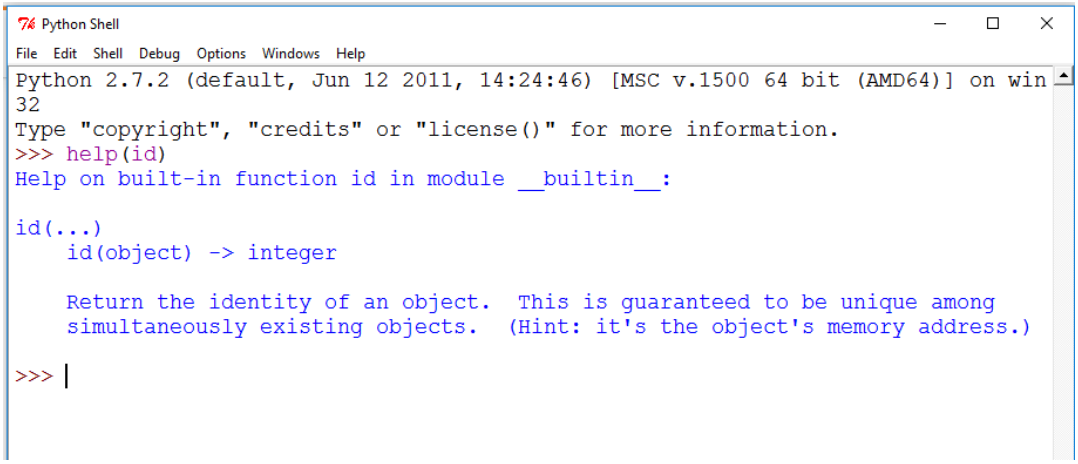
Αν θέλουμε να εμφανίσουμε την τιμή της μεταβλητής τότε μπορούμε να χρησιμοποιήσουμε την εντολή print μαζί με το όνομα της μεταβλητής (για το προηγούμενο παράδειγμα η εντολή print y θα εμφανίσει στην οθόνη 78).

```
>>> x = 28                  # εκχωρεί την ακέραια τιμή 28 στο x
>>> print x                 # εμφανίζει το περιεχόμενο της x
10
>>> print “x”                  # εμφανίζει το όνομα της x
x
>>>x=x+15                  # προσθέτει στο περιεχόμενο της x τον ακέραιο 15
>>> print x
25
>>>x=x *0.5               # πολλαπλασιάζει στο νέο περιεχόμενο της x τον αριθμό 0.5
>>> print x
2.5
>>> print x*100
250.0
>>> name = “Τένια”    # εκχωρεί τη λέξη ‘Τένια’ στη μεταβλητή name
>>>print name
Τένια
```

Χρησιμοποιούμε την εντολή print με την βοήθεια της οποίας εμφανίζουμε/εκτυπώνουμε ένα μήνυμα στην οθόνη. Είναι η εντολή με την οποία εμφανίζουμε τα αποτελέσματα του προγράμματος στον χρήστη. Την print θα την εξηγήσουμε στην επόμενη παράγραφο.

Η συνάρτηση id

Ένας τρόπος για να δούμε αν δυο αντικείμενα δείχνουν στην ίδια θέση μνήμης είναι να χρησιμοποιήσουμε τη συνάρτηση id. Η συνάρτηση αυτή μας επιστρέφει τη θέση στη μνήμη στην οποία είναι αποθηκευμένη η τιμή στην οποία αναφέρεται (με την οποία είναι "δεμένη") η μεταβλητή. Αν δώσουμε την εντολή help(id) στον διερμηνευτή της Python θα παρουμε πληροφορίες για την συνάρτηση id.



Γενικά όταν θέλουμε πληροφορίες για τη λειτουργία μιας εντολής ή συνάρτησης χρησιμοποιούμε την εντολή help όπως στο παραπάνω παράδειγμα. (φυσικά υπάρχει και το google ☺).

Να ερμηνεύσετε τα αποτελέσματα των παρακάτω εντολών.

```
>>> a = 28
>>> b = 28
>>> print id(a), id(b)
50683344  50683344
>>> a = a + 2
>>> print a
30
>>> print id(a)
50683296
>>> word = "zanneio"
>>> print id(word)
66002992
>>> word = word + " gymnasio"
>>> print  word
zanneio gymnasio
>>> print id(word)
50924264
```

Η εντολή x = x + 1

Στην παραπάνω εντολή εμφανίζεται η ίδια μεταβλητή και στις δυο πλευρές της εκχώρησης. Πρώτα υπολογίζεται η έκφραση που βρίσκεται στο δεξί μέρος δηλαδή (10+1) και στη συνέχεια το αποτέλεσμα (11) καταχωρείται στη μεταβλητή της εκχώρησης. Θα μπορούσαμε να γράψουμε την παραπάνω εκχώρηση λίγο πιο επεξηγηματικά ως εξής:

$$X_{\mu\varepsilon\tau\acute{\alpha}} = X_{\pi\rho\iota\nu} + 1$$

Μπορούμε να χρησιμοποιήσουμε αντί για την εντολή x = x + 1 την συντομη εντολή

x += 1

Εντολές Εισόδου και Εξόδου Δεδομένων

Σχετικά με την εισαγωγή τιμής σε μια μεταβλητή από το πληκτρολόγιο, κατάσταση όπου αναμένει από το χρήστη να εισάγει μια τιμή από το πληκτρολόγιο, την οποία την αποδίδει αυτόματα στη μεταβλητή, χρησιμοποιούμε δύο βασικές συναρτήσεις: την input() και την raw_input().

Εντολή	Επεξήγηση
x = input("Δώσε έναν αριθμό: ")	Διαβάζει από το πληκτρολόγιο έναν αριθμό από τον χρήστη και τον αποθηκεύει στη μεταβλητή **x**.
x = raw_input("Δώσε μια λέξη: ")	Διαβάζει από το πληκτρολόγιο μια λέξη και την αποθηκεύει στη μεταβλητή **x**.

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 μια μεταβλητής θα επιστρέψει το περιεχόμενο της μεταβλητής.

Για την **εισαγωγή σχολίων**, κατάσταση όπου μπορούμε να εισάγουμε επεξηγηματικά σχόλια στο πρόγραμμά μας, όπου θέτουμε μπροστά το σύμβολο #. Με αυτόν τον τρόπο όταν κάποιος δει το πρόγραμμά μας θα καταλάβει πιο εύκολα τι ακριβώς κάνει και πώς σκεφτήκαμε να το φτιάξουμε.

Για να εμφανίσουμε στην έξοδο (οθόνη) ένα μήνυμα ή το αποτέλεσμα μιας πράξης ή την τιμή μιας μεταβλητής χρησιμοποιούμε την εντολή print όπως φαίνεται παρακάτω:

```
>>> print “1 + 2 + 3”
1 + 2 + 3
>>> print 1 + 2 + 3
6
>>> print “1 + 2 + 3 = ”, 1 + 2 + 3
1 + 2 + 3 = 6
>>>  value = 28
>>> print “value”
value
>>> print  value
28
```

Αν θέλουμε να εμφανίσουμε πολλά μηνύματα στην ίδια γραμμή τα χωρίζουμε με κόμμα.

```
>>> print “1 + 2 + 3”
1 + 2 + 3
>>> print 1 + 2 + 3
6
>>> print “1 + 2 + 3 = ”, 1 + 2 + 3
1 + 2 + 3 = 6
>>>  value = 28
>>> print “value”
value
>>> print  value
28
```

Βασικές συναρτήσεις (ενσωματωμένες)

Η Python παρέχει μια ποικιλία **ενσωματωμένων συναρτήσεων** οι οποίες μετατρέπουν τιμές από έναν τύπο σε έναν άλλο.

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

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

Η γλώσσα Python διαθέτει μια Μαθηματική μονάδα λογισμικού (math module), η οποία περιέχει τις περισσότερο γνωστές μαθηματικές συναρτήσεις. Μια μονάδα ή άρθρωμα λογισμικού (module) είναι ένα αρχείο το οποίο περιέχει μια συλλογή από σχετικές συναρτήσεις. Προτού χρησιμοποιήσουμε μια μονάδα, πρέπει να την εισάγουμε: >>> import math.

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). Ας δούμε ένα παράδειγμα για τη συνάρτηση 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
```

Εκτός από τις ενσωματωμένες βιβλιοθήκες (μονάδες) συναρτήσεων που περιλαμβάνονται στη γλώσσα Python, μπορεί κανείς να βρει στους δικτυακούς τόπους υποστήριξης της γλώσσας και εξωτερικές μονάδες λογισμικού με πληθώρα επιπλέον συναρτήσεων για τη δημιουργία ποικίλων προγραμμάτων. Χαρακτηριστικό παράδειγμα είναι τα project για τη δημιουργία γραφικών και παιχνιδιών με ένα σύνολο πρόσθετων συναρτήσεων και έτοιμου λογισμικού. Στο επόμενο κεφάλαιο θα μάθουμε να δημιουργούμε τις δικές μας συναρτήσεις. Σε επόμενο κεφάλαιο θα δούμε επίσης διάφορα παραδείγματα από έτοιμες βιβλιοθήκες λογισμικού για να υλοποιούμε ελκυστικά προγράμματα στο χρήστη.

Δομή προγράμματος

Συνήθως ένα πρόγραμμα αποτελείται αποτελείται από τρία κύρια μέρη:

1. Εισαγωγή δεδομένων
2. Υπολογισμός αποτελεσμάτων
3. Εκτύπωση αποτελεσμάτων

Στη συνέχεια ακολουθεί ένα μικρό πρόγραμμα που υπολογίζει το εμβαδό ενός τριγώνου:

```
#Υπολογισμός του εμβαδού τριγώνου
base = input("Δώσε τη βάση του τριγώνου: " )
height = input("Δώσε το ύψος του τριγώνου: " )
area = ( base * height ) / 2.0
print “Το εμβαδό του τριγώνου είναι: ”, area
```

Δραστηριότητα

Να γραφεί αλγόριθμος και το αντίστοιχο πρόγραμμα σε γλώσσα Python, που να υπολογίζει το εμβαδό τετραγώνου πλευράς α. Το πρόγραμμα να διαβάζει από το πληκτρολόγιο το μήκος της πλευράς α, ζητώντας από το χρήστη να το πληκτρολογήσει. Στη συνέχεια να υπολογίζει το εμβαδόν του τετραγώνου και να εμφανίζει το αποτέλεσμα στην οθόνη με το ανάλογο μήνυμα.

2

Αλγοριθμικές Δομές

Εισαγωγή

Στο κεφάλαιο αυτό θα έχουμε την ευκαιρία να εξοικειωθούμε με τις αλγοριθμικές δομές και να υλοποιήσουμε πιο σύνθετα προγράμματα χρησιμοποιώντας τις αντίστοιχες δομές στη γλώσσα προγραμματισμού Python. Ένα πρόγραμμα λαμβάνει δεδομένα ως είσοδο, τα επεξεργάζεται σύμφωνα με τις εντολές που περιέχει και στη συνέχεια επιστρέφει τα αποτελέσματα. Τα βασικά χαρακτηριστικά ενός προγράμματος είναι:

Είσοδος: Περιλαμβάνει τα δεδομένα που απαιτεί το πρόγραμμα από το περιβάλλον για να παράγει την επιθυμητή έξοδο. Η είσοδος γίνεται συνήθως από περιφερειακές συσκευές, όπως ποντίκι, πληκτρολόγιο, αναγνώστη barcode, σαρωτής ή κάποιο αποθηκευτικό μέσο, όπως σκληρό δίσκο, usb flash memory, DVD, χρησιμοποιώντας την κατάλληλη εντολή. Στο προηγούμενο κεφάλαιο γνωρίσαμε τη συνάρτηση input(), η οποία διαβάζει δεδομένα από το πληκτρολόγιο .

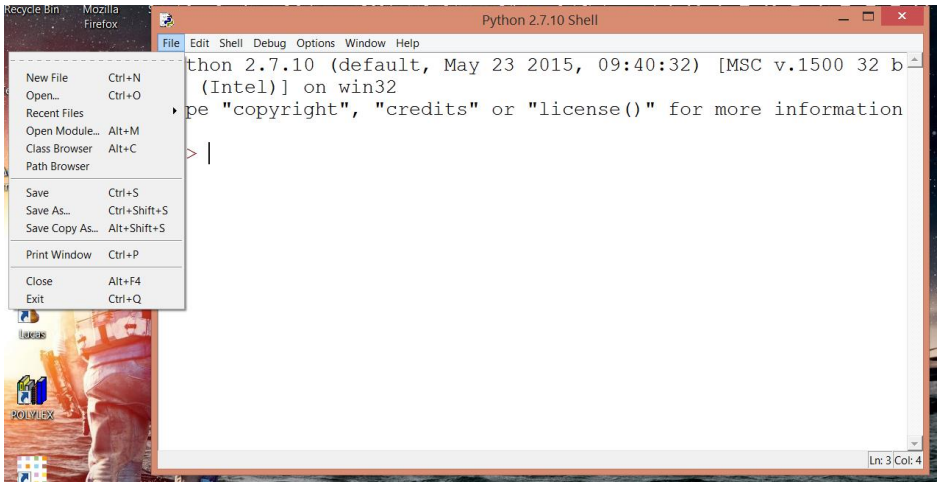
Εκτέλεση Υπολογισμών: Ο τρόπος με τον οποίο γίνεται η επεξεργασία των δεδομένων. εμπεριέχει μεταξύ άλλων τις ροές εκτέλεσης (ακολουθία, επιλογή, επανάληψη), τα μοντέλα υπολογισμού, τις συναρτήσεις, τις μεταβλητές.

Έξοδος: Περιλαμβάνει τα τελικά αποτελέσματα που παράγονται και συνήθως εμφανίζονται ή τυπώνονται για το χρήστη, με την έξοδο να πραγματοποιείται συνήθως στην οθόνη ή στον εκτυπωτή. Μπορεί να έχει τη μορφή κειμένου/αριθμού ή και γραφικών, όπως για παράδειγμα ένα πρόγραμμα που δημιουργεί ένα παιχνίδι. Ήδη, γνωρίσαμε στο προηγούμενο κεφάλαιο τη βασική εντολή εξόδου της γλώσσας προγραμματισμού Python την εντολή **print**.

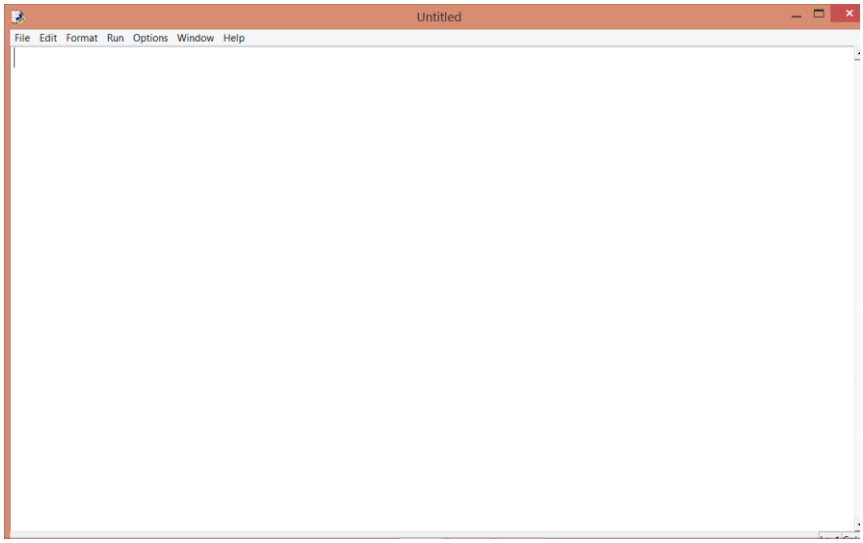
Σύνταξη και δημιουργία προγράμματος στο IDLE

Θα πρέπει να έχετε κάποια κάποια από τις εκδόσεις της Python 2. Η τελευταία έκδοση είναι η 2.7.13, την οποία θα κατεβάσετε, από τον σύνδεσμο: https://www.python.org/downloads . Στη συνέχεια κάνετε διπλό κλικ στο αρχείο που κατεβάσατε και γίνεται η εγκατάσταση. Καλό θα ήταν να μεταφέρετε μια συντόμευση στο IDLE από τα προγράμματα στην επιφάνεια εργασίας για να το βρίσκετε αμέσως.

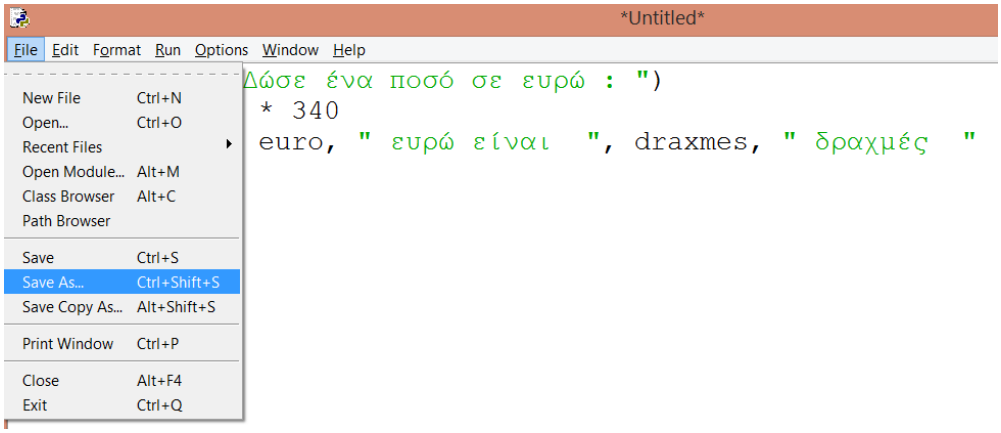
Στη συνέχεια ανοίγουμε το περιβάλλον IDLE της Python από τα προγράμματα (Έναρξη κλπ), οπότε θα δούμε την παρακάτω οθόνη, αφού επιλέξουμε το μενού File:



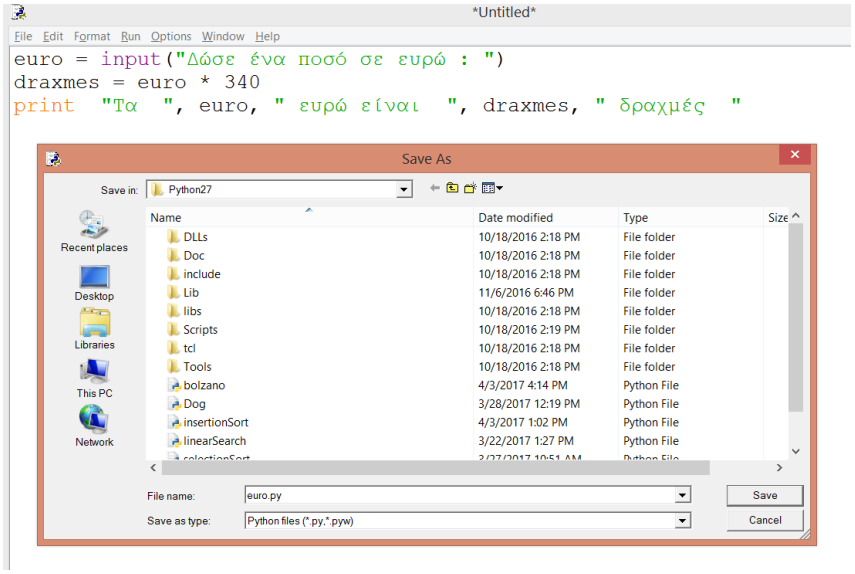
Επιλέγουμε New File και έτσι δημιουργείται ένα νέο αρχείο python, στο οποίο θα γράψουμε το νέο μας πρόγραμμα:



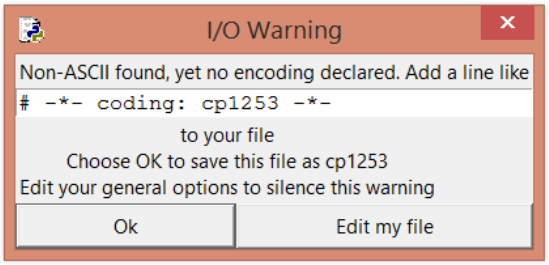
Αφού γράψουμε το πρόγραμμά μας πρέπει να το αποθηκεύσουμε στον δίσκο του υπολογιστή σε κάποιο φάκελο. Για αυτό επιλέγουμε File → Save As



και δίνουμε το όνομα του αρχείου euro.py αφού πρώτα επιλέξουμε σε ποιον φάκελο θα το αποθηκεύσουμε:

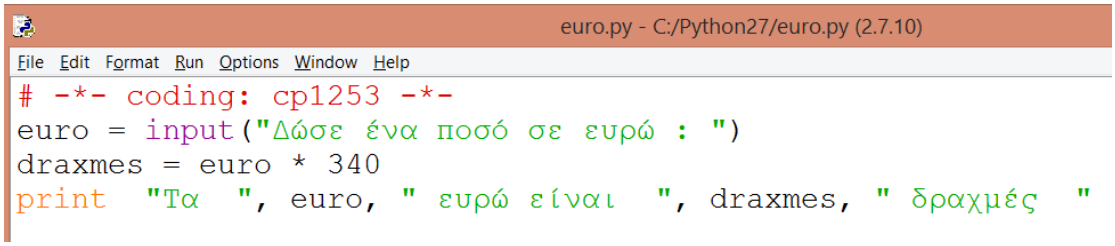


Αν έχετε γράψει ελληνικά στο πρόγραμμά σας, τότε το IDLE θα σας εμφανίσει το παρακάτω πλαίσιο διαλόγου, το οποίο σας λέει ότι πρέπει να προσθέσετε την παρακάτω γραμμή στο αρχείο σας.

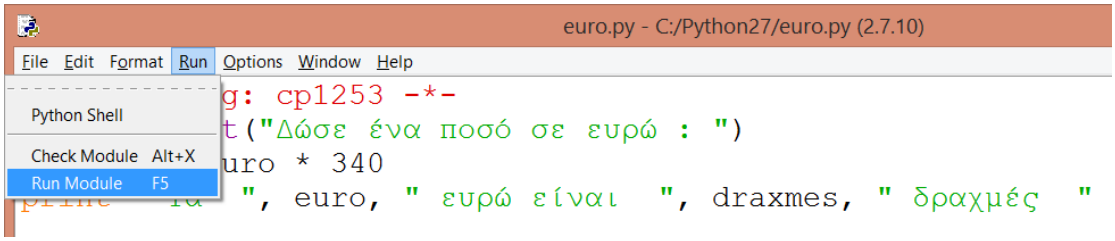


Θα πρέπει να επιλέξετε **Edit my file** και το IDLE θα προσθέσει αρχείου την παρακάτω γραμμή: # -*- coding: cp1253 -*-

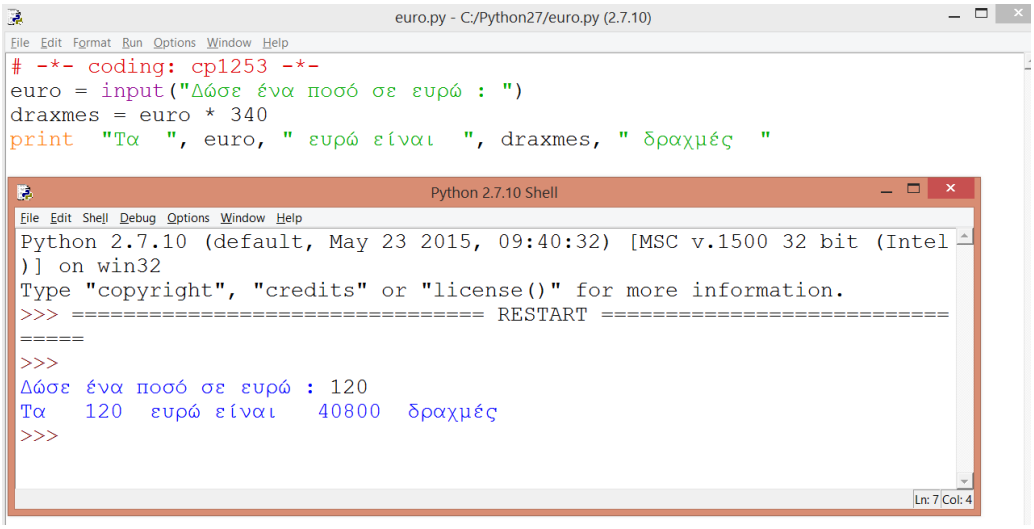
Η γραμμή αυτή δεν είναι εκτελέσιμη αλλά αποτελεί μια οδηγία στην Python ότι στο συγκεκριμένο αρχείο υπάρχουν ελληνικοί χαρακτήρες.



Στη συνέχεια το πρόγραμμά μας είναι έτοιμο και πατώντας F5 ή επιλέγοντας από το μενού Run → Run Module όπως φαίνεται παρακάτω:



Εφόσον το πρόγραμμά μας δεν έχει συντακτικά λάθη θα μεταφερθούμε στον διερμηνευτή όπου θα εκτελεστεί το πρόγραμμά μας:



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 1 : Μετατροπή ευρώ σε δραχμές

Το παρακάτω πρόγραμμα διαβάζει από το πληκτρολόγιο ένα ποσό σε ευρώ και το μετατρέπει σε δραχμές. Θυμίζουμε (για τους νέους) ότι ένα ευρώ είναι 340 δραχμές. Να γράψετε το παρακάτω πρόγραμμα στον συντάκτη (editor) του IDLE, να το αποθηκεύσετε σε ένα αρχείο euro.py και να το εκτελέσετε με F5, όπως δείξαμε στην προηγούμενη παράγραφο. Από εδώ και στο εξής εννοείται ότι σε κάθε αρχείο με ελληνικούς χαρακτήρες έχει προστεθεί στην αρχή η γραμμή: # -*- coding: cp1253 -*-

```
euro = input("Δώσε ένα ποσό σε ευρώ : ")
draxmes = euro * 340
print "Τα ", euro, " ευρώ είναι ", draxmes, " δραχμές "
```

Τώρα να γράψετε εσείς ένα αντίστοιχο πρόγραμμα που να εκτελεί την αντίστροφη μετατροπή, δηλαδή να διαβάζει ένα ποσό σε δραχμές και να το μετατρέπει σε ευρώ.

Παράδειγμα 2 : Ώρα για πίτσα

Μια πιτσαρία δίνει στις τρεις πίτσες τη μια δώρο. Έτσι για παράδειγμα αν παραγγείλουμε 7 πίτσες θα πληρώσουμε τις 5 αφού για κάθε τρεις πίτσες η μια είναι δωρεάν. Το παρακάτω πρόγραμμα διαβάζει από το πληκτρολόγιο πόσες πίτσες θα παραγγείλουμε, και την τιμή της πίτσας σε ευρώ και υπολογίζει το λογαριασμό.

```
euro = input("Δώσε ένα ποσό σε ευρώ : ")
draxmes = euro * 340
print  "Τα  ", euro, " ευρώ είναι  ", draxmes, " δραχμές  "
pitses = input("Πίτσες για παραγγελία = ")
price = input("Τιμή πίτσας = ")
free = pitses / 3                    # πόσες τριάδες έχει η παραγγελία
cost = (pitses - free) * price
print "Τελικό κόστος = ", cost
```

Παραπάνω χρειαζόμαστε ακέραια (ευκλείδεια) διαίρεση, αφού για να υπολογίσουμε πόσες είναι οι δωρεάν πίτσες θα πρέπει να υπολογίσουμε πόσες τριάδες από πίτσες έχουμε.

Δομή επιλογής if

Αν θέλουμε να εκτελεστεί μια ακολουθία εντολών, μόνον εφόσον πληρείται μία συγκεκριμένη συνθήκη, τότε χρησιμοποιούμε τη δομή επιλογής if (ΑΝ) με τη συνθήκη την οποία θέλουμε να ελέγξουμε. 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 το τέλος της if.

```
if  <συνθήκη ελέγχου>:
    <εντολές>

# πρόγραμμα εμφάνισης της απόλυτης τιμής ενός ακεραίου αριθμού
a=input('Δώσε ένα ακέραιο αριθμό ')
if a<=0:
    a=(-1)*a
print a
```

Αν ανάλογα με την αποτίμηση μιας συνθήκης θέλουμε να εκτελεστούν διαφορετικές εντολές, τότε μπορούμε να χρησιμοποιήσουμε τη δομή επιλογής **if…else** (ΑΝ…ΑΛΛΙΩΣ). Αν ισχύει η συνθήκη (True) θα εκτελεστεί το μπλοκ εντολών της if, αλλιώς, αν δεν ισχύει (False), θα εκτελεστεί το μπλοκ εντολών της else.

Η εντολή ελέγχου if … else συντάσσεται ως:

```
if <συνθήκη ελέγχου>:
    #γράψε τις εντολές εδώ
else:
    #γράψε τις εντολές εδώ
```

Σημείωση: Οι ομάδες εντολών που θα εκτελεστούν, αν ισχύει η συνθήκη, ορίζονται ως ένα μπλοκ με εσοχή (κενά διαστήματα) βάζοντας τη μία εντολή κάτω από την άλλη. Δεν πρέπει να διαγράψουμε τα κενά αυτά διαστήματα. Η Python προσφέρει τη δυνατότητα για σύνταξη σύνθετων δομών επιλογής με τη χρήση της εντολής elif. Η σύνταξη είναι ως εξής:

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

Δραστηριότητα. Δομή επιλογής if … else…

Να γραφεί πρόγραμμα που να διαβάζει τους βαθμούς των τετραμήνων ενός μαθητή στην πληροφορική, να ελέγχει αν ο μαθητής προβιβάζεται ή παραπέμπεται τον Σεπτέμβρη και να εμφανίζει κατάλληλο μήνυμα.

```
gradeA =  input (“Δώσε τον βαθμό του πρώτου τετραμήνου: ”)
gradeB =  input (“Δώσε τον βαθμό του δεύτερου τετραμήνου: ”)
final = ( gradeA + gradeB ) / 2.0
if   final >= 9.5 :
   print  “Συγχαρητήρια. Πέρασες.”
else:
   print  “Τα λέμε Σεπτέμβρη ☺ ”
```

Συνήθως έχουμε περισσότερες από δυο περιπτώσεις, οπότε χρειάζεται να συνεχίσουμε τους ελέγχους με εμφωλευμένες δομές επιλογής (if…else…) .

Τα παρακάτω προγράμματα ελέγχουν αν ένας αριθμός είναι αρνητικός, μονοψήφιος, διψήφιος ή μεγαλύτερος . Στο ένα χρησιμοποιούμε εμφωλευμένες if ενώ στο δεύτερο απλές if. Παρατηρήστε ότι στην πρώτη περίπτωση δεν χρειάζονται σύνθετες συνθήκες.

```
number =  input (“Δώσε έναν ακέραιο αριθμό: ”)

if  number < 0 :
   print  “Αρνητικός ”
else:
   if  number < 10 :
      print  “Μονοψήφιος ”
   else:
      if  number < 100 :
         print  “Διψήφιος ”
      else:
         print  “Πολυψήφιος ”

print  “Αριθμός ”
```

```
if  number < 0 :
   print  “Αρνητικός ”
if  number <= 0 and number < 10 :
      print  “Μονοψήφιος ”
if   number >= 10 and number < 100:
      print  “Διψήφιος ”
if   number >= 100:
      print  “Πολυψήφιος ”

print  “Αριθμός ”
```

Προσοχή!! Οι εσοχές έχουν σημασία! Από τις εσοχές ο διερμηνευτής καταλαβαίνει σε ποιο if και σε ποιο else αντιστοιχεί κάθε μπλοκ κώδικα. Δοκιμάστε να εκτελέσετε τα παρακάτω τμήματα κώδικα για να δείτε τη διαφορά:

```
number = 2017
if  number >= 0 :
   print  “Θετικός ή μηδέν”
else:
   print  “Αρνητικός ”

print  “Αριθμός ”
```

```
number = 2017
if  number > 0 :
   print  “Θετικός ή μηδέν”
else:
   print  “Αρνητικός ”

   print  “Αριθμός ”
```

Η εντολή **print** “Αριθμός ” θα εκτελεστεί σε κάθε περίπτωση στο αριστερό πλαίσιο ενώ στο δεξί πλαίσιο θα εκτελεστεί μόνο αν ο αριθμός είναι αρνητικός, αφού βρίσκεται μια εσοχή μέσα άρα ανήκει στο μπλοκ εντολών της else. Προσέξτε ότι ενώ οι εσοχές στον κώδικα έχουν μεγάλη σημασία, οι κενές γραμμές δεν έχουν καμία!

Όταν έχουμε πολλές περιπτώσεις για να μην δημιουργούνται πολλές εσοχές μπορούμε να χρησιμοποιήσουμε την εντολή πολλαπλών επιλογών if…elif…else , όπως φαίνεται παρακάτω:

```
number =  input (“Δώσε έναν ακέραιο αριθμό: ”)

if  number < 0 :
   print  “Αρνητικός ”
elif  number < 10 :
      print  “Μονοψήφιος ”
elif   number < 100:
      print  “Διψήφιος ”
else:
      print  “Πολυψήφιος ”
print  “Αριθμός ”
```

```
if  number < 0 :
   print  “Αρνητικός ”
if  number <= 0 and number < 10 :
      print  “Μονοψήφιος ”
if   number >= 10 and number < 100:
      print  “Διψήφιος ”
if   number >= 100:
      print  “Πολυψήφιος ”
print  “Αριθμός ”
```

Παρατηρήστε ότι στην δεύτερη περίπτωση έχουμε 4 απλές εντολές if ενώ στην πρώτη μια εντολή. Για αυτό τον λόγο στην πρώτη περίπτωση δεν χρειάζονται οι διπλές συνθήκες. Κάθε φορά που ένας ελέγχος είναι ψευδής (δεν ισχύει) προχωράμε στο επόμενο elif/else άρα δεν χρειάζεται να τον ξαναελέγξουμε. Για παράδειγμα όταν φτάσουμε στον έλεγχο number < 100 ξέρουμε ότι όλοι οι προηγούμενες συνθήκες δεν ισχύουν, αλλιώς δεν θα φτάναμε εκεί. Άρα δεν ισχύει το number < 10 , που σημαίνει ότι ισχύει το

not (number **<** 10) → number **>=** 10

άρα δεν χρειάζεται να το ξαναελέγξουμε.

Υπολογισμός μέγιστης τιμής 3 αριθμών:

Βασική ιδέα: Διαβάζουμε τους αριθμούς έναν – έναν. Κάθε φορά κρατάμε τον μεγαλύτερο. Αν ο νέος αριθμός που ήρθε είναι μεγαλύτερος από τον max τότε θα είναι μεγαλύτερος και από όσους έχουν περάσει ως τώρα άρα θέτουμε αυτόν ως max.

```
a = input ("a = ")
maximum = a

b = input ("b = ")
if b > maximum :
   maximum = b

c = input ("c = ")
if c > maximum :
   maximum = c

print "Η μεγαλύτερη τιμή είναι : ", maximum
```

```
a = input ("a = ")
b = input ("b = ")
c = input ("c = ")

if a >= b and a >= c :
    maximum = a
elif b >= a and b >= c :
    maximum = b
elif c >= a and c >= b :
    maximum = c
```

Αν θέλατε να υπολογίσετε το μέγιστο τεσσάρων (4) αριθμών ποιο από τα παραπάνω πρόγραμματα θα επεκτείνατε;

Λίστες

Μια λίστα είναι ουσιαστικά μια διατεταγμένη ακολουθία από αντικείμενα τα οποία συνήθως είναι του ίδιου τύπου.

```
>>> teams = ['ΑΕΚ', 'ΠΑΟ', 'ΟΣΦΠ', 'ΠΑΝΙΩΝΙΟΣ' ]
>>> numbers = [6, 28, 496, 8128]
```

Μπορεί όμως να περιέχει και αντικείμενα διάφορων τύπων:

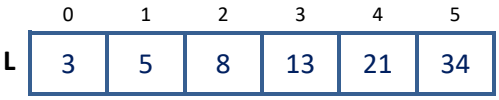
```
>>> values = [ 'zero', 13, 21, True ]
```

Για κάθε αντικείμενο που εισάγεται στη λίστα δίνεται ένας αύξων αριθμός, που χρησιμοποιείται για την αναφορά του στο αντικείμενο. Μπορεί κανείς ανά πάσα στιγμή να προσθέσει, να διαγράψει ή να αλλάξει ένα αντικείμενο σε μία λίστα. Η λίστα μπορεί να θεωρηθεί ως ένα σύνολο από αντικείμενα, οπότε μπορούμε να χρησιμοποιήσουμε τον τελεστή **in** για την ύπαρξη ενός στοιχείου στη λίστα και τη συνάρτηση **len** για να υπολογίσουμε το μέγεθος της λίστας.

```
>>> list1 = [10,20,30,40]
>>> fruits = ['apple', 'orange']
>>> len(list1)
4
>>> print fruits[0]
apple
>>> 'apple' in fruits
True
```

```
>>> list2 = [50,60,70,80]
>>> list1+ list2
[10, 20, 30, 40, 50, 60, 70, 80]
>>> list1 + fruits
[10, 20, 30, 40, 'apple', 'orange']
>>> list1[0]
10
```

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:



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 3ο, το 2ο το L[1] και τελευταίο το L[5].

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

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>
```

Αν θέλουμε να εμφανίσουμε όλα τα στοιχεία μιας λίστας μπορούμε να χρησιμοποιήσουμε τη επαναληπτική εντολή for με την οποία διατρέχουμε όλα τα στοιχεία μαις λίστας:

```
>>> fibonacci = [1, 1, 2, 3, 5, 8, 13, 21, 34]
>>> perfects = [6, 28, 496]

>>> for number in perfects :
        print number
6
28
496

>>> for number in fibonacci :
        print number ,
1 1 2 3 5 8 13 21 34
```

Παρατηρήστε ότι στην δεύτερη περίπτωση έχουμε προσθέσει ένα κόμμα στο τέλος της print έτσι ώστε μετά από κάθε εντολή ο επόμενος αριθμός να εμφανίζεται στην ίδια γραμμή με τον προηγούμενο.Θα μπορούσαμε να μεταφράσουμε τη λειτουργία της εντολής επανάληψης ως εξής:

```
for number in perfects :          Για κάθε αριθμό number στους perfects:
    print number                      Εκτύπωσε τον αριθμό number
```

Παράδειγμα 1. Μέσος όρων 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
```

Δομή Επανάληψης και η συνάρτηση range

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

```
>>> range( 4 )
[0,  1,  2,  3]
>>> range( 0, 4 )
[0, 1,  2,  3]
>>> range( 0, 4, 1 )
[0,  1,  2,  3]
>>> range( 1, 1, 100 )
[ ]
>>> range( 1, 5, –1 )
[ ]
>>> range( 0 )
[ ]
```

```
>>> range( 10, 30, 5 )
[10, 15, 20, 25]
>>> range( 30, 10, –5  )
[30, 25, 20, 15]
>>> range( 1, 2, 100 )
[1]
```

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
```

Τα παρακάτω τμήματα κώδικα είναι ισοδύναμα.

```
for item in [1,2,3,4,5,6,7,8] :
    print item ,
1 2 3 4 5 6 7 8
```

```
for item in range(1,9) :
    print item ,
1 2 3 4 5 6 7 8
```

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

```
sum = 0
for i in range(101) :
    sum = sum + i
print sum
5050
```

```
for index in range(5,51,5) :
        print index ,
5 10 15 20 25 30 35 40 45 50
```

Ας γυρίσουμε τώρα πίσω στην εύρεση μεγίστου 4 αριθμών που είχαμε θέσει ως άσκηση προηγουμένως. Η πρώτη μας σκέψη θα ήταν να χρησιμοποιήσουμε μια ακόμα μεταβλητή d. Δυστυχώς η χρήση διαφορετικών μεταβλητών μας αποτρέπει από τη γενίκευση του προβλήματος και τη χρήση επανάληψης που θα οδηγήσει στη συνοπτική γραφή του συγκεκριμένου τμήματος κώδικα. Αφού δεν μας ενδιαφέρει να αποθηκεύσουμε για περαιτέρω επεξεργασία τους αριθμούς που διαβάζουμε παρά μόνο η εύρεση της μέγιστης τιμής μπορούμε να χρησιμοποιούμε κάθε φορά την ίδια μεταβλητή a. Έτσι είναι πλέον φανερό ότι ένα τμήμα του κώδικα επαναλαμβάνεται και έτσι μπορούμε να ξαναγράψουμε το πρόγραμμα με τη χρήση της for.

Χρήση πολλών μεταβλητών	Χρήση μιας μόνο μεταβλητής	Δομή Επανάληψης

a = input (“a = ”)
maximum = a
 	
b = input (“b = ”)
if b > maximum :
  maximum = b
 	
c = input (“c = ”)
if c > maximum :
  maximum = c
 	
d = input (“d = ”)
if d > maximum :
  maximum = d
	a = input (“a = ”)
	maximum = a
	
	a = input (“a = ”)
	if a > maximum :
	  maximum = a
	
	a = input (“a = ”)
	if a > maximum :
	  maximum = a
	
	a = input (“a = ”)
	if a > maximum :
	  maximum = a
		a = input (“a = ”)
		maximum = a
		
		for i in range(3):
		  a = input (“a = ”)
		  if a > maximum :
		  maximum = a

Δομή επανάληψης while

Υπάρχουν και περιπτώσεις που δεν γνωρίζουμε εξαρχής το πλήθος των επαναλήψεων που θα εκτελεστούν. Σε αυτές τις περιπτώσεις χρησιμοποιούμε τη δομή επανάληψης while. Το παρακάτω τμήμα κώδικα διαβάζει από το πληκτρολόγιο αριθμούς μέχρι να δοθεί 0 και στη συνέχεια εμφανίζει πόσοι αριθμοί δόθηκαν. Για τον σκοπό αυτό χρησιμοποιούμε μια μεταβλητή η οποία αυξάνεται σε κάθε επανάληψη. Θα λέμε ότι αυτή η μεταβλητή παίζει το ρόλο του μετρητή της επανάληψης.

```
counter = 0
number = input("number = ")
while  number != 0  :
    counter = counter + 1
    number = input("number = ")
```

Γενική μορφή της while :

```
while <συνθήκη είναι αληθής> :
    <Εντολές >
```

Τα παρακάτω προγράμματα υπολογίζουν το άθροισμα όλων των ζυγών αριθμών από το 10 μέχρι και το 1000.

```
sum = 0

for number in range(10,1001, 2) :
    sum = sum + number

print number
```

```
sum = 0
number = 10
while number < 1001 :
    sum = sum + number
    number = number + 2
print number
```

Όταν γνωρίζουμε εκ των προτέρων πόσες επαναλήψεις θα γίνουν μπορούμε να χρησιμοποιήσουμε είτε τη while είτε τη for ανάλογα ποια μας βολεύει καλύτερα. Φυσικά μέσα στη for είναι "κρυμμένες" η αρχικοποίηση της μεταβλητής number όπως και η αύξησή της κατά 2.

Δραστηριότητα : *Εισαγωγή στις Συναρτήσεις*

Δημιουργήστε ένα αρχείο στο IDLE και γράψτε το παρακάτω πρόγραμμα το οποίο υπολογίζει και εμφανίζει το άθροισμα όλων των αριθμών από το 1 έως το N:

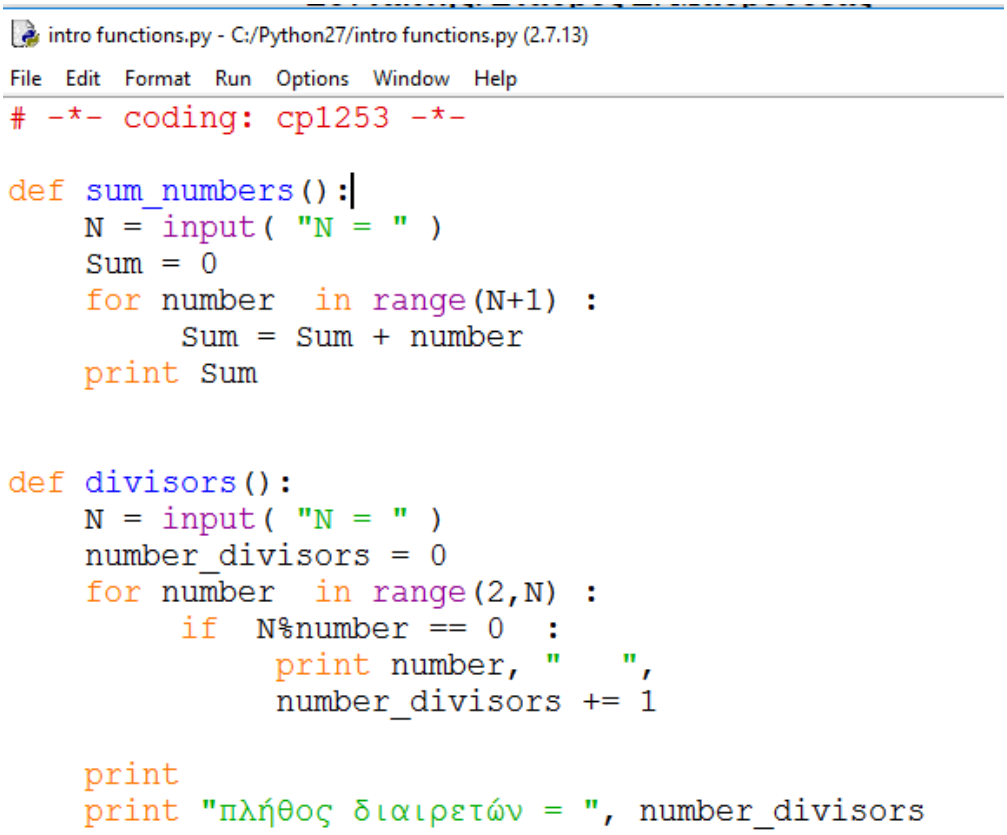
```
N = input( "N = " )
Sum = 0
for number  in range(N+1) :      # Για να προσθέσει και το N
    Sum = Sum + number
print Sum
```

Αφού το εκτελέσετε με F5 και βεβαιωθείτε ότι λειτουργεί σωστά αφήστε 2 κενές γραμμές και στη συνέχεια γράψτε το παρακάτω πρόγραμμα το οποίο εμφανίζει όλους τους γνήσιους διαιρέτες του αριθμού N εκτός του 1 στην ίδια γραμμή και υπολογίζει και το πλήθος τους:

```
N = input( "N = " )
divisors = 0
for number  in range(N) :
    if  N%number == 0  :
        print number, "    " ,
        divisors += 1              # όποτε βρίσκει έναν διαιρέτη αυξάνεται κατά ένα
print
print divisors
```

Τώρα αν εκτελέσετε με F5 το αρχείο θα εκτελεστούν και τα δυο προγράμματα με τη σειρά. Εμείς όμως θα θέλαμε να εκτελούμε μόνο ένα από τα δυο όποτε εμείς θέλουμε. Για να τα το πετύχουμε αυτό ένας απλός τρόπος είναι να γράψουμε κάθε πρόγραμμα σε ξεχωριστό αρχείο. Έτσι όμως θα γεμίσουμε με πολλά μικρά αρχεία. Πως θα καταφέρουμε να μαζέψουμε όλα τα προγράμματα/ασκήσεις μας σε ένα αρχείο?

Μπορούμε να αντιστοιχήσουμε σε κάθε τμήμα κώδικα ένα νέο όνομα, κάτι σαν παραπομπή. Κάθε φορά που θα γράφουμε αυτό το όνομα θα εκτελείται τα αντίστοιχο πρόγραμμα. Ουσιαστικά δημιουργούμε μια νέα εντολή για κάθε τμήμα κώδικα. Για να γίνει αυτό θα πρέπει να μετακινήσουμε και τα δυο τμήματα κώδικα μια εσοχή δεξιά. Μαρκάρουμε τον κώδικα και είτε πατάμε **Ctrl +]** ή από το μενού επιλέγουμε Format → Indent Region. Στη συνέχεια πάνω από κάθε τμήμα κώδικα γράφουμε τη δήλωση **def** με το όνομα της αντίστοιχης εντολή ακολουθούμενο από το σύμβολο **:** το οποίο μας ενημερώνει ότι ακολουθεί ένα εμφωλευμένο μπλοκ κώδικα. Στο τέλος το αρχείο μας θα φαίνεται όπως παρακάτω.



Μια εκτέλεση των νέων εντολών μας στον διερμηνευτή φαίνεται παρακάτω:

```
>>>
================== RESTART: C:/Python27/i
ntro functions.py ==================
>>> sum_numbers()
N = 100
5050
>>> sum_numbers()
N = 10000000
50000005000000
>>> divisors()
N = 20
2     4     5     10
πλήθος διαιρετών =  4
>>> divisors()
N = 1000
2     4     5     8     10     20     25
   40     50     100     125     200
250     500
πλήθος διαιρετών =  14
>>>
```

Πλέον μπορούμε να εκτελέσουμε τα δυο τμήματα κώδικα όσες φορές θέλουμε και όποτε θέλουμε. Έχουμε ορίσει δυο **συναρτήσεις**! Οι συναρτήσεις είναι λοιπόν ονόματα τα οποία έχουμε δώσει σε τμήματα κώδικα έτσι ώστε να μπορούμε να τα εκτελούμε όταν θέλουμε. Θεωρήστε τα σαν αυτόνομα μικρά προγραμματάκια. Εδώ απλά σημειώστε ότι κάθε συνάρτηση έχει τη δική της μεταβλητή N. Δηλαδή το N της divisors είναι σε διαφορετική θέση μνήμης από το N της sum_numbers.

Συναρτήσεις

Οι συναρτήσεις αποτελούν ένα από τα πιο σημαντικά δομικά στοιχεία ενός προγράμματος σε όλες τις γλώσσες προγραμματισμού. Οι συναρτήσεις μπορεί να είναι είτε έτοιμες από τη γλώσσα προγραμματισμού, είτε να δημιουργούνται από εμάς, όπως έγινε προηγουμένως.

Οι συναρτήσεις είναι επαναχρησιμοποιήσιμα τμήματα κώδικα. Μας επιτρέπουν να δίνουμε ένα όνομα σε ένα σύνολο εντολών και να το εκτελούμε καλώντας το όνομά τους, οπουδήποτε στο πρόγραμμα και όσες φορές θέλουμε. Αυτή η διαδικασία ονομάζεται κλήση (call) της συνάρτησης. Στα παραπάνω προγράμματα χρησιμοποιήσαμε αρκετές ενσωματωμένες συναρτήσεις, όπως την int() και τη range.

Οι συναρτήσεις ορίζονται χρησιμοποιώντας τη χαρακτηριστική λέξη **def**, από το define που σημαίνει ορίζω, στη συνέχεια ακολουθεί ένα όνομα που ταυτοποιεί την εκάστοτε συνάρτηση και μετά προσθέτουμε ένα ζευγάρι παρενθέσεων που μπορούν να περικλείουν μερικά ονόματα μεταβλητών και η γραμμή τελειώνει με διπλή τελεία (:). Ας δούμε ένα απλό παράδειγμα:

```
# Δέχεται σαν όρισμα έναν αριθμό x και
# επιστρέφει το τετράγωνό του
def square( x ) :
    return  x*x

# Υπολογίζει την 3η δύναμη του x
def cube( x ) :
    return  x*square(x)
```

```
>>> square(3)
9
>>> cube(2)
8
>>> square(cube(2))
64
>>> cube(square(2))
64
```

Ας ορίσουμε τη συνάρτηση printPython3, ώστε να εμφανίζει τη λέξη Python τρεις (3) φορές. Στη συνέχεια ορίζουμε τη συνάρτηση printPython9, ώστε να εμφανίζει τη λέξη Python εννέα (9) φορές.

```
def  printPython3():
    print 'Python'
    print 'Python'
    print 'Python'
```

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

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

Πώς μπορούμε να γράψουμε τη συνάρτηση printPython9 χρησιμοποιώντας λιγότερες εντολές; Πώς μπορούμε να ορίσουμε μια συνάρτηση η οποία να εμφανίζει 21 φορές τη λέξη Python, με αποκλειστική χρήση των συναρτήσεων printPython3() και printPython9().

Η ανάγκη για γενίκευση

Δίνονται οι παρακάτω συναρτήσεις σε Python. Η συνάρτηση που δίνεται δεξιά είναι η γενίκευση των τριών, αφού ανάλογα με την τιμή του N μπορούμε να σχηματίσουμε όποια συνάρτηση θέλουμε. Το N λέγεται παράμετρος της συνάρτησης.

```
def printPython3( ):
    for i in range(3):
        print 'Python'

def printPython12( ):
    for i in range(12):
        print 'Python'

def printPython100( ):
    for i in range(100):
        print 'Python'
```

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

>>> printPython(3)
>>> printPython(9)
>>> printPython(21)
>>> printPython(100)
```

Οι εσοχές

Οι εσοχές στην αρχή των εντολών είναι πολύ σημαντικές στην Python. Ο κενός χώρος πριν από μια εντολή και γενικότερα η στοίχιση των εντολών, δεν είναι μόνο θέμα αισθητικής, όπως σε άλλες γλώσσες, αλλά θέμα ουσίας που μπορεί να αλλάξει το αποτέλεσμα του προγράμματος, όπως φαίνεται παρακάτω:

```
>>> def print2():
        print "********"
        print "********"
>>>  print2()
********
********
```

```
>>> def print2():
        print "********"
      print "********"
********
>>>  print2()
********
```

Στη δεύτερη περίπτωση, στη δεξιά στήλη, η δεύτερη εντολή print βρίσκεται έξω από τη συνάρτηση και εκτελείται κανονικά. Η συνάρτηση εμφανίζει μόνο μια γραμμή με αστέρια και όχι δυο όπως θέλουμε. Δεν υπάρχει δηλαδή κάποια ειδική εντολή που να υποδηλώνει το τέλος του μπλοκ εντολών της συνάρτησης, όπως π.χ. τέλος_συνάρτησης, end_def. Όλα εξαρτώνται από τη στοίχιση των εντολών**, άρα πρέπει να είμαστε ιδιαίτερα προσεκτικοί με τη στοίχιση των εντολών** και την εσοχή πριν από κάθε εντολή, ώστε να εξασφαλίσουμε πως ανήκει στο σωστό μπλοκ.

Ορισμός Συνάρτησης

Όπως είδαμε ο ορισμός συναρτήσεων στην Python είναι αρκετά απλός:

```
def  <όνομα συνάρτησης> ( [ {λίστα παραμέτρων} ] ):
       εντολές
     [ return  <αποτέλεσμα> ]
```

Οι αγκύλες [ ] σημαίνουν, πως ότι περικλείεται μέσα σε αυτές είναι προαιρετικό. Η λίστα των παραμέτρων μπορεί να είναι κενή. Μια συνάρτηση δεν είναι υποχρεωτικό να επιστρέφει κάποια τιμή.

Δραστηριότητα

Ορίστε τις δύο συναρτήσεις, με ονόματα add και times3, όπως φαίνεται παρακάτω, και πειραματιστείτε με τα διαφορετικά παραδείγματα κλήσης τους στο περιβάλλον της Python:

```
def add(arg1, arg2):
    result = arg1+arg2
    return result
def times3(arg):
    ginomeno = 3*arg
    return ginomeno
```

```
>>> add(10, 18)
28
>>> add(10, 18.5)
28.5
>>> times3(10)
30
```

Μπορούμε να συνδυάσουμε την κλήση συναρτήσεων, με το σκεπτικό ότι το αποτέλεσμα της μιας συνάρτησης μπορεί να αποτελέσει τα δεδομένα εισόδου μιας άλλης.

```
>>> times3(2.5)
7.5
>>> times3('python')
'python python python'
>>> times3( times3( '9' ) )
'9 9 9 9 9 9 9 9 9 '
>>> times3(add(add('ab','ba'),' '))
'abba  abba  abba'
```

Παρατηρήστε ότι έχουμε ορίσει μια συνάρτηση, η οποία δέχεται όλους τους τύπους των ορισμάτων και η λειτουργία της αναπροσαρμόζεται δυναμικά ανάλογα με αυτά, οπότε αν δοθούν αριθμοί τους προσθέτει, ενώ τα αλφαριθμητικά τα συνενώνει.

Παράμετροι συναρτήσεων

Μια συνάρτηση μπορεί να δεχθεί παραμέτρους, οι οποίες χρησιμοποιούνται για να δίνουμε διάφορες τιμές στη συνάρτηση, έτσι ώστε αυτή να παράγει κάποιο αποτέλεσμα ή να εκτελεί κάποιες ενέργειες χρησιμοποιώντας τις τιμές αυτές. Αυτές οι παράμετροι μοιάζουν με τις μεταβλητές, διαφέροντας ως προς το ότι οι τιμές αυτών των μεταβλητών ορίζονται όταν καλούμε τη συνάρτηση και τους έχουν ήδη εκχωρηθεί τιμές όταν τρέχει η συνάρτηση.

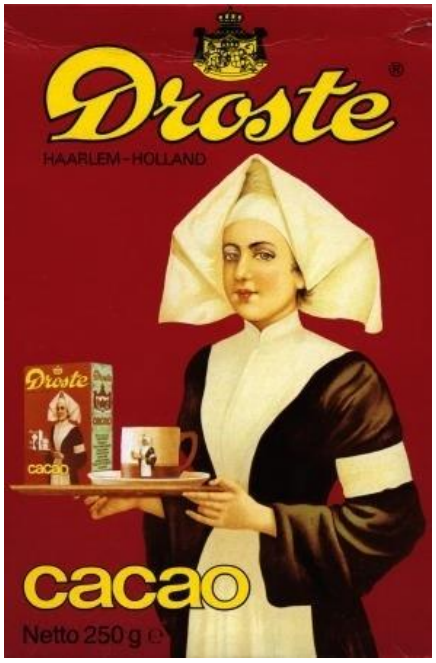
Οι παράμετροι καθορίζονται μέσα στο ζευγάρι των παρενθέσεων στον ορισμό της συνάρτησης και διαχωρίζονται με κόμμα. Όταν καλούμε τη συνάρτηση δίνουμε και τις τιμές με τον ίδιο τρόπο. Σημείωση για την ορολογία που χρησιμοποιείται: οι ονομασίες που δίνουμε στον ορισμό της συνάρτησης ονομάζονται παράμετροι, ενώ οι τιμές που δίνουμε όταν καλούμε τη συνάρτηση ονομάζονται ορίσματα.

```
def hypotenuse (a,b):
   from math import sqrt
   return sqrt( a*a + b*b )

>>> hypotenuse( 3, 4 )
5.0
>>> hypotenuse( 1, 1 )
1.4142135623730951
>>> hypotenuse( 1, 1 ) * hypotenuse( 1, 1 )
2.0000000000000
```

Χρησιμοποιούμε τη συνάρτηση sqrt (τετραγωνική ρίζα) της βιβλιοθήκης math της Python. Πριν την χρησιμοποιήσουμε όμως πρέπει να το δηλώσουμε στην Python και να τις δείξουμε σε ποια βιβλιοθήκη θα τη βρει. Αυτό κάνει η δήλωση from math import sqrt.

Αναδρομή



Μπορούμε στην Python να ορίσουμε μια συνάρτηση με αναδρομικό τρόπο όπως φαίνεται παραπάνω. Η Python θα κάνει τα υπόλοιπα.

```
def recursive_sum( N ):
    if N == 0 :
        return 0
    else :
        return recursive_sum(N–1) + N

>>> recursive_sum( 100 )
5050
>>> recursive_sum( 3 )
6
```

```
def fibonacci( N ):
    if N <=2 :
        return 1
    else :
        return fibonacci(N–1) + fibonacci(N-2)

>>> fibonacci( 10 )
5050
>>> fibonacci( 6 ), fibonacci( 7 ), fibonacci( 8 )
( 8, 13, 21 )
```

3

Δομές Δεδομένων

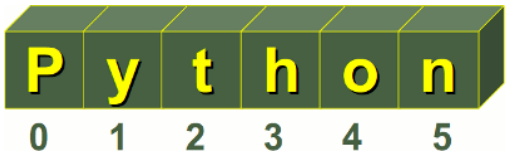
Εισαγωγή

Μια δομή δεδομένων είναι ένα σχήμα οργάνωσης των δεδομένων τα οποία χρησιμοποιεί το πρόγραμμά μας, με την επιλογή της κατάλληλης δομής δεδομένων να παίζει σημαντικό ρόλο στην ανάπτυξη του αλγορίθμου για την επίλυση ενός προβλήματος. Οι βασικές δομές δεδομένων της Python που θα εξετάσουμε στο κεφάλαιο αυτό είναι οι **συμβολοσειρές**, οι **λίστες**, οι **πλειάδες**, τα **σύνολα** και τα **λεξικά**.

Η λίστα αποτελεί τη βασική δομή δεδομένων της Python, ενώ η δομή του λεξικού χρησιμεύει όταν θέλουμε να κάνουμε γρήγορη αναζήτηση και ανάκληση πληροφορίας σχετική με κάποια συμβολοσειρά. Μια άλλη ενσωματωμένη δομή είναι η **πλειάδα** (tuple), η οποία μοιάζει με μια λίστα αλλά είναι δομή που δεν μπορεί να τροποποιηθεί. Ο συνδυασμός λεξικού και πλειάδας ή λίστας έχει πολλές εφαρμογές, όπως για παράδειγμα στην αναπαράσταση **γράφων**.

Συμβολοσειρές – Αλφαριθμητικά

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:



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:

```
>>> word = “PYTHON”
>>> len( word )
6
>>> print  word[5]+word[0]
NP
>>> str(28) == ‘28’
True
>>> print int( ‘496’ ) + 4
500
```

- 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.

```
>>> "Py" in "Python"
True
>>> "a" in "Python"
False
>>> "a" not in "Python"
True
```

```
>>> 'antonis' > 'antonia'
True
>>> '1000' < '2'
True
>>> 'babylon5' > 'babylon4'
True
```

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 = "AEIOUaeiou"
    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 μία 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.

Μπορούμε να πάρουμε ένα τμήμα της συμβολοσειράς με χρήση του τελεστή διαμέρισης (slice operator) ":". Όταν γράφουμε word[αρχή:τέλος] επιστρέφεται το μέρος της συμβολοσειράς που ξεκινάει από τον χαρακτήρα στη θέση αρχή μέχρι τη θέση τέλος, χωρίς να περιλαμβάνει τον χαρακτήρα της θέσης τέλος.

```
>>> s = 'Monty Python'
>>> s[0:5]
Monty
>>> s[6:len(s)]
Python
```

```
>>> s[len(s)-1]
'n'
>>> s[-1]
'n'
>>> s[0:len(s)]
Monty Python
```

Τα αντικείμενα του τύπου str δεν είναι τροποποιήσιμα, είναι όπως λέμε αμετάβλητα (immutable). Δηλαδή δεν μπορούμε να αλλάξουμε μέρος της συμβολοσειράς. Για παράδειγμα η εντολή word[0] = 'p' θα μας επιστρέψει λάθος, αφού δεν μπορεί να εκτελεστεί. Οποιαδήποτε επεξεργασία θέλουμε να κάνουμε σε συμβολοσειρές γίνεται με χρήση του τελεστή : για αποκοπή του μέρους που θέλουμε και του + για συνένωση.

Ένα εξαιρετικά ενδιαφέρον χαρακτηριστικό της Python είναι η πολυμορφική συμπεριφορά της, η οποία φαίνεται στο παρακάτω παράδειγμα:

```
>>> def times( a, b ):
        return a * b
```

```
>>> times( 2, 3 )
6
>>> times( "python#", 3 )
python#python#python#
```

Θα λέγατε ότι ισχύει η παρακάτω σχέση;

"Python" + "Python" + "Python" = 3 * "Python" .

Είναι φανερό λοιπόν ότι όλες οι αποφάσεις λαμβάνονται σε χρόνο εκτέλεσης, δηλαδή την τελευταία στιγμή (lazy evaluation). Αυτό προσδίδει στη γλώσσα τη δυναμική της συμπεριφορά.

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 3ο, το 2ο το L[1] και 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.

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.

```
>>> fruits = ['apple', 'orange']
>>> len(fruits)
2
>>> print fruits[0]
apple
>>> 'apple' in fruits
True
>>> powers = [2, 4, 8, 16]
```

```
>>> 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]
```

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 μία 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]
```

Α

ν 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 όρων 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.append( number )
        # πρόσθεσέ τον στη λίστα με τους θετικούς
    else :
        negatives.append( 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.

Ένας πολύ χρήσιμος τελεστής είναι ο **τελεστής διαμέρισης (slice operator)** που συμβολίζεται με την άνω κάτω τελεία **:**, και ο οποίος έχει αναφερθεί στις παραγράφους 5.2, 5.3 του βιβλίου της Β’ τάξης. Ο τελεστής διαμέρισης μπορεί να μας επιστρέψει ένα τμήμα μιας συμβολοσειράς ή μιας λίστας word. Η έκφραση word[a : b] μας επιστρέφει το τμήμα της συμβολοσειράς ή της λίστας από το στοιχείο word[a] μέχρι και το στοιχείο word[b-1]. Μπορούμε να παραλείψουμε την αρχή ή το τέλος όπως φαίνεται στα παραδείγματα παρακάτω αν θέλουμε ένα τμήμα από την αρχή ή το τέλος της λίστας/συμβολοσειράς αντίστοιχα.

Δίνονται τα παρακάτω παραδείγματα:

```
>>> word = “zanneio gymnasio”
>>> word[ :7 ]
‘zanneio’
>>> word[ 8: ]
‘zanneio’
>>> word[ : ]
‘zanneio gymnasio’
>>> word[ 3:11 ]
‘neio gym’
>>> word[ 0:7 ]
‘zanneio’
>>> word[ 8:len(word) ]
‘gymnasio’
>>> word[ 0:len(word) ]
‘zanneio gymnasio’
```

Ο τελεστής **:** είναι πολύ σημαντικός στην επεξεργασία λιστών στην Python, γιατί μπορούμε να τον χρησιμοποιήσουμε για να πάρουμε ένα αντίγραφο μιας λίστας όπως παρακάτω

```
>>> fibonacci = [5,8,13,21,34]
>>> fib = fibonacci[ : ]      # δημιουργεί αντίγραφο
>>> a = fib          # δείχνουν στην ίδια λίστα
>>> a.pop()        # οι αλλαγές επηρεάζουν και την a και τη fib
34
>>> fib.pop()
21
>>> a[0]=a[1]=55
>>> print a
[55, 55, 13]
>>> print fib
[55, 55, 13]
>>> print fibonacci    # Η  αρχική λίστα fibonacci δεν έχει αλλάξει
[5, 8, 13, 21, 34]
```

Προσοχή!!! Δεν μπορεί να γίνει το ίδιο με τις συμβολοσειρές. Δηλαδή η εντολή word[**:**] δεν δημιουργεί αντίγραφο της συμβολοσειράς word. Αυτό μπορείτε να το διαπιστώσετε με χρήση της συνάρτησης id που δείχνει τη διεύθυνση στη μνήμη του κάθε αντικειμένου.

Ο τελεστής διαμέρισης μπορεί να φανεί πολύ χρήσιμος σε κάποιες περιπτώσεις όπως για παράδειγμα στην δραστηριότητα 8 του κεφαλαίου 5 του βιβλίου της Β’ τάξης όπου ζητείται η υλοποίηση του αλγορίθμου κρυπτογράφησης του Καίσαρα. Αν κάθε γράμμα αντιστοιχίζεται με το γράμμα που βρίσκεται 3 θέσεις μπροστά τότε το νέο αλφάβητο μπορεί να προκύψει με τις παρακάτω εντολές:

```
>>> alphabet = “ABCDEFGHIJKLMNOPQRSTUVWXYZ”
>>> cipherAlphabet = alphabet[ 3 : ] + alphabet[ : 3]
>>> print  cipherAlphabet
DEFGHIJKLMNOPQRSTUVWXYZABC
```

Σημαντική σημείωση

Η εντολή

aList = aList + [item]

κάθε φορά που καλείται δημιουργεί μια νέα λίστα ως προϊόν συνένωσης των δυο λιστών η οποία αποθηκεύεται σε μια νέα θέση στη μνήμη. Αυτή η λειτουργία έχει υπολογιστικό κόστος σε αντίθεση με την εντολή

aList.append(item) ή την εντολή aList += [item]

η οποία προσθέτει το στοιχείο item στο τέλος της λίστας.

Πλειάδες

Ένας σύνθετος τύπος του οποίου τα στοιχεία είναι *αμετάβλητα* (immutable), είναι η **πλειάδα** (tuple). Την χρησιμοποιούμε όταν θέλουμε να συγκρατήσουμε μαζί πολλαπλά αντικείμενα. Για παράδειγμα, θέλουμε να έχουμε σε μια λίστα τα στοιχεία των υπαλλήλων μιας επιχείρησης, όπως όνομα, επώνυμο, βαθμός, τηλέφωνο, τμήμα, μισθός κ.λπ. Αυτό θα το υλοποιήσουμε με μια λίστα από πλειάδες, όπου κάθε πλειάδα θα αντιστοιχεί σε έναν εργαζόμενο. Η πλειάδα μοιάζει με μια λίστα όπως φαίνεται στα παραδείγματα παρακάτω:

```
>>> rec = 'a', 'b', 120, 'r'
>>> print rec
('a', 'b', 120, 'r')
>>> rec[2]
120
>>> len( rec )
4
>>> rec[1:3]
('b',  120)
>>> rec = ('apple' ,) + rec[2:3]
>>> print rec
( 'apple', 120 )
```

Συνήθως οι πλειάδες χρησιμοποιούνται για την αναπαράσταση μιας σειράς χαρακτηριστικών/ιδιοτήτων μιας οντότητας, εφόσον τα χαρακτηριστικά αυτά δεν μεταβάλλονται. Μια σημαντική εφαρμογή των πλειάδων είναι ότι μπορούμε να ορίσουμε συναρτήσεις με ορίσματα μεταβλητού μεγέθους.

Μια άλλη γνωστή εφαρμογή τους είναι **η αντιμετάθεση των τιμών δύο μεταβλητών** χωρίς τη χρήση βοηθητικής μεταβλητής, με την παρακάτω εντολή:

```
>>> a , b = b, a
```

Λεξικά

Το λεξικό είναι μια εξαιρετικά χρήσιμη ενσωματωμένη (built-in) δομή της Python. Φανταστείτε το σαν έναν τηλεφωνικό κατάλογο ο οποίος μας δίνει τη δυνατότητα να βρούμε πολύ γρήγορα τα στοιχεία κάποιου μόνο από το όνομά του. Προγραμματιστικά, φανταστείτε το σαν μια λίστα που έχει ως δείκτες αλφαριθμητικά και όχι ακέραιους αριθμούς. Για παράδειγμα, μπορούμε να γράψουμε version["python"] = 3.4. Όπως σε έναν πίνακα 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 μια 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:

```
>>> dictionary = { 'A':2, 'B':4, 'Ω':8, 'M':16 }
>>> list(dictionary.keys() )
['A', 'B', 'Ω', 'M']
>>> dictionary ['Ω']
8
>>> len(dictionary )
4
>>> del dictionary ['Ω']
>>> del dictionary ['A']
>>> print dictionary
{'B':4, 'M':16}
>>> dictionary ['Λ'] = 32
>>> print dictionary
{'B':4, 'Λ':32, 'M':16}
>>> 'Λ' in dictionary
True
```

Παραπάνω φαίνονται κάποιες λειτουργίες του λεξικού, όπως η διαγραφή (**del**) και ο έλεγχος ύπαρξης ενός στοιχείου με τον τελεστή **in**.

Θα πρέπει να σημειωθεί ότι κάθε κλειδί έχει μοναδική τιμή, ενώ με την εκχώρηση *Λεξικό[κλειδί]=τιμή* δημιουργείται το ζεύγος {κλειδί : τιμή}. Αν θέλουμε να αντιστοιχήσουμε σε ένα κλειδί περισσότερες από μια τιμές, τότε η τιμή του κλειδιού είναι μια λίστα από τις τιμές αυτές.

Αν θέλετε να μάθετε περισσότερα για τη γλώσσα Python μπορείτε να ανατρέξετε στα παρακάτω βιβλία τα οποία μπορείτε να κατεβάσετε δωρεάν από το διαδίκτυο.

Βιβλιογραφία

1. Αράπογλου Α., Βραχνός Ε., Κανίδης Ε., Μακρυγιάννης Π., Μπελεσιώτης Β., Τζήμας Δ. ***Αρχές Προγραμματισμού Υπολογιστών***, Β' τάξη ΕΠΑΛ, εκδόσεις Διόφαντος. http://ebooks.edu.gr/new/books-pdf.php?course=DSEPAL-B133
2. Αράπογλου Α., Βραχνός Ε., Κανίδης Ε., Λέκκα Δ., Μακρυγιάννης Π., Μπελεσιώτης Β., Παπαδάκης Σ., Τζήμας Δ. ***Προγραμματισμός Υπολογιστών***, Γ' τάξη ΕΠΑΛ, εκδόσεις Διόφαντος. http://ebooks.edu.gr/new/books-pdf.php?course=DSEPAL-C219
3. Downey A., ***Think Python, How to think like a computer scientist***. Green Tea Press.
4. Interactive Edition, how to think like a computer scientist, http://interactivepython.org/runestone/static/thinkcspy