ΠΛΗΡΟΦΟΡΙΚΗ Ι (Python)
Εντολές ελέγχου και επανάληψης

Λογικές εκφράσεις

Οι λογικές ή Boolean εκφράσεις σχηματίζονται με τη χρήση σχεσιακών τελεστών που προσδιορίζουν την ύπαρξη ή μη κάποιας συγκεκριμένης σχέσης μεταξύ δύο τιμών.

Οι ***σχεσιακοί τελεστές*** στην Python:

Μαθηματικά	*Python*
=	==
<	<
≤	<=
>	>
≥	>=
≠	!=

Το αποτέλεσμα μιας σύγκρισης είναι τύπου `bool`. Οι μεταβλητές τύπου `bool` λέγονται *Λογικές Μεταβλητές*. Οι τιμές των λογικών μεταβλητών είναι `True` ή `False`. Η τιμή 0 θεωρείται `False`, ενώ οποιαδήποτε τιμή διαφορετική του μηδενός θεωρείται `True`.

π.χ.,
```
>>> a=5
>>> b=5
>>> c = a==b
>>> print(c)
True
>>> d = a>b
>>> print(d)
False
```

Οι λογικές εκφράσεις συνδέονται μεταξύ τους με τους ***λογικούς τελεστές***, σχηματίζοντας πιο σύνθετες εκφράσεις:

- Σύζευξη (AND): `and`
- Διάζευξη (OR): `or`
- Άρνηση (NOT): `not()`

Πίνακας αληθείας:

p	q	p and q	p or q	not(p)
True	True	True	True	False
True	False	False	True	
False	True	False	True	True
False	False	False	False	

Short-circuit: Οι τελεστές `and` και `or` λειτουργούν με τέτοιον τρόπο ώστε εάν μπορεί να βγει συμπέρασμα ως προς το αποτέλεσμα της σύνθετης λογικής έκφρασης από τον υπολογισμό της πρώτης λογικής έκφρασης, δεν υπολογίζεται η δεύτερη. Στην περίπτωση του `and`, αυτό ισχύει αν η πρώτη λογική έκφραση είναι ψευδής (οπότε το αποτέλεσμα θα είναι σίγουρα ψευδές, ό,τι και αν ισχύει για τη δεύτερη), ενώ στην περίπτωση του `or`, αυτό ισχύει αν η πρώτη λογική έκφραση είναι αληθής (οπότε το αποτέλεσμα θα είναι σίγουρα αληθές, ό,τι και αν ισχύει για τη δεύτερη).

Π.χ.,
```
>>> a = 2
>>> b = 4
>>> c = 6
>>> c >= 6 and a > 3
False
>>> a >= -1 or a <= b
True
>>> not(a>2)
True
```

- Παράδειγμα συνηθισμένου σφάλματος: Το x δεν ισούται με 2 ή 3:

  Λάθος: `x!=2 or x!=3` # → Είναι πάντα `True`!
  Σωστό: `x!=2 and x!=3` # ή: `not(x==2 or x==3)`

- Ισχύουν οι ιδιότητες:
  - Το: `not(p or q)` ταυτίζεται με το: `not(p) and not(q)`
  - Το: `not(p and q)` ταυτίζεται με το: `not(p) or not(q)`

- Έλεγχος διαστημάτων
  - Το: $x \in$ [0, 1) , δηλαδή το: $0 \le x < 1$, γράφεται: `0 <= x and x < 1`
  - Το: $x \notin$ (0, 1), γράφεται: `x < 0 or x > 1`

Σημείωση: Στην Python, το $0 \le x < 1$ μπορεί να γραφεί και ως: `0 <= x < 1` , δηλαδή μπορεί να γίνει διπλή σύγκριση στην ίδια έκφραση. Στην περίπτωση που στη θέση του `x` υπάρχει κάποια έκφραση, αυτή υπολογίζεται *μία φορά* (σε αντίθεση με την περίπτωση της σύνθετης μορφής της λογικής έκφρασης με τη χρήση του τελεστή `and`).

Παράδειγμα:

- Για δεδομένο έτος (`year`) να δοθεί τιμή στη `bool` μεταβλητή `is_leap_year` ανάλογα με το αν το έτος είναι δίσεκτο (`True`) ή όχι (`False`):

Λύση:
Πότε ένα έτος είναι δίσεκτο;
Δίσεκτα είναι τα έτη που διαιρούνται με το 4, *εκτός* από τα έτη των αιώνων (π.χ., 1700, 1800, 1900, κτλ.). Κατ' εξαίρεση, *είναι* δίσεκτα τα έτη των αιώνων που διαιρούνται με το 400 (π.χ., 1600, 2000, 2400, κτλ.).

Άρα, ένα έτος είναι δίσεκτο:

α) αν διαιρείται με το 4 αλλά όχι με το 100 **ή** β) αν διαιρείται με το 400

Οπότε:
```
year = int(input('Δώσε έτος: '))
is_leap_year = ((year%4==0) and (year%100!=0)) or (year%400==0)
print(is_leap_year)
# Βελτίωση: Καλύτερα πρώτα η πιο απλή έκφραση (λόγω short-circuit):
# is_leap_year = (year%400==0) or ((year%4==0) and (year%100!=0))
```

ΕΛΕΓΧΟΣ ΡΟΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

- Ελεγκτές συνθηκών ή περιπτώσεων (δομές ελέγχου):

  `if / if-else / if-elif-else`

- Επαναληπτικές διαδικασίες (δομές επανάληψης):

  `for`
  `while  (με έλεγχο συνθήκης)`

Α. Δομές ελέγχου – Η εντολή `if`

Η εντολή `if` χρησιμοποιείται για τη δημιουργία δομών απόφασης που επιτρέπουν σε ένα πρόγραμμα να έχει πολλαπλές διαδρομές εκτέλεσης.

Ουσιαστικά, επιτρέπει σε μία ή περισσότερες εντολές να εκτελεσθούν μόνο εάν κάποια λογική συνθήκη είναι αληθής.

i) Δομή απόφασης μίας εναλλακτικής:

```
if συνθήκη:
    εντολή 1
    εντολή 2
    ...
```

ii) Δομή απόφασης διπλής εναλλακτικής:

```
if συνθήκη:
    εντολή 1
    εντολή 2
    ...
else:
    εντολή 3
    εντολή 4
    ...
```

iii) Δομή απόφασης πολλαπλών συνθηκών

```
if συνθήκη_1:
    εντολή 1
    εντολή 2
    ...
elif συνθήκη_2:
   εντολή 2
   εντολή 4
   ...
...
else:
    εντολή 5
    εντολή 6
    ...
```

iv) Ένθετες (nested) δομές απόφασης:

```
if συνθήκη_1:
    εντολή 1
    if συνθήκη_2:
        εντολή 2
    else:
        εντολή 3
    ...
else:
    εντολή 4
    εντολή 5
    ...
```

Σημείωση: ***Μπλοκ κώδικα*** *ονομάζεται ένα σύνολο εντολών που έχει σαφή όρια αρχής και τέλος, όπως π.χ. αυτό που ανήκει σε ένα* `if` *ή σε ένα* `else`*. Το κάθε μπλοκ κώδικα έχει συγκεκριμένη εσοχή (indentation).*

Πολύ σημαντικό: *Στην Python, η οριοθέτηση των μπλοκ κώδικα (πού ξεκινάει κάτι και πού τελειώνει), δε γίνεται με άγκιστρα ή αγκύλες κτλ όπως σε πολλές άλλες γλώσσες προγραμματισμού, αλλά με τη δημιουργία εσοχών στον κώδικα (indentation), δηλαδή με την κατάλληλη στοίχιση των στηλών έναρξης της κάθε γραμμής. Κάθε μπλοκ κώδικα έχει το δικό του επίπεδο εσοχής, δηλαδή τη δική του στοίχιση. Με αυτόν τον τρόπο διακρίνει ο διερμηνέας το πού ξεκινάει και το πού τερματίζεται το κάθε μπλοκ κώδικα.*

- Στην περίπτωση (i) εκτελείται το μπλοκ των εντολών μόνο εάν η συνθήκη είναι αληθής. Αν είναι ψευδής, το μπλοκ παρακάμπτεται.
- Στην περίπτωση (ii) εκτελείται είτε το 1ο μπλοκ εντολών (αν η συνθήκη είναι αληθής), είτε το 2ο μπλοκ εντολών (αν η συνθήκη είναι ψευδής).
- Στην περίπτωση (iii) εκτελείται είτε το 1ο μπλοκ (αν η συνθήκη_1 είναι αληθής), είτε το 2ο μπλοκ (αν η συνθήκη_1 είναι ψευδής και η συνθήκη_2 είναι αληθής), κοκ. Αν όλες οι συνθήκες είναι ψευδείς, εκτελείται το τελευταίο μπλοκ, του `else`. Η περίπτωση του `else` δεν είναι απαραίτητο να υπάρχει. Από όλα τα μπλοκ κώδικα, *μόνο ένα* θα εκτελεστεί (ή κανένα, στην περίπτωση που δεν υπάρχει `else` και όλες οι συνθήκες είναι ψευδείς).
- Στην περίπτωση (iv), το 2ο `if` (με πορτοκαλί χρώμα στο παράδειγμα) ανήκει στο 1ο μπλοκ κώδικα, το οποίο εκτελείται εάν η συνθήκη_1 είναι αληθής. Εάν και η συνθήκη_2 είναι αληθής, εκτελείται η εντολή 2, διαφορετικά εκτελείται η εντολή 3. Αν η συνθήκη_1 είναι ψευδής, τότε ολόκληρο το 2ο `if` παρακάμπτεται και εκτελείται το τελευταίο μπλοκ κώδικα (εντολές 4 και 5).

Παραδείγματα `if`:

1. Ο ακόλουθος κώδικας:

```
x = 2
y = 5
if x < y:
    print('Το x είναι μικρότερο του y.')
else:
    print('Το x είναι μεγαλύτερο του y.')
print('Αυτό εκτυπώνεται πάντα!')
```

εκτυπώνει:

```
Το x είναι μικρότερο του y.
Αυτό εκτυπώνεται πάντα!
```

2. Παράδειγμα άρνησης (`not`)

```
>>> temp = 65
>>> if not(temp >= 100):
      print('Η θερμοκρασία είναι κάτω από τη θερμοκρασία βρασμού.')

Η θερμοκρασία είναι κάτω από τη θερμοκρασία βρασμού.
```

3. Μέγιστος δύο αριθμών

```
if x > y:
    max = x
else:
    max = y
```

4. Απόλυτη τιμή του x:

```
if x >= 0:
    abs = x
else:
    abs = -x
```

ή:

```
abs = x
if x<0:
    abs = -x
```

Σημείωση: Προφανώς η «λογική έκφραση» ή η «συνθήκη» που ελέγχεται σε ένα `if` μπορεί να είναι και απλά μια `bool` μεταβλητή, αφού και αυτή, όπως το αποτέλεσμα μιας λογικής έκφρασης, παίρνει τιμές `True` ή `False`. Άρα, π.χ.,, για μια `bool` μεταβλητή `test`, αντί να γράψουμε:

```
if test == True:
```

είναι το ίδιο να γράψουμε:

```
if test:
```

αφού και στις δύο περιπτώσεις το αποτέλεσμα είναι αληθές αν η `test` έχει την τιμή `True` και ψευδές αν η `test` έχει την τιμή `False`.

Αντίστοιχα, για τον έλεγχο:

```
if (test == False)
```

γράφουμε απλά:

```
if not(test)
```

Πρόγραμμα – Παράδειγμα:

Όταν κάποιος υπάλληλος μιας εταιρείας δουλέψει πάνω από 40 ώρες τη βδομάδα, για όλες τις επιπλέον ώρες εργασίας του πληρώνεται 1.5 φορές το κανονικό του ωρομίσθιο. Να γραφεί πρόγραμμα μισθοδοσίας που να υπολογίζει την ακαθάριστη αμοιβή του κάθε υπαλλήλου, συμπεριλαμβανομένων των υπερωριών.

```
# Μεταβλητές για τις βασικές ώρες εργασίας
# και τον πολλαπλασιαστή υπερωριών.
base_hours = 40     # Βασικές ώρες την εβδομάδα
ot_multiplier = 1.5 # Πολλαπλασιαστής υπερωριών

# Εισαγωγή των ωρών εργασίας και του ωρομισθίου.
hours = float(input('Δώσε τον αριθμό των ωρών εργασίας: '))
pay_rate = float(input('Δώσε το ωρομίσθιο: '))

# Υπολογισμός και εμφάνιση της ακαθάριστης αμοιβής.
if hours > base_hours:
    # Υπολογισμός της ακαθάριστης αμοιβής με τις υπερωρίες.
    # Αρχικά, υπολογισμός του αριθμού των υπερωριών.
    overtime_hours = hours - base_hours

    # Υπολογισμός του ποσού πληρωμής των υπερωριών.
    overtime_pay = overtime_hours * pay_rate * ot_multiplier

    # Υπολογισμός της ακαθάριστης αμοιβής.
    gross_pay = base_hours * pay_rate + overtime_pay
else:
    # Υπολογισμός της ακαθάριστης αμοιβής χωρίς υπερωρίες.
    gross_pay = hours * pay_rate

# Εμφάνιση της ακαθάριστης αμοιβής.
print(f'Η ακαθάριστη αμοιβή είναι € {gross_pay:.2f}’)
```

Πρόγραμμα – Παράδειγμα:

Πρόγραμμα που μετατρέπει τη βαθμολογία από την κλίμακα 0-100 στην κλίμακα “A, B, C, D, F”.

```
score = int(input('Δώσε τη βαθμολογία σου (0-100): '))

if score > 90:
    grade = 'A'
elif score > 80:
    grade = 'B'
elif score > 70:
    grade = 'C'
elif score > 60:
    grade = 'D'
else:
   grade = 'F'

print(f'Με βαθμολογία {score} στα 100 ο βαθμός σου είναι: {grade}’)
```

Πρόγραμμα – Παράδειγμα:

Ας μελετήσουμε το πρόβλημα του υπολογισμού του ελάχιστου της συνάρτησης $f(x)=x^2+bx+c$ στο διάστημα $[L,R]$.

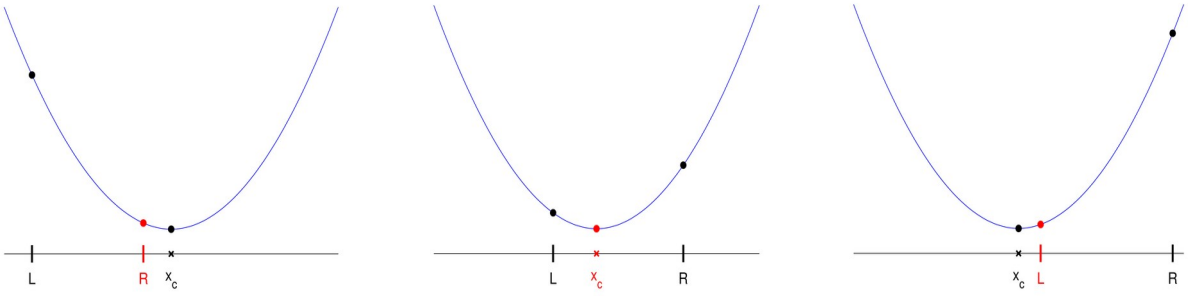
Γνωρίζουμε πως το ελάχιστο βρίσκεται στο κρίσιμο σημείο $x_c=-\frac{b}{2}$.

- Αν το $x_c \in [L,R]$, τότε το ελάχιστο είναι το $f(x_c)$.
- Αν το $x_c \notin [L,R]$, τότε το ελάχιστο είναι είτε το $f(L)$, είτε το $f(\mathrm{R})$.

Να γραφεί πρόγραμμα, το οποίο να ζητάει τους πραγματικούς αριθμούς *L*, *R*, *b* και *c* και να εμφανίζει το ελάχιστο της $f(x)$ στο $[L,R]$, καθώς και την τιμή του *x* στην οποία εμφανίζεται.

Υπάρχουν 3 περιπτώσεις:

(1) (2) (3)



(1)	(2)	(3)
$x_c > R$	$L \le x_c \le R$	$x_c < L$
min = f(R)	min = f(x_c)	min = f(L)

Το πρόγραμμα σε Python:

```
# Είσοδος δεδομένων
b = float(input('Δώσε το b: '))
c = float(input('Δώσε το c: '))
L = float(input('Δώσε το L: '))
R = float(input('Δώσε το R, με L<R: '))
print(f'Δευτεροβάθμια: x^2+bx+c, b = {b:.2f}, c = {c:.2f}’)
print(f'Διάστημα: [L,R], L = {L:.2f}, R = {R:.2f}')

# Υπολογισμός κρίσιμου σημείου
xc= - b/2
if xc < L:
    xmin = L
elif L <= xc <=R:
    xmin = xc
else:
    xmin = R
fmin = xmin**2 + b*xmin + c
# Εμφάνιση υπολογισμών
print(f'x ελαχιστοποίησης = {xmin:.2f}')
print(f'Ελάχιστη τιμή της f = {fmin:.2f}')
```

Σύγκριση συμβολοσειρών

Δύο συμβολοσειρές μπορούν να συγκριθούν με τη χρήση σχεσιακών τελεστών, ως εξής:

α) Με τους τελεστές == και != για τον έλεγχο ταύτισης ή μη ταύτισής τους.

π.χ.:

```
username = input('Δώσε το username σου: ')
registered_user = 'admin'
if username == registered_user:   # Case sensitive σύγκριση
    print('Απόκτηση πρόσβασης')
else:
    print('Απόρριψη πρόσβασης')
```

β) Με τους τελεστές <, <=, >, >= για τη σύγκριση μεταξύ τους, ως εξής:

- Αν πρόκειται για συμβολοσειρές ενός χαρακτήρα, συγκρίνονται οι κωδικοί ASCII τους, π.χ., το:

```
if 'a' < 'b':     # σύγκριση του ascii κωδικού τους
    print('a μικρότερο του b')
else:
    print('a μεγαλύτερο του b')
```

εκτυπώνει:

```
a μικρότερο του b
```

αφού ο κωδικός ASCII του a είναι το 97 ενώ του b το 98.

- Αν πρόκειται για συμβολοσειρές πολλών χαρακτήρων:
  - Σύγκριση των χαρακτήρων έναν προς έναν
  - Στην πρώτη διαφοροποίηση χαρακτήρα, μικρότερη συμβολοσειρά είναι αυτή με τον χαρακτήρα με τον μικρότερο κωδικό ASCII
  - Για διαφορετικού μήκους συμβολοσειρές που μέχρι το τέλος της μικρότερης συμπίπτουν, μικρότερη είναι αυτή με το μικρότερο μήκος.

Π.χ., το:

```
name1 = 'Marios'
name2 = 'Markos'
if name1 > name2:
    print('Το', name1, 'είναι μεγαλύτερο του', name2)
else:
    print('Το', name2, 'είναι μεγαλύτερο του', name1)
```

εκτυπώνει:

```
Το Markos είναι μεγαλύτερο του Marios
```

αφού το k έχει μεγαλύτερο κωδικό ASCII από το i.
Το:

```
name1 = 'Hello'
name2 = 'Hell'
if name1 > name2:
    print('Το', name1, 'είναι μεγαλύτερο του', name2)
else:
    print('Το', name2, 'είναι μεγαλύτερο του', name1)
```

εκτυπώνει:

```
Το Hello είναι μεγαλύτερο του Hell
```

αφού μέχρι το μήκος της μικρότερης συμβολοσειράς οι δύο συμβολοσειρές συμπίπτουν, άρα μικρότερη είναι αυτή με το μικρότερο μήκος.

Β. Δομές επανάληψης

Μία δομή επανάληψης κάνει μία εντολή ή ένα σύνολο εντολών να εκτελούνται επαναλαμβανόμενα.

Υπάρχουν δύο βασικές δομές επανάληψης:

1. Βρόχος ελεγχόμενος με μετρητή (count-controlled loop)
   Επαναλαμβάνεται για ένα συγκεκριμένο αριθμό φορών (εντολή `for`)

2. Βρόχος ελεγχόμενος με συνθήκη (condition-controlled loop)
   Χρησιμοποιεί μία συνθήκη αληθούς/ψευδούς για να ελέγξει το πόσες φορές θα επαναληφθεί (εντολή `while`)

Β1. Η εντολή `for` (for-loop)

Ο βρόχος που υλοποιείται με την εντολή `for` εκτελεί έναν συγκεκριμένο αριθμό επαναλήψεων.

Γενική σύνταξη της εντολής `for`:

```
for μεταβλητή in [τιμή1, τιμή2, κτλ.]:
    εντολή
    εντολή
    κτλ.
```

Το μπλοκ των εντολών που ακολουθούν σε εσοχή την εντολή `for` εκτελούνται τόσες φορές όσο είναι το πλήθος των τιμών μέσα στις αγκύλες. Σε κάθε επανάληψη, εκχωρείται στη μεταβλητή μία από τις τιμές αυτές, ξεκινώντας από την πρώτη προς την τελευταία.

Π.χ.,

```
for num in [1, 2, 3, 4, 5]:
```

```
    print(num)
```

```
1
2
3
4
5
```

```
for name in ['Χιούη', 'Λιούη', 'Ντιούη']:
    print(name)
```

```
Χιούη
Λιούη
Ντιούη
```

Χρήση της συνάρτησης `range`:

Η ενσωματωμένη συνάρτηση `range` της Python απλουστεύει τη διαδικασία σύνταξης της εντολής `for`. Δημιουργεί έναν τύπο αντικειμένου γνωστό ως αντικείμενο τιμών βρόχου ή λίστα τιμών βρόχου (iterable). Η λίστα αυτή περιέχει μια αλληλουχία τιμών που μπορεί να τις διατρέξει ένας βρόχος.

Η `range` μπορεί να χρησιμοποιηθεί με *τρεις* τρόπους σε ένα `for`:

1. *Με ένα όρισμα:*

   ```
   for μεταβλητή in range(x):
   ```

   Δημιουργείται λίστα τιμών βρόχου με στοιχεία *από 0 έως και x-1*. Π.χ.,

   ```
   for num in range(5):
       print(num)
   ```

   ```
   0
   1
   2
   3
   4
   ```

2. *Με δύο ορίσματα:*

   ```
   for μεταβλητή in range(x, y):
   ```

   Δημιουργείται λίστα τιμών βρόχου με στοιχεία *από x έως και y-1*. Π.χ.,

   ```
   for num in range(1,4):
       print(num)
   ```

   ```
   1
   2
   3
   ```

3. *Με τρία ορίσματα:*

```
for μεταβλητή in range(x, y, z):
```

Δημιουργείται λίστα τιμών βρόχου με στοιχεία *από x έως το πολύ y-1 με βήμα z*, αν το z είναι θετικό, ή ώς το πολύ y+1 εάν το z είναι αρνητικό. Π.χ.,

```
for num in range(1,10, 2):
    print(num, end = ' ')
```

1 3 5 7 9

```
for num in range(10,5,-2):
    print(num, end = ' ')
```

10 8 6

Ουσιαστικά, πρώτα ελέγχονται τα όρια (x, y) και αρχικοποιούνται οι οριακές αυτές τιμές εάν είναι εφικτό, και στη συνέχεια δημιουργούνται οι ενδιάμεσες τιμές του μετρητή του `for`. Άρα το παρακάτω:

```
for i in range(2, 9, 9)
    print(i)
```

*θα εκτυπώσει **2*** (δηλαδή θα κάνει μόνο μία επανάληψη, αφού θα δημιουργηθεί το πρώτο στοιχείο του μετρητή (2), ενώ το επόμενο δεν θα δημιουργηθεί (με βάση το βήμα 0, θα ήταν το 11), αφού είναι μετά την τιμή τερματισμού (9). Ενώ το:

```
for i in range(2, 2, 9)
    print(i)
```

δεν θα εκτυπώσει τίποτα (δηλαδή δεν θα κάνει καμία επανάληψη, αφού η αρχική τιμή (2) είναι μεγαλύτερη από την τελική τιμή (2-1=1)).

Παραδείγματα `for`:

```
for odd in range(1, 100, 2):
    print(odd)

for even in range(2, 101, 2):
    print(even)

for countdown in range(10, -1, -1):
   print(countdown)
```

1. Υπολογισμός τρέχοντος αθροίσματος (άθροισμα αριθμών που αυξάνεται με κάθε επανάληψη ενός βρόχου).

```
max = 5      # Το μέγιστο πλήθος αριθμών
total = 0.0  # Αρχικοποίηση της μεταβλητής του αθροιστή.

for counter in range(max):
```

```
    number = int(input('Δώσε έναν αριθμό: '))
    total = total + number

print('Το άθροισμα είναι', total)
```

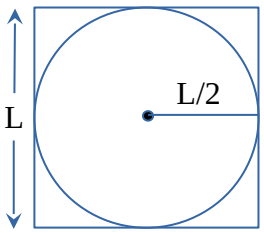
Έξοδος (η είσοδος με έντονους χαρακτήρες):

```
Δώσε έναν αριθμό: 3
Δώσε έναν αριθμό: 2
Δώσε έναν αριθμό: 5
Δώσε έναν αριθμό: 4
Δώσε έναν αριθμό: 1
Το άθροισμα είναι 15.0
```

2. Μέσος όρος 10 ακέραιων αριθμών από τον χρήστη:

```
n=10
total=0
for k in range(n):
   num = int(input('Δώσε έναν αριθμό: '))
   total += num
avg = total/n
print(f'Μέσος όρος: {avg:.3f}')
```

3. Προσέγγιση του π με τη μέθοδο Monte Carlo



→ Ρίχνουμε *n* βελάκια στο τετράγωνο

→ hit: το βελάκι μέσα στον κύκλο

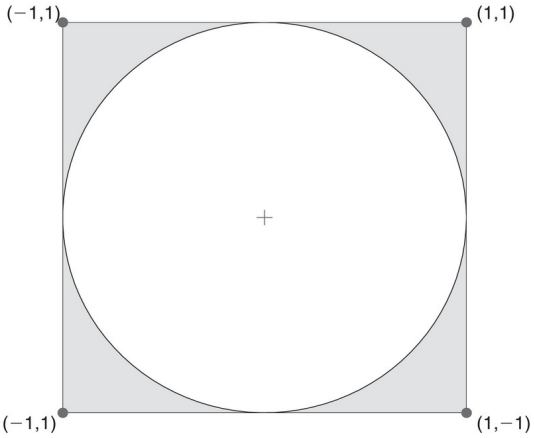
→ Το ποσοστό των hits προσεγγίζει τον λόγο των δύο εμβαδών:

$$\frac{hits}{n} \simeq \frac{E_{κύκλου}}{E_{τετραγώνου}} = \frac{\pi \frac{L^2}{4}}{L^2} \Rightarrow \pi \simeq 4\frac{hits}{n}$$

Άρα:

```
for i in range(n)
    # ρίχνω το βελάκι i
    # αν το βελάκι είναι μέσα στον κύκλο:
    # hits = hits+1
myPi = 4*hits/n
```

Ας υποθέσουμε ότι έχουμε τον μοναδιαίο κύκλο με κέντρο (0,0):



→ Ένα σημείο (x,y) είναι hit εάν $x^2+y^2 \leq 1$

→ Πρέπει να βρεθεί τρόπος να παράγουμε τυχαία σημεία στο τετράγωνο.

Θέλουμε: -1<x<1 και -1<y<1.

Η `random.uniform(a,b)` δίνει τυχαίους `float` αριθμούς στο διάστημα [a, b) ή [a, b] (εξαρτάται από στρογγυλοποίησεις). Για να την χρησιμοποιήσουμε σε πρόγραμμά μας πρέπει να συμπεριληφθεί στο πρόγραμμα η εντολή: `import random` – περισσότερα για αυτό σε επόμενη ενότητα).

```
import random
n = 100000
hits = 0
for i in range(n):
    # ρίχνω το βελάκι i
    x = random.uniform(-1,1)
    y = random.uniform(-1,1)
    # ελέγχω αν είναι hit
    if x**2 + y**2 <= 1:
        hits += 1
myPi = 4*hits/n
print(myPi)
```

4. Εισαγωγή συγκεκριμένου πλήθους αριθμών και εύρεση του ελάχιστου και του μέγιστου, καθώς και των θέσεών τους (σειρά εισαγωγής τους).

α) Με την 1η εισαγωγή εκτός του for-loop και μία λιγότερη επανάληψη του loop:

```
n = int(input('Δώσε το πλήθος των αριθμών: '))
num = int( input('Δώσε ακέραιο αριθμό: '))
minimum = num; maximum = num   # Το ; λειτουργεί ως διαχωριστής
pos_min = 1; pos_max = 1       # γραμμών κώδικα
for rep in range(1,n):
    num = int(input('Δώσε ακέραιο αριθμό: '))
    if num > maximum:
        maximum = num
        pos_max = rep+1
    if num < minimum:
```

```
        minimum = num
        pos_min = rep + 1
print('Ο ελάχιστος είναι το', minimum, 'και ο μέγιστος το', maximum)
print('Ο ελάχιστος βρίσκεται', pos_min, '-ος και ο μέγιστος ', pos_max, '-ος')
```

β) Με έλεγχο μέσα στο for-loop για διαφοροποίηση κατά την πρώτη επανάληψη:

```
n = int(input('Δώσε το πλήθος των αριθμών: '))
for rep in range(n):
    num = int(input('Δώσε ακέραιο αριθμό: '))
    if rep == 0:   # πρώτη επανάληψη
        minimum = num; maximum = num
        pos_max = 1; pos_min = 1
    if num > maximum:
        maximum = num
        pos_max = rep+1
    if num < minimum:
        minimum = num
        pos_min = rep + 1
print('Ο ελάχιστος είναι το', minimum, 'και ο μέγιστος το', maximum)
print('Ο ελάχιστος βρίσκεται', pos_min, '-ος και ο μέγιστος ', pos_max, '-ος')
```

5. Καταμέτρηση άρτιων και περιττών

```
n = int(input('Δώσε το πλήθος των αριθμών: '))
count_even = 0; count_odd = 0
for rep in range(n):
    num= int(input('Δώσε ακέραιο αριθμό: '))
    if num%2 == 0:
        count_even += 1
    else:
        count_odd += 1
print('Δόθηκαν', count_even, 'άρτιοι και', count_odd, 'περιττοί.')
```

---

2 βασικές ιδιότητες της εντολής `for` στην Python:

- Οποιαδήποτε αλλαγή της τιμής της μεταβλητής-μετρητή της `for` μέσα στο μπλοκ των εντολών, δεν επηρεάζει την τιμή που θα πάρει ο μετρητής στην επόμενη επανάληψη, η οποία θα είναι η επόμενη τιμή της λίστας τιμών βρόχου. Π.χ.,

```
for i in range(5):
    print(i, end = ' ')
    i *= 2
    print(i)
```

```
0 0
1 2
2 4
3 6
```

```
4 8
```

- Η μεταβλητή-μετρητής ενός `for` συνεχίζει να υπάρχει και μετά το τέλος της εκτέλεσης των εντολών του `for`, έχοντας ως τιμή την τιμή που είχε κατά το τέλος της τελευταίας επανάληψης των εντολών. Π.χ.,

```
for i in range(3):
    print('Hello')
print(i)
```

```
Hello
Hello
Hello
2
```

Ένθετα `for` (nested for-loops)

```
N = int(input('N? '))
for i in range(N):
    for j in range(N):
        if i<j:
            print('.', end='  ')
        else:
            print('*', end='  ')
    print()
```

π.χ. για N=5:

```
*   .   .   .   .
*   *   .   .   .
*   *   *   .   .
*   *   *   *   .
*   *   *   *   *
```

```
N = int(input('N? '))
for i in range(N):
    for j in range(i+1):
        print('*', end='')
    print()
```

ή

```
N = int(input('N? '))
for i in range(N):
    print('*' * i)
```

π.χ. για N=8:

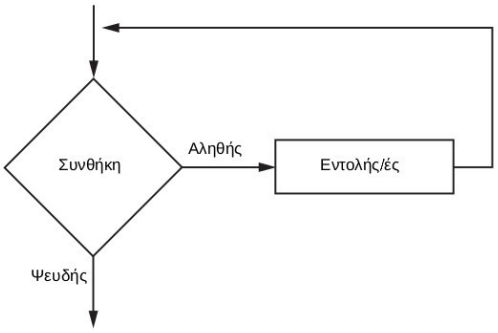
```
*
**
***
****
*****
******
*******
********
```

Β2. Η εντολή `while` (while-loop)

Ο βρόχος που υλοποιείται με την εντολή `while` εκτελείται επαναλαμβανόμενα όσο κάποια συνθήκη είναι αληθής.

Γενική σύνταξη της εντολής `while`:

```
while συνθήκη:
    εντολή
    εντολή
    κτλ.
```



Το μπλοκ των εντολών που ακολουθούν σε εσοχή την εντολή `while` εκτελούνται όσο η *συνθήκη* είναι `True`. Ο βρόχος `while` είναι *βρόχος προελέγχου*, δηλαδή ελέγχει αρχικά τη συνθήκη πριν πραγματοποιήσει την πρώτη επανάληψη.

Η συνθήκη θα πρέπει με κάποιον τρόπο να μεταβάλλεται κατά τη διάρκεια εκτέλεσης του μπλοκ εντολών του `while`, έτσι ώστε σε κάποια στιγμή να γίνει `False` και να τερματιστεί η εκτέλεση του βρόχου. Σε διαφορετική περίπτωση, ο βρόχος είναι *ατέρμονας βρόχος* (infinite loop).

Ο βρόχος `while` χρησιμοποιείται όταν *δεν* είναι δεδομένος ο αριθμός επαναλήψεων ενός τμήματος κώδικα. Η δομή του είναι πιο γενική από αυτή του `for`, και μπορεί πάντα να χρησιμοποιηθεί αντί ενός `for` (το αντίθετο *δεν* ισχύει).

Π.χ.,

```
x = 5
while x > 0:
   print(x, end = ' ')
   x -=1
```

```
5 4 3 2 1
```

Άλλα παραδείγματα `while`:

1. Άθροισμα άγνωστου πλήθους θετικών αριθμών (χρήση τιμής/ών τερματισμού - sentinel):

```
sum = 0
x = int(input('Δώσε θετικό αριθμό: '))
while x > 0:
    sum += x
    x = int(input('Δώσε θετικό αριθμό ή μη-θετικό για τερματισμό: '))
print(sum)
```

→ Αν θέλαμε να εμφανίσουμε και πόσοι αριθμοί δόθηκαν;

2. Άθροισμα ψηφίων θετικού ακεραίου:

Για τον ακέραιο *n*:

a) Εύρεση τελευταίου ψηφίου: `n%10`
   π.χ. `1975%10 → 5`
b) Αποκοπή τελευταίου ψηφίου: `int(n/10)`
   π.χ. `int(1975/10) → 197`

```
n = int(input('Δώσε θετικό ακέραιο: '))
n2=n # κρατάμε το n που δόθηκε για να το εμφανίσουμε στην έξοδο στο τέλος.
dsum = 0
while n>0:   # αν βάζαμε n>=0: ατέρμονες επαναλήψεις
    dsum  += n%10
    n = int(n/10)
print('Το άθροισμα των ψηφίων του', n2, 'είναι', dsum)
```

3. Βρόχος επαλήθευσης εισόδου:

Πρόγραμμα υπολογισμού δείκτη μάζας σώματος (βάρος / ύψος^2). Να εκτελείται για μία τιμή ύψους αλλά πολλαπλές τιμές βάρους (επαναλαμβανόμενη ερώτηση βάρους και εξαγωγή του αντίστοιχου δείκτη). Να τερματίζει με τιμή βάρους 0 και να ελέγχει για μη αποδεκτές (αρνητικές) τιμές σε ύψος ή βάρος (αν δοθεί μη αποδεκτή τιμή να την ζητάει ξανά μέχρι να δοθεί αποδεκτή τιμή).

```
# Είσοδος ύψους (με έλεγχο)
height = int(input('Δώσε το ύψος σου: '))
while height <= 0:
    print("Μάλλον έδωσες λάθος ύψος, προσπάθησε ξανά!")
    height = int(input('Δώσε το ύψος σου: ') )

# Είσοδος βάρους (με έλεγχο)
weight = int(input('Δώσε το βάρος σου (0 για έξοδο): ') )
while weight < 0:
    print("Μάλλον έδωσες λάθος βάρος, προσπάθησε ξανά!")
    weight = int(input('Δώσε το βάρος σου (0 για έξοδο): '))

# Υπολογισμός δείκτη μάζας για επαναλαμβανόμενες τιμές βάρους
while weight != 0:
    bmi = weight/height**2
    print("Ο δείκτης μάζας σώματός σου είναι:", bmi)
    weight = int(input('Δώσε το βάρος σου (0 για έξοδο): '))
    while weight < 0:
        print("Μάλλον έδωσες λάθος βάρος, προσπάθησε ξανά!")
        weight = int(input('Δώσε το βάρος σου (0 για έξοδο): '))
```

Σχέση `for` και `while`:

`for` *μετ/τη* `in range(AT,TT,B):`
`    ` *εντολές*
	μετ/τη `= AT;`
	`while` *μετ/τη* `< TT: # ή: > TT αν B<0`
	`    ` *εντολές*
	`    ` *μετ/τη* `+= B`

Παράδειγμα:

`for i in range(5,20,2):`
`    print(i, end = ' ')`
 	
Έξοδος:
 	
`5 7 9 11 13 15 17 19`
	`i = 5`
	`while i < 20:`
	`    print(i, end = ' ')`
	`    i += 2`
	
	Έξοδος:
	
	`5 7 9 11 13 15 17 19`

- Το `while` μπορεί να χρησιμοποιηθεί πάντα.
- Το `for` προτιμάται όταν το πλήθος των επαναλήψεων είναι γνωστό.
- Όταν το πλήθος των επαναλήψεων είναι άγνωστο: `while`

Π.χ., στο ακόλουθο παράδειγμα δε μπορεί να χρησιμοποιηθεί `for`:

- Αριθμός ετών που απαιτούνται για διπλασιασμό κεφαλαίου x ευρώ, με ετήσιο επιτόκιο 5%.

```
money = float(input('Κεφάλαιο? '))
years = 0
goal = 2*money
while money < goal
    money *= 1.05
    years += 1
print(years)
```