Ζάννειο Πειραματικό Γυμνάσιο

Προγραμματισμός σε Python

Εισαγωγή στην Γλώσσα Προγραμματισμού Python

Ευριπίδης Βραχνός

http://evripides.mysch.gr/

2020 – 2021

Τύποι δεδομένων

Ένα πρόγραμμα συνήθως επεξεργάζεται δεδομένα τα οποία μπορεί να είναι αποθηκευμένα στην κύρια μνήμη του υπολογιστή, σε κάποιο αποθηκευτικό μέσο ή να εισάγονται από κάποια εξωτερική πηγή. Οι τύποι δεδομένων προσδιορίζουν τον τρόπο αναπαράστασης των δεδομένων εσωτερικά στον υπολογιστή, καθώς και το είδος της επεξεργασίας τους από αυτόν, δηλαδή τις πράξεις/λειτουργίες που ορίζονται για κάθε τύπο.

Οι **αριθμητικοί τύποι** στην Python είναι οι ακέραιοι (integer) (**int**) και οι αριθμοί κινητής υποδιαστολής (floating point) (**float**) που αναπαριστούν πραγματικούς αριθμούς. **Προσοχή** για την υποδιαστολή χρησιμοποιούμε την 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/αλφαριθμητικά** είναι μια ακολουθία από χαρακτήρες οι οποίοι μπορούν να είναι ψηφία, γράμματα ή σημεία στίξεως. 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, όμως με ότι ξεκινάμε, θα πρέπει πάλι να κλείνουμε.

Οι **λίστες** είναι ο σημαντικότερος σύνθετος τύπος (ή δομή δεδομένων) της Python. Η φιλοσοφία της γλώσσας βασίζεται στη δομή αυτή. Μια λίστα είναι ουσιαστικά μια ακολουθία από αντικείμενα οποιουδήποτε τύπου. Δεν είναι απαραίτητο σε μια λίστα να είναι όλα τα στοιχεία του ίδιου τύπου. Οι λίστες είναι δυναμικές δομές, δηλαδή μπορούμε να προσθέσουμε ή να αφαιρέσουμε στοιχεία αυξομειώνοντας το μέγεθος της λίστας.

Μπορούμε να ελέγξουμε τον τύπο δεδομένων μιας έκφρασης με χρήση της εντολής type () όπως φαίνεται παρακάτω:

```
>>> type(8128)
<class 'int'>
>>> type(3.14159)
<class 'float'>
>>> type(False)
<class 'bool'>
>>> type("Ευριπίδης")
<class 'str'>
>>> type( [ 8128, True, "Ευριπίδης" ] )
<class 'list'>
>>> type( ( "uniwa", 12243 ) )
<class 'tuple'>
```

Η λέξη class θα χρησιμοποιείται ως συνώνυμο του τύπου (type) σε αυτό το μάθημα, μια και στην Python τα πάντα είναι αντικείμενα.

Στον παρακάτω πίνακα συνοψίζονται οι βασικοί τύποι με τους οποίους θα ασχοληθούμε σε αυτό το μάθημα

Τύπος	Ονομασία	Παράδειγμα
Ακέραιοι	int	1, 0, –1, 496
Κινητής Υποδιαστολής	float	3.14159, 0.5, 3.0
Λογικές	bool	True, False
Αλφαριθμητικά	str	“uniwa”, “Πανεπιστημιούπολη 1”
Λίστες	list	[6, 28, 496], [“a”, “34w”, 45, True]
Πλειάδες	tuple	(“uniwa”, “Αιγάλεω”)
Απουσία τιμής	None	None

Αριθμητικές 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	/
Ακέραια Διαίρεση	//
Ύψωση σε δύναμη	**
Το υπόλοιπο της ακέραιας διαίρεσης	%

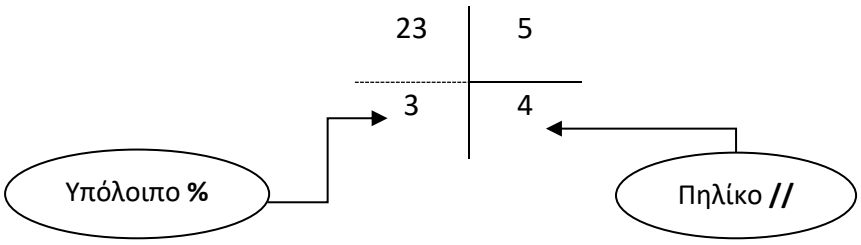
Σε κάθε έκφραση στην οποία υπάρχουν αριθμητικοί τελεστές ο υπολογισμός της ακολουθεί μια αυστηρά καθορισμένη ιεραρχία πράξεων, που είναι:

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 (200+1)*10 θα εκτελεστεί πρώτα η πρόσθεση μέσα στην παρένθεση και μετά, το αποτέλεσμα θα πολλαπλασιαστεί επί 10 που μας κάνει 2010, σε αντίθεση με την έκφραση 200+1*10 στην οποία, πρώτα θα γίνει ο πολλαπλασιασμός και μετά η πρόσθεση, άρα θα πάρουμε 210.

Αριθμητικές πράξεις	Αποτελέσματα	Παρατηρήσεις
>>> 8+10*2	28	
>>> 2**10 + 6	1030	
>>> 23 / 5	4.6	Διαίρεση με δεκαδικό μέρος
>>> 23 // 5	4	ακέραιο πηλίκο της Ευκλείδειας Διαίρεσης
>>> 23 % 5	3	υπόλοιπο ακέραιας (Ευκλείδειας) Διαίρεσης

Θα πρέπει να προσέξετε τους τελεστές της ακέραιας διαίρεσης // και %. η λειτουργία των οποίων φαίνεται στο παρακάτω επεξηγηματικό σχήμα



Δραστηριότητα

Να σημειώσετε δίπλα από κάθε έκφραση το αποτέλεσμα που αναμένετε και στη συνέχεια να το επαληθεύσετε στον διερμηνευτή της Python.

```
>>> 25 / 4          >>> 1 / 100
>>> 25 // 4         >>> 1 % 100
>>> 25 % 4          >>> 1 // 100
>>> 6 / 3           >>> 99 % 100
>>> 6 // 3          >>> 99 / 100
>>> 6 % 3           >>> 99 // 100
```

Λογικές εκφράσεις

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.

Λογικές εκφράσεις με συγκριτικούς τελεστές	Αποτελέσματα
>>> 496 == 496	True
>>> 12 != 13	True
>>> 100 <= 22	False
Σύνθετες Λογικές εκφράσεις	**Αποτελέσματα**
>>> (12<11) **and** (23>10)	False
>>> (12<11) **or** (23>10)	True
>>> **not**(56<=12)	True

Αλφαριθμητικές εκφράσεις

Ο τελεστής (συνάρτηση) str όπως είδαμε προηγουμένως μετατρέπει ένα αντικείμενο άλλου τύπου σε αλφαριθμητικό. Ο τελεστής της πρόσθεσης '+' όταν εφαρμόζεται σε αλφαριθμητικά εκτελεί συνένωση ενώ αυτός του πολλαπλασιασμού '*' παράγει τη συμβολοσειρά επαναλαμβανόμενη τόσες φορές όσες είναι ο αριθμός.

Αλφαριθμητικές εκφράσεις	Αποτελέσματα
>>> "Inter" + "stellar"	"Interstellar"
>>> "Δυτική " + "Αττική"	"Δυτική Αττική"
>>> 4 * "bla"	"blablablabla"
>>> 3 + 4 * "bla"	TypeError
>>> (3 + 4)* "bla"	"blablablablablablabla"
>>> str(3) + 4 * "bla"	"3blablablabla"
>>> 3 * "bla" * 2	"blablablablablabla"

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.

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

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.

Βασικές Ιδιότητες των μεταβλητών στην Python

Επειδή η Python χρησιμοποιεί διερμηνευτή χαρακτηρίζεται από μεγάλη ευελιξία. Έτσι στην Python οι μεταβλητές:

- Δεν χρειάζεται να δηλωθούν σε κανένα σημείο του προγράμματος
- Έχουν τύπο ο οποίος καθορίζεται όταν τους εκχωρείται μια τιμή
- Αν σε κάποιο σημείο του προγράμματος του εκχωρηθεί τιμή διαφορετικού τύπου από αυτόν που έχουν εκείνη τη στιγμή ο τύπος τους αλλάζει.

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.

Καλό είναι να δίνουμε στις μεταβλητές ονόματα σχετικά με το σκοπό τους και την ερμηνεία των δεδομένων που είναι καταχωρημένα σε αυτές.

Για να δώσουμε μια τιμή σε μια μεταβλητή, χρησιμοποιούμε τον τελεστή ίσον " = ", όπως για παράδειγμα *average_grade = 8.76*.

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.

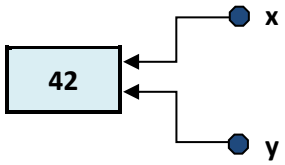
Η εντολή x = 42 σημαίνει ότι η μεταβλητή με όνομα x αντιστοιχεί σε μια θέση μνήμης του υπολογιστή και η ακέραια τιμή 42 εισάγεται στη θέση αυτή. Αν στη συνέχεια θέσουμε x = 250, στη θέση μνήμης που αναφέρεται η μεταβλητή a εισάγεται η νέα ακέραια τιμή 250 και η παλαιά τιμή 42 παύει πλέον να υπάρχει, 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).

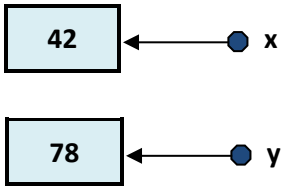
a=1 a=2 a=b=2

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.

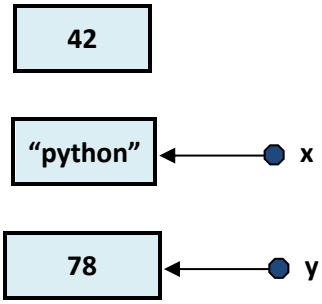
Όταν θέσουμε στη συνέχεια **y = x**, η μεταβλητή y αναφέρεται και αυτή στο 42. Δηλαδή και οι δυο μεταβλητές x, y έχουν συσχετιστεί με το 42. Στην πληροφορική αργκό λέμε ότι έχουν ένα δέσιμο (binding) με την τιμή 42.



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.



Αν στη συνέχεια εκχωρήσουμε στη μεταβλητή x το αλφαριθμητικό “python” τότε η μεταβλητή x θα δείχνει στη λέξη “python”. Σε αυτή την περίπτωση η μεταβλητή δείχνει σε μια τιμή διαφορετικού τύπου. Για αυτό λέμε ότι η Python έχει δυναμικό σύστημα τύπων.



Η Python παρακολουθεί όλες τις τιμές και τις διαγράφει όταν πάψουν να υπάρχουν μεταβλητές που να αναφέρονται σε αυτές, όπως είναι το 42 παραπάνω. Η διαδικασία αυτή ονομάζεται συλλογή σκουπιδιών (garbage collection).

Εκτύπωση με την εντολή εξόδου print

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).

```
>>> number = 28              # εκχωρεί την ακέραια τιμή 28 στο number
>>> print ( number )          # εμφανίζει το περιεχόμενο της x
28
>>> print ( “number” )        # εμφανίζει το όνομα της number
number
>>> number = number + 72      # προσθέτει στην τιμή της number το 72
>>> print ( number )
100
>>> number = number * 0.5          # μειώνει την τιμή της στο μισό
>>> print ( number )
50.0
```

Χρησιμοποιούμε την εντολή print για να εκτυπώσουμε τα αποτελέσματα του προγράμματός μας ή ένα μήνυμα στον χρήστη στην οθόνη. Η print στην Python 2

ήταν εντολή (statement) και συντασσόταν χωρίς παρενθέσεις ενώ στην Python 3 είναι πλέον ενσωματωμένη συνάρτηση της γλώσσας.

Για να εισάγουμε σχόλια στο πρόγραμμά μας για την επεξήγηση κάποιων δύσκολων σημείων 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.

Δραστηριότητα

Να εκτελέσετε τις παρακάτω εντολές στον διερμηνευτή και εξηγήστε τα αποτελέσματα.

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

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

Η εντολή x = x + 1

Στην εντολή x = x + 1 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:

$$X_{\text{μετά}} = X_{\text{πριν}} + 1$$

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

x += 1

Δραστηριότητα

Να εκτελέσετε τις παρακάτω εντολές στον διερμηνευτή και να εξηγήσετε τη λειτουργία τους. Εκτελούν και οι δυο ομάδες εντολών την ίδια λειτουργία; Μπορείτε να σκεφτείτε μια περίπτωση που δεν ισχύει αυτό;

```
>>> a = 6
>>> b = 100
>>> a = a + b
>>> b = a – b
>>> a = a – b
>>> print ( a, b )
```

```
>>> a = 6
>>> b = 100
>>> temp = a
>>> a =  b
>>> b = temp
>>> print ( a, b )
```

Η Εντολή Εισόδου Δεδομένων input( )

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 την input() .

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

Αν ο χρήστης μας δώσει μια αριθμητική τιμή θα πρέπει να την μετατρέψουμε πρώτα στον αντίστοιχο τύπο. Αυτό γίνεται με το όνομα του τύπου και παρενθέσεις όπως φαίνεται στο παρακάτω πρόγραμμα στο οποίο ο χρήστης εισάγει δυο αριθμούς έναν ακέραιο και έναν πραγματικό για να πάρει το γινόμενό τους.

```
>>> a = int ( input( “Ηλικία = ” ) )
>>> b = float ( input( “Ηλικία = ” ) )
>>> product = a * b                    # υπολογισμός γινομένου
>>> print ( product )
```

Δραστηριότητα

Να γράψετε ένα πρόγραμμα test.py το οποίο θα εκτελεί τα εξής βήματα:

1. Εισαγωγή μιας ακέραιας τιμής στη μεταβλητή x από το πληκτρολόγιο
2. Αύξηση του x κατά 2
3. Διπλασιασμός του x
4. Μείωση του x στο μισό.
5. Αύξηση του x κατά 75%.
6. Εκτύπωση της τετραγωνικής ρίζας του x
7. Εκτύπωση του x^x.

Μετά από την εκτέλεση κάθε εντολής να εμφανίζετε την τιμή της μεταβλητής print.

Ενσωματωμένες συναρτήσεις

Η Python παρέχει μια ποικιλία **ενσωματωμένων συναρτήσεων**, κάποιες από τις οποίες έχουμε ήδη χρησιμοποιήσει για τη μετατροπή τύπων.

- Η **συνάρτηση float( )** μετατρέπει μια τιμή σε δεκαδικό αριθμό.
- Η **συνάρτηση int( )** δέχεται οποιαδήποτε τιμή και τη μετατρέπει σε ακέραιο με αποκοπή των δεκαδικών ψηφίων αν υπάρχουν.
- Η **συνάρτηση str( )** μετατρέπει μια τιμή σε αλφαριθμητικό.
- Η **συνάρτηση list( )** μετατρέπει ένα αντικείμενο σε λίστα εφόσον ορίζεται αυτή η μετατροπή.

Αλλά και κάποιες χρήσιμες συναρτήσεις όπως οι :

- **abs()** επιστρέφει την απόλυτη τιμή ενός αριθμού.
- **pow(a,b)** επιστρέφει τη δύναμη του α υψωμένη στο β.
- **divmod(x,y)** επιστρέφει το πηλίκο και το υπόλοιπο της διαίρεσης x/y.
- **len**(L) επιστρέφει το μήκος της λίστας ή του αλφαριθμητικού L.

```
>>> float(10)
10.0
>>> int(5.999)
5
>>> abs(-45)
45
>>> divmod (10,3)
(3, 1)
>>> pow (2,3)
8
```

```
>>> str(0) + str(12) + str(35)
'01235'
>>> list('uniwa')
['u', 'n', 'i', 'w', 'a']
>>> len( 'uniwa' )
5
>>> len ( [1, 2, 3, 4, 5, 6] )
6
>>> pow (2,3)
8
```

Η γλώσσα Python διαθέτει μια βιβλιοθήκη μαθηματικών συναρτήσεων (math module), η οποία περιέχει τις περισσότερο γνωστές μαθηματικές συναρτήσεις. Προτού χρησιμοποιήσουμε μια βιβλιοθήκη, 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
>>> print ( math.e )
2.718281828459045
>>> print (math.pi )
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.

Δομή προγράμματος

Συνήθως ένα πρόγραμμα αποτελείται από τρία κύρια μέρη:

α) Εισαγωγή δεδομένων

β) Υπολογισμός αποτελεσμάτων

γ) Εκτύπωση αποτελεσμάτων

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

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

Ασκήσεις

Άσκηση 1

Το παρακάτω πρόγραμμα σχεδιάστηκε ώστε να διαβάζει έναν αριθμό και να υπολογίζει και να εμφανίζει το δεκαπλάσιό του. Να το εκτελέσετε και να εξηγήσετε το αποτέλεσμα. Τι αλλαγές πρέπει να κάνετε έτσι ώστε το πρόγραμμα να υπολογίζει το σωστό αποτέλεσμα;

```
number = input ("Δώσε έναν αριθμό: " )
result = 10 * number
print ("Το δεκαπλάσιό του είναι: ", result )
```

Άσκηση 2

Να γράψετε αλγόριθμο ο οποίος θα διαβάζει ένα ποσό σε ευρώ και θα το μετατρέπει σε δραχμές. Στη συνέχεια θα το εκτυπώνει. Δίνεται ότι 1€ = 340 δρχ.

Άσκηση 3

Θα γράψετε έναν αλγόριθμο που θα διαβάζει την ακτίνα ενός κύκλου και θα εμφανίζει στην οθόνη την περίμετρο και το εμβαδό του κύκλου.

Άσκηση 4

Θα γράψετε έναν αλγόριθμο που θα διαβάζει μια γωνία σε μοίρες και θα την μετατρέπει σε ακτίνια.

Άσκηση 5

Θα γράψετε έναν αλγόριθμο που θα διαβάζει τους προφορικούς και γραπτούς βαθμούς σας για κάθε μάθημα και θα υπολογίζει τον τελικό βαθμό σας.

Άσκηση 6

Να γράψετε αλγόριθμο ο οποίος θα διαβάζει το πλήθος των μαθητών ενός σχολείου, το πλήθος των μαθητών που προβιβάστηκαν και το πλήθος των μαθητών που αρίστευσαν και θα υπολογίζει και θα εμφανίζει τα αντίστοιχα ποσοστά επί τις εκατό.

Άσκηση 8

Να γράψετε αλγόριθμο ο οποίος θα διαβάζει τις συντεταγμένες δυο σημείων Α($\mathbf{x_1}$, $\mathbf{y_1}$), Β($\mathbf{x_2}$, $\mathbf{y_2}$), θα υπολογίζει την απόστασή τους στο επίπεδο με βάση τον παρακάτω τύπο

$$d(A,B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Άσκηση 10

Μπορείτε να γράψετε έναν αλγόριθμο ανταλλαγής των ακέραιων τιμών των μεταβλητών a, b

α) χωρίς να χρησιμοποιήσετε βοηθητική μεταβλητή

β) με τη χρήση βοηθητικής μεταβλητής

Άσκηση 11

Να γράψετε αλγόριθμο ο οποίος θα διαβάζει έναν διψήφιο ακέραιο αριθμό και θα εμφανίζει το άθροισμα των ψηφίων του.

Άσκηση 12

Να γράψετε αλγόριθμο ο οποίος θα διαβάζει έναν τρίψηφιο ακέραιο αριθμό και θα εμφανίζει το άθροισμα των ψηφίων του.

Άσκηση 13

Να γράψετε αλγόριθμο ο οποίος θα διαβάζει έναν τετραψήφιο ακέραιο αριθμό και θα εμφανίζει το άθροισμα των ψηφίων του.

Άσκηση 14

Να γράψετε αλγόριθμο ο οποίος θα διαβάζει έναν τετραψήφιο ακέραιο αριθμό και θα εμφανίζει τον ανάστροφό του. π.χ. ο ανάστροφος του 4876 είναι ο 6784. Οι αριθμοί αυτοί είναι γνωστοί και ως κατοπτρικοί. Στη συνέχεια να εμφανίζει και τον ανάστροφο + 1, π.χ. 6785.

Άσκηση 15

Μια πιτσαρία στην Πετρούπολη δίνει στις 3 πίτσες τη 1 δωρεάν. Να γράψετε έναν αλγόριθμο ο οποίος θα διαβάζει πόσες πίτσες παραγγείλαμε και θα υπολογίζει και θα εμφανίζει στην οθόνη τα χρήματα το συνολικό κόστος σε ευρώ. Δίνεται ότι μια πίτσα κοστίζει 6 ευρώ.

Δομή Ακολουθίας

Η πρώτη αλγοριθμική δομή που θα συναντήσουμε είναι η δομή της ακολουθίας η οποία λέει κάτι που φαίνεται ότι είναι αυτονόητο. Ότι σε ένα πρόγραμμα οι εντολές εκτελούνται με τη σειρά με την οποία δίνονται. Ωστόσο αυτή η αρχή μπορεί να παραβιαστεί αν χρησιμοποιηθούν εντολές άλματος σε προηγούμενη η επόμενη εντολή όπως ήταν παλιά η goto ή όπως είναι σήμερα η break και η continue για τις οποίες θα μιλήσουμε στη συνέχεια.

Παρακάτω δίνουμε παραδείγματα απλών προγραμμάτων

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.

```
euro = int ( 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 = int ( input ( "Δώσε ένα ποσό σε ευρώ : ") )
draxmes = euro * 340
print  ( "Τα  ", euro, " ευρώ είναι  ", draxmes, " δραχμές  " )
pitses = int ( input("Πίτσες για παραγγελία = ") )
price = int ( 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 μετά το τέλος της if.

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

```
# πρόγραμμα εμφάνισης της απόλυτης τιμής ενός ακεραίου αριθμού
a=int ( 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 εξής:

```
if <συνθήκη>:
    # εκτελούνται όταν η συνθήκη είναι True
else:
    # εκτελούνται όταν η συνθήκη είναι False
```

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 =  int ( input (“Δώσε τον βαθμό του πρώτου τετραμήνου: ”) )
gradeB =  int ( input (“Δώσε τον βαθμό του δεύτερου τετραμήνου: ”) )
final = ( gradeA + gradeB ) / 2
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 =  int ( input (“Δώσε έναν ακέραιο αριθμό: ”) )

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

print ( “Αριθμός ” )
```

```
if  number < 0 :
   print ( “Αρνητικός ” )
if  number >= 0 and number < 10 :
      print ( “Θετικός Μονοψήφιος ” )
if   number >= 10 and number < 100:
      print ( “Θετικός Διψήφιος ” )
if   number >= 100:
      print ( “Θετικός 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 = 2019
if  number >= 0 :
    print ( “Θετικός ή μηδέν” )
else:
    print ( “Αρνητικός ” )

print ( “Αριθμός ” )
```

```
number = 2019
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 = int ( input (“Δώσε έναν ακέραιο αριθμό: ”) )

if number < 0 :
   print ( “Αρνητικός ” )
elif number < 10 :
   print ( “Μονοψήφιος Θετικός ” )
elif number < 100 :
   print ( “Διψήφιος Θετικός” )
else:
   print ( “Πολυψήφιος Θετικός” )

print ( “Αριθμός ” )
```

```
if number < 0 :
   print ( “Αρνητικός ” )
if number >= 0 and number < 10 :
      print ( “Θετικός Μονοψήφιος ” )
if  number >= 10 and number < 100:
      print ( “Θετικός Διψήφιος ” )
if  number >= 100:
      print ( “Θετικός 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 έλεγχος είναι ψευδής (δεν ισχύει) προχωράμε στο επόμενο elif/else άρα δεν χρειάζεται να ελέγξουμε την ίδια συνθήκη ξανά. 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 ελέγξουμε ξανά.

Δομή επανάληψης while

Η ισχύς των υπολογιστών φαίνεται όταν θέλουμε να εκτελέσουμε επαναληπτικές διαδικασίες. 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 = int( input( “number = ” ) )
while  number != 0  :
    counter = counter + 1
    number = int( input(“number = ”) )
```

Γενική μορφή της while :

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

Χρησιμοποιώντας την εντολή while μπορούμε να βρούμε το άθροισμα όλων των αριθμών από το 1 έως και το 1000.

```
sum = 0
number = 1
while number <= 1000 :
    sum = sum + number
    number = number + 1
print( sum )
```

Η μεταβλητή number παίζει το ρόλο του **μετρητή** της επανάληψης, έτσι ώστε να γνωρίζουμε κάθε στιγμή ποια επανάληψη εκτελείται. Όταν ξεπεράσει την 1000στη επανάληψη τότε η συνθήκη γίνεται False και η επαναληπτική εντολή τερματίζεται. Αμέσως μετά η εκτέλεση συνεχίζεται με την εντολή που βρίσκεται αμέσως μετά την επανάληψη.

Η μεταβλητή sum είναι ένας αθροιστής αφού σε αυτή καταχωρείται το άθροισμα όλων των αριθμών. Η ιδέα είναι κάθε στιγμή να προσθέτουμε στο άθροισμα που έχουμε υπολογίσει μέχρι εκείνη τη στιγμή τον επόμενο αριθμό.

Ακολουθία βημάτων εκτέλεσης της εντολής while:

1. Αποτιμάται η Συνθήκη και
2. Αν είναι True
   εκτελούνται οι Εντολές
   Μετάβαση στο βήμα 1
3. Αν είναι False η επανάληψη τερματίζεται και εκτελείται η εντολή που είναι αμέσως μετά την επανάληψη

Η ώρα της γενίκευσης: *Υπολογισμός μέγιστης τιμής 3 αριθμών*

Μια πρώτη σκέψη για τον υπολογισμό του μεγαλύτερου από τρεις αριθμούς είναι ο παρακάτω αλγόριθμος :

```
a = int( input ("a = ") )
b = int( input ("b = ") )
c = int( 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
print( "Η μεγαλύτερη τιμή είναι : ", maximum )
```

Ο παραπάνω αλγόριθμος έχει ένα σημαντικό πρόβλημα. Δεν επεκτείνεται εύκολα για περισσότερους αριθμούς (scaling). Γενικά όταν σχεδιάζουμε έναν αλγόριθμο για την επίλυση ενός προβλήματος δεν θα πρέπει να σκεφτόμαστε μόνο τα δεδομένα του συγκεκριμένου προβλήματος αλλά και μελλοντικές επεκτάσεις του αλγορίθμου σε άλλα σύνολα δεδομένων. Για παράδειγμα οι αλλαγές που θα χρειαστούν στον παραπάνω αλγόριθμο για τον υπολογισμό της μέγιστης τιμής 5 αριθμών είναι αρκετές.

Ας φανταστούμε ότι οι αριθμοί εισάγονται ένας-ένας, δηλαδή δεν τους έχουμε όλους στην διάθεσή μας από την αρχή.

```
a = int ( input ("a = ") )
maximum = a                  # Ο πρώτος αριθμός είναι ο μεγαλύτερος μέχρι στιγμής

b = int ( input ("b = ") )
if b > maximum :             # Αν ο δεύτερος αριθμός είναι μεγαλύτερος από τον πρώτο
   maximum = b               # τότε θέτουμε αυτόν ίσο με maximum

c = int ( input ("c = ") )
if c > maximum :             # Αν ο τρίτος αριθμός είναι μεγαλύτερος από τον από τον
   maximum = c               # μεγαλύτερο των άλλων δυο τότε θέτουμε αυτόν ίσο με maximum
```

Βασική ιδέα του νέου αλγορίθμου

Δεν χρειάζεται να συγκρίνουμε κάθε αριθμό με όλους τους προηγούμενους αλλά μόνο με τον μεγαλύτερο όλων αυτών. Αν ο αριθμός δεν είναι μεγαλύτερος δεν μας ενδιαφέρει. Αν είναι όμως τότε θα πρέπει αυτός να οριστεί ως ο νέος μέγιστος.

Παρατηρούμε λοιπόν ότι το μοτίβο των εντολών που επαναλαμβάνονται είναι :

```
next =  int ( input ("b = ") )      # Διάβασε τον επόμενο αριθμό από τον χρήστη

if  next > maximum :                # Αν ο αριθμός αυτός είναι μεγαλύτερος από τον μέχρι
                                    # στιγμής μέγιστο
    maximum = next
                                    Τότε θέσε αυτόν ως μέγιστο
```

Επίσης σκεφτόμαστε ότι οι παραπάνω δυο εντολές θα μπουν μέσα σε μια επαναληπτική εντολή και επίσης η αρχική τιμή του maximum θα είναι ο πρώτος αριθμός που θα δοθεί. Τέλος θα χρειαστούμε και έναν μετρητή για να ξέρουμε πόσους αριθμούς διαβάζουμε και πότε πρέπει να σταματήσουμε. Έτσι ο παρακάτω αλγόριθμος δέχεται από τον χρήστη 100 αριθμούς και υπολογίζει και εμφανίζει τη μεγαλύτερη τιμή που δόθηκε.

```
next =  int ( input ("next = ") )
maximum  = next
count = 1                           # έχει ήδη διαβάσει έναν αριθμό
while count <= 100 :
    next =  int ( input ("next = ") )
    if  next > maximum :
        maximum = next
    count = count + 1               # γράφεται και έτσι  :  count+=1
print( maximum )
```

Παρατηρήστε ότι τώρα χρησιμοποιούμε μια μόνο μεταβλητή τη next ενώ προηγουμένως αποθηκεύονταν όλοι οι αριθμοί σε διαφορετικές μεταβλητές (a, b, c, …).

Ένας τέτοιος αλγόριθμος όπως ο παραπάνω λέγεται *αυξητικός (incremental)* διότι δέχεται δεδομένα συνεχώς και ενημερώνει το αποτέλεσμα, χωρίς να χρειάζεται να κάνει υπολογισμούς από την αρχή. Για παράδειγμα ο συγκεκριμένος αλγόριθμος δεν χρειάζεται να συγκρίνει τον επόμενο αριθμό με όλους τους προηγούμενους αλλά μόνο με τη μέγιστη τιμή. Σε περίπτωση που θέλαμε να ξέρουμε τη σειρά με την οποία δόθηκε ο μέγιστος αριθμός (εφόσον είναι μοναδικός) προσθέτουμε μια ακόμα μεταβλητή στην οποία καταχωρείται η θέση.

```
next =  int ( input ("next = ") )
maximum  = next
position = 1
count = 1                           # έχει ήδη διαβάσει έναν αριθμό
while count <= 100 :
    next =  int ( input ("next = ") )
    count = count + 1               # γράφεται και έτσι  :  count+=1
    if  next > maximum :
        maximum = next
        position = count
print( "maximum = ", maximum, "  position = ", position )
```

Λίστες

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

```
>>> teams = [‘ΑΕΚ’, ‘ΠΑΟ’, ‘ΟΣΦΠ’, ‘ΠΑΟΚ’ ]
>>> numbers = [6, 28, 496, 8128]
```

Μπορεί όμως να περιέχει και αντικείμενα διάφορων τύπων:

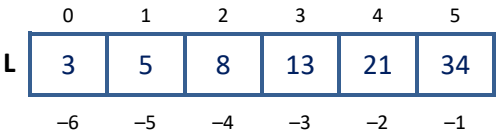
```
>>> values = [ ‘zero’, 13, 21, True ]
```

Για κάθε αντικείμενο που εισάγεται στη λίστα δίνεται ένας αύξων αριθμός που παριστάνει τη θέση του και 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
```

Συνάρτηση: **len** Χρήση: len(*<sequence>*) Επιστρέφει *το μέγεθος της δομής sequence, δηλαδή το πλήθος των στοιχείων της. Η δομή αυτή μπορεί να είναι λίστα, αλφαριθμητικό η πλειάδα.*
Τελεστής **in** : Χρήση: *<object>* **in** *<sequence>* Επιστρέφει *την τιμή* **True** *αν το αντικείμενο object βρίσκεται στο sequence, σε διαφορετική περίπτωση επιστρέφει* **False**.

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 παρακάτω:



Η αρίθμηση των στοιχείων, όπως στις συμβολοσειρές, έτσι και στις λίστες, ξεκινάει από το 0. Άρα το 1ο στοιχείο της λίστας είναι το L[0], το οποίο είναι ίσο με το 3, το 2ο το L[1] και τελευταίο το L[5]. Η Python παρέχει και αρνητική δεικτοδότηση για να διευκολύνει την αναφορά στα τελευταία στοιχεία της λίστας.

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

```
>>> print( L[ –1 ] )
34
>>> print( L[ –3 ] )
13
```

Η δομή επανάληψης for … in για τη 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:

```
>>> perfect = [6, 28, 496]

>>> for number in perfect :
        print( number )
6
28
496
```

```
>>> fibonacci = [1, 1, 2, 3, 5, 8, 13, 21, 34]

>>> for number in fibonacci :
        print( number, end = "  " )

1  1  2  3  5  8  13  21  34
```

Θα μπορούσαμε να μεταφράσουμε τη λειτουργία της εντολής επανάληψης ως εξής:

```
for number in List :
    print( number )
```

```
Για κάθε αριθμό της Λίστας:
    Εκτύπωσε τον αριθμό αυτό
```

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                     # το sum είναι ο αθροιστής

for number in L :           # Κάθε στοιχείο number της λίστας L
    sum = sum + number      # Το προσθέτω στο sum

average = sum / len( L )    # Μέσος όρος = άθροισμα / πλήθος στοιχείων
print( average )
```

Ο τελεστής list

Ο συναρτησιακός τελεστής **list** δέχεται ένα αντικείμενο και το μετατρέπει αν αυτό είναι δυνατόν σε λίστα, όπως φαίνεται στα παρακάτω παραδείγματα:

```
>>> list( "uniwa" )          >>> list( )          >>> list(False )
[ 'u', 'n', 'i', 'w', 'a' ]  [ ]                  TypeError: 'bool' Object is not iterable
>>> list( "  " )             >>> list( "" )       >>> list( str( "False" ) )
[' ']                        [ ]                  [ 'F', 'a', 'l', 's', 'e' ]
```

Η συνάρτηση range

Η συνάρτηση **range** δημιουργεί και επιστρέφει ένα αντικείμενο το οποίο περιγράφει μια λίστα από αριθμούς αν δώσουμε την αρχική τιμή, την τελική τιμή και το βήμα.

```
>>> range( 4 )          >>> range( 10, 30, 5 )     >>> range( 0 )
range(0, 4)             range(10, 30, 5)          range(0, 0)
```

Για να πάρουμε όμως τη λίστα με τους αριθμούς θα πρέπει να εφαρμόσουμε τον τελεστή **list**.

```
>>> list( range( 4 ) )          >>> list( range( 10, 30, 5 ) )     >>> list( range( 1, 1, 100 ) )
[0, 1, 2, 3]                    [10, 15, 20, 25]                   [ ]
>>> list( range( 0, 4 ) )       >>> list( range( 30, 10, –5 ) )    >>> list( range( 1, 5, –1 ) )
[0, 1, 2, 3]                    [30, 25, 20, 15]                   [ ]
>>> list( range( 0, 4, 1 ) )    >>> list( range( 1, 2, 100 ) )     >>> list( range( 0 ) )
[0, 1, 2, 3]                    [1]                                [ ]
```

> Η συνάρτηση **range**(A, M, B) επιστρέφει μια λίστα αριθμών ξεκινώντας από 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
```

Επίσης 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
```

Η δομή επανάληψης for

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 "κρύβονται" η αρχικοποίηση και η μεταβολή της μεταβλητής i.

Άρα αν θέλουμε να εκτελεστεί μπλοκ εντολών N φορές μπορούμε να χρησιμοποιήσουμε ένα από τα παρακάτω τμήματα κώδικα:

```
for index in range(N) :
    <Εντολές>
```

```
index = 0
while index < N :
    <Εντολές>
    index = index + 1
```

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

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

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

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

d = int( input ("d = ") )
if d > maximum :
    maximum = d
```

Χρήση μιας μόνο μεταβλητής

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

a = int( input ("a = ") )
if a > maximum :
    maximum = a

a = int( input ("a = ") )
if a > maximum :
    maximum = a

a = int( input ("a = ") )
if a > maximum :
    maximum = a
```

Δομή Επανάληψης

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

for i in range(3):
    a = int( input ("a = ") )
    if a > maximum :
        maximum = a
```

Όταν θέλουμε να εκτελέσουμε ένα μπλοκ εντολών έναν προκαθορισμένο αριθμό φορών μπορούμε να χρησιμοποιήσουμε είτε την εντολή for είτε την while. Το αποτέλεσμα θα είναι το ίδιο όπως φαίνεται παρακάτω:

```
for index in range(A, M, B) :
    <Εντολές>
```

```
index = A
while index < M :
    <Εντολές>
    index = index + B
```

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.

Οι συναρτήσεις είναι επαναχρησιμοποιήσιμα τμήματα κώδικα. Μας επιτρέπουν να δίνουμε ένα όνομα σε ένα σύνολο εντολών και να το εκτελούμε καλώντας το όνομά του, 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 (:). Η λίστα αυτή των μεταβλητών είναι γνωστή ως *λίστα παραμέτρων*. Οι μεταβλητές που χρησιμοποιούνται στον ορισμό της συνάρτησης λέγονται *παράμετροι* ενώ αυτές που χρησιμοποιούνται κατά την κλήση της *ορίσματα*.

Μια συνάρτηση επιστρέφει ένα αποτέλεσμα με την εντολή return. Η τιμή που επιστρέφεται αντικαθιστά τη συνάρτηση στην έκφραση στην οποία βρίσκεται. Για παράδειγμα η κλήση square(3), αντικαθιστά το 9 ως αποτέλεσμα της συνάρτησης. Αν μια συνάρτηση δεν επιστρέφει κάποιο αποτέλεσμα τότε η τιμή επιστροφής είναι η τιμή *None*.

```
# 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; Μπορούμε να χρησιμοποιήσουμε τη συνάρτηση printPython3 που έχουμε ήδη ορίσει, όπως φαίνεται παραπάνω στην τρίτη στήλη κώδικα. Οι παραπάνω συναρτήσεις εμφανίζουν κάποια μηνύματα στην οθόνη. Τι επιστρέφουν όμως ως αποτέλεσμα στο σημείο που έγινε η κλήση τους;

```
>>> printPython3( )
Python
Python
Python
```

```
>>> result = printPython3( )
Python
Python
Python
```

```
>>> result
>>>            # κενό, δεν έχει τιμή
>>> print ( result )
None
```

Η συνάρτηση printPython3 δεν επιστρέφει τιμή, οπότε η μεταβλητή result δεν έχει πάρει κάποια τιμή. Για αυτό όταν γράφουμε result δεν μας επιστρέφει τίποτα. Μόνο αν επιμείνουμε με την εντολή print να δούμε την τιμή της θα μας επιστρέψει None.

Γενίκευση μέσω της λίστας παραμέτρων

Δίνονται οι παρακάτω συναρτήσεις σε Python. Η συνάρτηση που δίνεται δεξιά είναι η γενίκευση των τριών, αφού ανάλογα με την τιμή της παραμέτρου 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' )
```

```
>>> printPython(3)
>>> printPython(12)
>>> printPython(100)
```

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

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 δεν 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 modulus (a, b, c):
    from math import sqrt
    return sqrt( a*a + b*b + c*c )

>>> modulus( 1, 2, 2 )
3.0
>>> modulus( 1, 1, 0 )
1.4142135623730951
>>> modulus( 1, 1, 0 ) * modulus( 1, 1, 0 )
2.00000000000004                    # κάτι δεν πάει καλά? ☺
```

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*.

Η εντολή break

Η εντολή break χρησιμοποιείται για τη διακοπή της εκτέλεσης μιας εντολής επανάληψης. Οι παρακάτω αλγόριθμοι διαβάζουν το πολύ 100 αριθμούς και σταματούν όταν το άθροισμά τους ξεπεράσει το 1000.

```
mysum = 0
for i in range(100):
    number = int( input( ) )
    mysum = mysum + number
    if mysum>1000:
        break
print(" sum = ", mysum)
```

```
mysum = 0
i = 0
while i<100 and mysum<=1000:
    number = int( input( ) )
    mysum = mysum + number
    i = i + 1
print(" sum = ", mysum)
```

Σε κάποιες περιπτώσεις η εντολή break μπορεί να φανεί πολύ χρήσιμη όταν η συνθήκη διακοπής του βρόχου είναι σύνθετη ή πολύπλοκη, αλλά καλό είναι να χρησιμοποιείται με φειδώ γιατί μπορεί να γίνει πηγή λαθών στα προγράμματα που αναπτύσσουμε.

Παραγωγή τυχαίων αριθμών

Σε διάφορες εφαρμογές πχ. Παιχνίδια ο υπολογιστής χρειάζεται να παράξει έναν ή περισσότερους αριθμούς με τυχαίο τρόπο, αλλιώς θα γίνει προβλέψιμος. Για την παραγωγή τυχαίων αριθμών μπορούμε να χρησιμοποιήσουμε τη βιβλιοθήκη random. Στις επόμενες εφαρμογές θα χρησιμοποιήσουμε τη συνάρτηση randint της βιβλιοθήκης random.

> Η συνάρτηση ***random.randint***(a, b) επιστρέφει έναν ακέραιο αριθμό **από a έως και b** με τυχαίο τρόπο.

περισσότερους αριθμούς με τυχαίο τρόπο, αλλιώς θα γίνει προβλέψιμος. Για την παραγωγή τυχα

```
>>>
>>> from random import randint
>>> index = 1
>>> while index<=100:
        print( randint(0,3), end = " ")
        index += 1


1 0 1 3 2 3 0 3 1 3 2 2 2 1 2 3 1 2 0 1 1 0 3 1 3
 1 0 1 2 3 2 2 1 0 1 0 1 2 3 3 2 0 2 3 0 2 2 1 2
2 3 2 3 1 3 2 1 3 2 3 1 2 0 2 2 1 3 0 3 3 0 0 2 3
 1 2 0 2 1 0 0 0 1 0 1 0 0 0 1 3 2 3 3 1 3 2 2 2
3 3
>>>
```

Εφαρμογές

Στη συνέχεια δίνουμε κάποιες εφαρμογές στις οποίες συνδυάζονται όλες οι αλγοριθμικές δομές (ακολουθία, επιλογή, επανάληψη) για την επίλυση διάφορων υπολογιστικών προβλημάτων.

Επεξεργασία αριθμών

Η παρακάτω συνάρτηση διαβάζει από τον χρήστη αριθμούς μέχρι να δοθεί 0 και υπολογίζει τον μέσο όρο των θετικών αριθμών.

```
def countPositives():
    positives = 0
    sum_positives = 0
    number = int( input( "number = " ) )
    while number != 0 :
        if number>0:
            positives = positives + 1
            sum_positives = sum_positives + number
        number = int( input( "number = " ) )

    average = sum_positives / positives
    print("average = ", average)
```

Υπολογισμός ακέραιων ριζών εξίσωσης

Η παρακάτω συνάρτηση υπολογίζει τις ακέραιες λύσεις οποιασδήποτε εξίσωσης μπορεί να περιγραφεί ως λάμδα έκφραση (**λ**-expression).

```
def solve(equation) :
    for x in range(-10000, 10000):
        if equation(x)==0:
            print(x)
```

```
>>> solve(lambda x: x*x-5*x+6)
2
3
>>> solve(lambda x: x**5-3*x**4-5*x**3+15*x**2+4*x-12)
-2
-1
1
2
3
```

Δοκιμάστε να εκτελέσετε την παραπάνω συνάρτηση για την επίλυση μιας δική σας εξίσωσης

Τέλειοι Αριθμοί

Τέλειοι ονομάζονται οι αριθμοί που είναι ίσοι με το άθροισμα των γνήσιων διαιρετών τους.

```
def isPerfect(number):
    divisor_sum = 0
    for divisor in range(1,number):
        if number%divisor == 0 :
            divisor_sum = divisor_sum + divisor

    return  divisor_sum == number

def perfect_gen(N):
    for number in range(2, N+1):
        if isPerfect(number):
            print(number)
```

Πρώτοι αριθμοί

Πρώτοι ονομάζονται οι αριθμοί οι οποίοι δεν έχουν άλλους διαιρέτες εκτός από το 1 και τον εαυτό τους.

Διεξοδικός έλεγχος όλων των πιθανών διαιρετών

```
from math import sqrt

def isPrime(n):
    root = int(sqrt(n))+1
    for j in range(2,root):
        if (n%j==0):
            return False
    return True


def allPrimes(n):
    primes = 0
    for i in range(2,n):
        if isPrime(i):
            primes = primes + 1
    return primes
```

Ελέγχει μόνο μέχρι τη ρίζα του αριθμού. (Γιατί;)

```
from math import sqrt

def isPrime(n):
    root = int(sqrt(n))+1
    for j in range(2,root):
        if (n%j==0):
            return False
    return True


def allPrimes(n):
    primes = 0
    for i in range(2,n):
        if isPrime(i):
            primes = primes + 1
    return primes
```

Παρατηρήστε ότι στην γρήγορη έκδοση της συνάρτηση isPrime όπου ο έλεγχος γίνεται μέχρι τη ρίζα, δεν γίνεται καμία αλλαγή στη συνάρτηση allPrimes. Αυτό είναι ένα παράδειγμα καλής πρακτικής τμηματικού προγραμματισμού και ανεξαρτησίας των υποπρογραμμάτων

Υπολογισμός Τέλειων Τετραγώνων μόνο με προσθέσεις

Η παρακάτω συνάρτηση υπολογίζει τα τέλεια τετράγωνα μέχρι κάποιο όριο μόνο με προσθέσεις. Μπορείτε να εξηγήσετε γιατί δουλεύει; Να διατυπώσετε την αλγεβρική ιδιότητα στην οποία στηρίζεται η ορθότητα του αλγορίθμου.

```
def squares(N):
    square = 1
    step = 3
    for i in range(N):
        print(square, end = "  ")
        square = square + step
        step = step + 2
```

Άθροισμα των ψηφίων ενός αριθμού

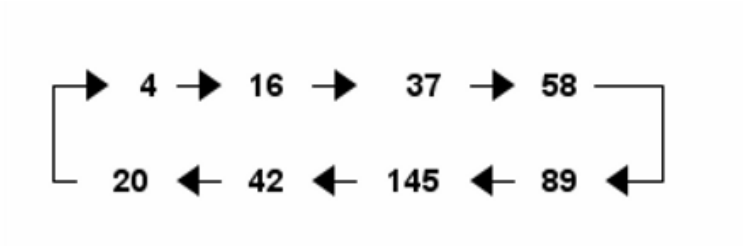
Η παρακάτω συνάρτηση υπολογίζει το άθροισμα των ψηφίων ενός αριθμού

```
def sum_digits(N):
    s = 0
    while N>0:
        s = s + N%10
        N = N//10
    return s
```

Άσκηση για Λύση

Ευτυχισμένοι και Δυστυχισμένοι αριθμοί

Παίρνουμε οποιονδήποτε θετικό ακέραιο αριθμό, και υπολογίζουμε το άθροισμα των τετραγώνων των ψηφίων του. Στον αριθμό που θα προκύψει εφαρμόζουμε την ίδια διαδικασία μέχρι οι πράξεις να δείξουν τον αριθμό. Οι αριθμοί που μετά από αυτήν τη διαδικασία τελειώνουν σε 1, ονομάζονται ***ευτυχείς αριθμοί*** (happy numbers), ενώ εκείνοι που τελειώνουν σε κάποιον άλλον, ονομάζονται ***δυστυχείς αριθμοί*** (sad numbers). Οι δυστυχείς αριθμοί καταλήγουν πάντα στον αριθμό 4, ενώ ο ίδιος ο 4 οδηγεί σε βρόχο επαναλαμβανόμενων αριθμών που καταλήγει πάλι στο 4.



Ασκήσεις

Άσκηση 1
Να γραφεί αλγόριθμος ο οποίος θα διαβάζει επαναληπτικά αριθμούς μέχρι το άθροισμα τους να γίνει μεγαλύτερο ή ίσο του 100. Στο τέλος να εμφανίζει το άθροισμα, το πλήθος των αριθμών που διαβάστηκαν και το ποσοστό αυτών που ήταν μεγαλύτεροι από 10.

Άσκηση 2
Να σχεδιάσετε αλγόριθμο ο οποίος να διαβάζει αριθμούς μέχρι να δοθεί το 0 και στη συνέχεια να εμφανίζει τη λέξη ΑΥΞΟΥΣΑ αν οι αριθμοί ήταν όλοι σε αύξουσα σειρά.

Άσκηση 3
Να γράψετε αλγόριθμο ο οποίος θα διαβάζει έναν αριθμό στο διάστημα [1,20]. Για οποιαδήποτε άλλη τιμή θα εμφανίζει μήνυμα λάθους και θα ζητάει από τον χρήστη να δώσει ξανά τον αριθμό. Στο τέλος να εμφανίζει πόσες φορές δόθηκε λάθος αριθμός.

Άσκηση 4
Να γράψετε αλγόριθμο ο οποίος θα διαβάζει επαναληπτικά αριθμούς με αποδεκτές τιμές στο διάστημα [1,20]. Για οποιαδήποτε άλλη τιμή θα εμφανίζει μήνυμα λάθους και θα ζητάει από τον χρήστη να δώσει ξανά τον αριθμό. Ο αλγόριθμος θα τερματίζει όταν ο χρήστης κάνει 4 συνεχόμενες φορές λάθος κατά την εισαγωγή. Στο τέλος να εμφανίζει πόσες φορές έκανε συνολικά λάθος ο χρήστης, όπως και πόσοι αριθμοί δόθηκαν σωστοί με την πρώτη φορά.

Άσκηση 5
Θα γράψετε έναν αλγόριθμο ο οποίος θα εμφανίζει στην οθόνη όλους τους εξαψήφιους κατοπτρικούς αριθμούς, π.χ. 123321, 657756, 112211.

Άσκηση 6
Θα γράψετε έναν αλγόριθμο που θα διαβάζει δυο ακέραιους αριθμούς και θα υπολογίζει το γινόμενό τους χωρίς να χρησιμοποιεί την πράξη του πολλαπλασιασμού.

Άσκηση 7
Θα γράψετε έναν αλγόριθμο που θα διαβάζει έναν αριθμό και θα ελέγχει αν αυτός διαιρείται με το 3 ή το 9 χωρίς να χρησιμοποιεί την πράξη της διαίρεσης.
Hint: Ένας αριθμός διαιρείται με το 3 ή το 9 αν και μόνον αν το άθροισμα των ψηφίων του διαιρείται με το 3 ή το 9 αντίστοιχα.
Επιτρέπεται μόνο η πράξη της διαίρεσης με το 10 για να βρείτε τα ψηφία του αριθμού.

Άσκηση 8
Θα γράψετε έναν αλγόριθμο ο οποίος θα εμφανίζει όλους τους αριθμούς που είναι ίσοι με το άθροισμα των ψηφίων τους υψωμένο κάθε ένα στο πλήθος των ψηφίων του αριθμού. Οι αριθμοί αυτοί λέγονται αριθμοί Armstrong.
π.χ. $1634 = 14 + 64 + 34 + 44$

Άσκηση 9

Ένα από τα άλυτα προβλήματα της θεωρίας αριθμών που έχουν εφαρμογή στην κρυπτογραφία είναι η ανάλυση ενός ακέραιου αριθμού σε πρώτους παράγοντες.
Να σχεδιάσετε αλγόριθμο ο οποίος να διαβάζει έναν ακέραιο αριθμό και να τον αναλύει σε γινόμενο πρώτων παραγόντων με όσο πιο αποδοτικό τρόπο μπορείτε. Για παράδειγμα η ανάλυση του αριθμού 300 είναι $\mathbf{300 = 3 \cdot 2 \cdot 2 \cdot 5 \cdot 5}$

Θέμα 10

Ο Ινδός μαθηματικός και αστρονόμος Madhava υπολόγισε την τιμή του π σύμφωνα με τη σχέση:

$$\pi = \sqrt{12}\left(1 - \frac{1}{3 \cdot 3} + \frac{1}{5 \cdot 3^2} - \frac{1}{7 \cdot 3^3} + \frac{1}{9 \cdot 3^4} - \frac{1}{11 \cdot 3^5} + \ldots\right)$$

Να σχεδιαστεί αλγόριθμος που θα διαβάζει σαν είσοδο **ένα θετικό αριθμό κ** μικρότερο του 0.1 **πραγματοποιώντας** τον κατάλληλο έλεγχο και θα υπολογίζει την τιμή του π σύμφωνα με την παραπάνω σχέση. Ο υπολογισμός θα σταματά όταν **η απόλυτη τιμή κάποιου όρου** γίνει **μικρότερη από κ**.
Ο αλγόριθμος θα εκτυπώνει την τιμή του π που υπολόγισε, καθώς και το πλήθος των όρων του αθροίσματος που χρειάστηκαν μέχρι να ικανοποιηθεί η συνθήκη διακοπής.

Άσκηση 11

Να γράψετε αλγόριθμο ο οποίος θα διαβάζει ακέραιους αριθμούς και θα σταματάει όταν δοθεί ο ίδιος αριθμός 4 συνεχόμενες φορές.