ΕΚΠΑ 2024
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ

PYTHON

Inspired by

Michael Drakopoulos

Created by

Angelos Giannopoulos

Κάποιες Πράξεις…

```
//                  πηλίκο ακέραιας διαίρεσης
%                   υπόλοιπο ακέραιας διαίρεσης
%10                 τελευταίο ψηφίο
%100                δύο τελευταία ψηφία κλπ
2e5 = 2*100,000     5 μηδενικά
1e2 = 1*100         δεν είναι ορισμένο το e1 - θέλει το 1 μπροστά!
Aen = A * 10**N
```

Ενσωματωμένες Συναρτήσεις

abs() απόλυτη τιμή
max() μέγιστο στοιχείο
min() ελάχιστο στοιχείο

Κάποιες συναρτήσεις στο **module math**

sqrt() ρίζα
floor() στρογγυλοποίηση προς τα κάτω
ceil() στρογγυλοποίηση προς τα πάνω
cos() συνημίτονο
sin() ημίτονο
tan() εφαπτομένη
exp() εκθετική
log() λογάριθμος με βάση το e
log2() λογάριθμος με βάση το 2
log10() λογάριθμος με βάση το 10
gcd() ΜΚΔ

Για να χρησιμοποιήσουμε αυτές τις συναρτήσεις και τις σταθερές, πρέπει πρώτα να εισάγουμε το module math, μέσω της import math και στη συνέχεια να τις καλούμε όποτε τις θέλουμε, γράφοντας math.function(...)

Σταθερές στο **module math**

pi, e, inf

Μετατροπή Τύπων

float(10**)**	**int(**6.2**)**	**int(**'2000'**)**	**str(**2**)**
10.0	6 **κόβει το δεκαδικό μέρος**	2000	'2'

! Για τις Bool κάθε τιμή διαφορετική του μηδενός είναι True

Εδώ το 0 εννοείται με τη γενική έννοια και όχι απαραίτητα με την αριθμητική. Για παράδειγμα, στις int το μηδέν είναι το 0, στις float είναι το 0.0 και στις str ειναι το κενό ''.

Βασικές Εντολές: if, elif, else, while, for, end

! Η εντολή **for** έχει πάρα πολλές χρήσεις στην Python και γράφεται με τη χρήση της συνάρτησης **range()** η οποία καλείται με 3 τρόπους:
   Η **range(K)** σκανάρει τις τιμές από 0 έως και K-1 με βήμα 1.
   Η **range(M,N)** σκανάρει τις τιμές από M έως και N-1 με βήμα 1.
   Η **range(M,N,S)** σκανάρει τις τιμές από M έως και N-1 με βήμα S, αν S>0. Αν S<0, θα πρέπει M>N και τότε η συνάρτηση σκανάρει τιμές από M έως και N+1.

Άλλες Εντολές: break, continue

- Η break σταματάει τις επαναλήψεις.
- Η continue συνεχίζει στην επόμενη εντολή κάνοντας skip τις εντολές του ίδιου κόμβου.

STRINGS

- Οι str περικλείονται μέσα σε '...' ή "...".
  Αν μέσα στα εισαγωγικά υπάρχουν κι άλλα εισαγωγικά, τότε χρησιμοποιώ τριπλά εισαγωγικά '''...''' ή """...""".

- Τα τριπλά εισαγωγικά χρησιμοποιούνται και για str που εκτείνονται σε πολλές γραμμές.

- Μπορώ να αλλάξω γραμμή με τη χρήση του χαρακτήρα **\n** και να δημιουργήσω μεγάλο κενό (tab) με το χαρακτήρα **\t**, τοποθετώντας τους μεταξύ των υπόλοιπων χαρακτήρων **μέσα** στα εισαγωγικά.

- Η συνάρτηση **len()** επιστρέφει το πλήθος των χαρακτήρων μιας str.

```
S = 'PYTHON'
len(S)
------------
6
```

! Ο 1ος χαρακτήρας βρίσκεται στη θέση 0 και ο τελευταίος βρίσκεται στη θέση len(s)-1

! Μπορώ να δουλέψω και με αρνητικούς δείκτες:
S[-1] = S[len(S)-1] και **S[0] = S[-len(S)]**

Γράφοντας **S[I:J]** επιστρέφονται όλοι οι χαρακτήρες από τη θέση I έως και τη θέση J-1.

```
S = 'PYTHON'
S[0:3]
------------
'PYT'
```

Γράφοντας **S[I:J:K]** επιστρέφονται όλοι οι χαρακτήρες από τη θέση I έως και τη θέση J-1 με βήμα K.

! **Παράλειψη του I ⇒ I=0**

Παράλειψη του J ⇒ J=len(S)

```
S = 'PYTHON'
S[1::2]
------------
'YHN'
```

! Αν θέλουμε να διατρέξουμε μια str S, μπορούμε κι έτσι:

```
for c in S:
    εντολές
        .
        .
        .
```

#Έτσι αποφεύγουμε τους δείκτες!

! Η Python ακολουθεί την κωδικοποίηση **Unicode** για την αναπαράσταση χαρακτήρων. Κάθε χαρακτήρας αντιστοιχεί σε έναν ακέραιο στο διάστημα [0, 1114111].
Η συνάρτηση **ord('x')** επιστρέφει **τη θέση του χαρακτήρα x**.

! Υπάρχουν ορατοί και αόρατοι χαρακτήρες. Οι συνηθέστεροι ορατοί είναι μεταξύ των θέσεων 32 και 128.
Η συνάρτηση **chr(i)** επιστρέφει **το χαρακτήρα που βρίσκεται στη θέση i**.

Κάτι για την print…

Υπάρχουν 2 παράμετροι που μπορούν να χρησιμοποιηθούν μέσα στην print: Η **sep** και η **end**.

- Η παράμετρος sep διαχωρίζει τις μεταβλητές που θα τυπωθούν με το χαρακτήρα της επιλογή μας.

```
print('apple', 'banana', 'cherry', sep='-')
-----------------------------------------
apple-banana-cherry
```

- Η παράμετρος end προσθέτει τον χαρακτήρα που θα επιλέξουμε στο τέλος της γραμμής. Αν έχουμε πολλές print τη μία κάτω από την άλλη με την end μέσα σε όλες, παρακάμπτεται το default format της Python (που είναι να αλλάζει γραμμή για κάθε print) και οι print "συνενώνονται" με τα ορίσματα των end.

```
print('apple', end=', ')
print('banana', end='! ')
print('cherry')
------------------------
apple, banana! Cherry
```

- Για τη **σύγκριση str** χρησιμοποιούμε τους τελεστές σύγκρισης:
== != < <= > >=

! Γενικά, ισότητα αντικειμένων δεν σημαίνει και ταύτιση τους

Για τον έλεγχο της **ισότητας**, χρησιμοποιούμε τον τελεστή **==**
Για τον έλεγχο της **ταυτότητας**, χρησιμοποιούμε τον τελεστή **is**

ΜΕΘΟΔΟΙ

Πρόκειται για εξειδικευμένες συναρτήσεις που σχετίζονται με κάποιο αντικείμενο και αντιστοιχούν στις ενέργειες που μπορούμε να κάνουμε με η πάνω στο αντικείμενο αυτό.

Καλούμε τη μέθοδο method στο αντικείμενο obj με ορίσματα args, μέσω της **obj.method(args)**

Μέθοδοι για str **! ΔΕΝ ΑΛΛΑΖΟΥΝ ΤΗΝ ΑΡΧΙΚΗ STR - ΔΗΜΙΟΥΡΓΟΥΝ ΑΝΤΙΓΡΑΦΑ !**

(επιστρέφουν True ή False)

.isalnum() είναι όλα αλφαριθμητικοί;
.isalpha() είναι όλα γράμματα;
.isdigit() είναι όλα ψηφία 0-9;
.islower() είναι όλα τα γράμματα πεζά;
.isupper() είναι όλα τα γράμματα κεφαλαία;
.isspace() είναι όλα κενά ή αλλαγές γραμμών;

.startswith() ξεκινάει με συγκεκριμένο substr;
.endswith() τελειώνει με συγκεκριμένο substr;

(επιστρέφουν νέα str)

.lower() μετατροπή σε πεζά
.upper() μετατροπή σε κεφαλαία
Πολύ χρήσιμές για να εξαλείψω τη διάκριση πεζών-κεφαλαίων σε έλεγχο

.lstrip() αφαιρεί τους λευκούς χαρακτήρες από αρχή και τέλος
.rstrip() αφαιρεί τους λευκούς χαρακτήρες από την αρχή
.strip() αφαιρεί τους λευκούς χαρακτήρες από το τέλος

Λευκοί χαρακτήρες είναι τα κενά και τα tabs

.lstrip('c') αφαιρεί όλα τα 'c' από την αρχή
.rstrip('c') αφαιρεί όλα τα 'c' από το τέλος
.strip('c') αφαιρεί όλα τα 'c' από αρχή και τέλος

.replace(old,new**)** Αντικαθιστά όλα τα *old* str με το *new* str

(επιστρέφουν int)

.count(substr**)** επιστρέφει το πλήθος εμφανίσεων της substr
.find(substr**)** επιστρέφει την 1η θέση εμφάνισης της substr ή -1

! ΔΕΝ ΑΛΛΑΖΟΥΝ ΤΗΝ ΑΡΧΙΚΗ STR – ΔΗΜΙΟΥΡΓΟΥΝ ΑΝΤΙΓΡΑΦΑ !

Παραδείγματα

```
bot1='R2d2'
bot2='c3po'
bot1.isalnum(), bot1.isalpha(), bot2.islower(). '42'.isdigit()
-------------------------------------------------------------
(True, False, True, True)
```

```
'\n \t\n'.isspace()
-------------------
True
```

```
filename=input('Give a filename ')
if filename.endswith('.py'):
   print(filename, ' is a Python file')
elif filename.endswith('.docx'):
   print(filename, ' is a Word Document')
-----------------------------------------
Give a filename myproject.py
myproject.py is a Python file
```

```
movie='The Life Of Brian'
new=movie.upper()
print(new, movie.lower(), movie, sep='\n')
------------------------------------------
THE LIFE OF BRIAN
the life of brian
The Life Of Brian
```

```
s=input('Type Yes to continue ')
if s.upper()=='YES':
   print('Continuing...')
else
   print('Bye!')
m='speed v=10m/s'
print(m.lstrip('pse d')
print(m.rstrip('ms/')
print(m.strip('pse d/'))
------------------------
v=10m/s
speed v=10
v=10m
```

```
fruit='Banana'
print(fruit.count('a'), fruit.count('an'), fruit.count('b'))
print(fruit.replace('an','aNN'))
-------------------------------------------------------------
(3,2,0)
BaNNaNNa
```

Μορφοποίηση Str

f-strings

Γράφω **f** πριν τα εισαγωγικά και **{variable}** εντός τους, για να αναφερθώ στη *μεταβλητή* variable, αντί του '{variable}' σαν str.

```
first='Isaac'
last='Newton'
message=f"{last}'s name was {first}"
fact=f'{first[0]}. {last.upper()} likes apples'
print(message, fact)
-----------------------------------
Isaac's name was Newton
I. NEWTON likes apples
```

```
import math
R=10
line=f'The perimeter of a circle with R={R} is {2*math.pi*R}'
print(line)
-------------------------------------------------------------
The perimeter of a circle with R=10 is 62.83185307179586
```

```
import math
R=10/3
line=f'The per. of cir. with R={R:.3f} is {2*math.pi*R:12.1f}'
-------------------------------------------------------------
The per. Of cir. with R=3.333 is           20.9
```

! Η {2*math.pi*R:12.1f} επιστρέφει την περίμετρο με έυρος 12 θέσεων σύνολο, εκ των οποίων η 1 είναι δεκαδική

LISTS

Δημιουργία

```
L = ['a','b','c']
```

Μετατροπή τύπων

Με τη συνάρτηση **list()**

```
list('abc')
-------------
['a','b','c']
```

! Πρέπει αυτό που θα λάβει η συνάρτηση list() να είναι iterable αντικείμενο, δηλαδή να τηρεί το πρωτόκολλο επανάληψης, ώστε να μπορεί να το "σπάσει"

Παράδειγματα iterable αντικειμένων είναι οι str και η range()

```
list(range(1,9,3))
------------------
[1,4,7]
```

- **Συνένωση** λιστών κάνω με τον τελεστή **+**
- **Επανάληψη** λιστών κάνω με τον τελεστή ***

- **Η συνάρτηση len() επιστρέφει το μήκος της λίστας**

Διαχείριση στοιχείων με δείκτες

Έστω λίστα **L**.

- **L[i]** είναι το στοιχείο στη θέση i **Το 1ο στοιχείο είναι στη θέση 0**
  όπου **i ακέραιος στο διάστημα [-len(L),len(L)-1]**
  Εκτός αυτού του διαστήματος έχουμε **IndexError**

- Με τον τελεστή **:** **τεμαχίζουμε λίστες**

  L[I:J] υπολίστα από τον δείκτη I έως και τον J-1

  ! Αν γράψω L[5:-1] τότε το -1 ανάγεται στο θετικό του, len(L)

  L[I:J:K] υπολίστα από το δείκτη I έως και τον J-1 με βήμα K

  ! Παράλειψη J ⇒ J=-1

  Παράλειψη I ⇒ I=0

  ! I, J εκτός ορίων ⇒ μέγιστη δυνατή sublist, ΟΧΙ σφάλμα!

Τροποποίηση Λιστών

- **Αντικατάσταση**
- **Διαγραφή**
- **Παρεμβολή**

Όλες γίνονται με εκχωρήσεις!

Παραδείγματα

```
L=[1,2,3]
L[2]=10
print(L)
---------
[1,2,10]
```

```
L=[7,18,31]
L[1:3]=[-40,40]
print(L)
----------------
[7,-40,40]
```

Αντικατάσταση

```
L=[7,18,31]
L[1:2]=[]
print(L)
-----------
[7,31]
```

```
L=[7,18,31]
L[1:3]=[]
print(L)
-----------
L=[7]
```

Διαγράφει και περισσότερα από 1 στοιχεία

Διαγραφή

! Η εντολή L[i]=[] ΔΕΝ διαγράφει το στοιχείο στη θέση i, αλλά το αντικαθιστά με την κενή λίστα

```
L=[7,18,31]
L[2]=[]
print(L)
-----------
[7,18,[]]
```

```
L=[7,18,31]
L[1:1]=[-40,40]
print(L)
----------------
[7,-40,40,18,31]
```

L[i:i]=[*κάτι*] για να βάλουμε το *κάτι* στη θέση i, αφού στείλουμε το sublist L[i:-1] μια θέση δεξιότερα

Παρεμβολή

Συναρτήσεις για lists

κλήση μέσω function(list)

`len()`	μήκος της λίστας
`max()`	μέγιστο στοιχείο
`min()`	ελάχιστο στοιχείο
`sum()`	άθροισμα στοιχείων
`sorted()`	διάταξη σε αύξουσα σειρά
`sorted( ,reverse=True)`	διάταξη σε φθίνουσα σειρά

Μέθοδοι για lists

κλήση μέσω `list.method()`

```
.append()    πρόσθεση στοιχείου στο τέλος της λίστας
.clear()     άδεισμα λίστας - επιστροφή κενής
.copy()      αντιγραφή της λίστας (shallow copies)
.count()     πλήθος εμφανίσεων ενός στοιχείου
.extend()    προέκταση λίστας
.index()     ο δείκτης όπου εμφανίζεται για 1η φορά ένα στοιχείο
.insert()    πρόσθεση στοιχείου σε συγκεκριμένη θέση
.pop()       αφαίρεση στοιχείου από συγκεκριμένη θέση
.remove()    αφαίρεση όλων των στοιχείων με ορισμένη τιμή
.reverse()   αντιστροφή λίστας

.sort()                 διάταξη σε αύξουσα σειρά
.sort(reverse=True)     διάταξη σε φθίνουσα σειρά
```

! Η sorted() επιστρέφει νέα λίστα, όπως και όλες οι συναρτήσεις
! Η .sort() αλλάζει την υπάρχουσα, όπως και όλες (σχεδόν) οι μέθοδοι

Παραδείγματα

```
words=['Xyz','aBc','ABb']
sorted(words)
------------------------
['Abb', 'aBc', 'Xyz']        #πρώτα τα κεφαλαία
```

```
words=['Xyz','aBc','ABb']
print(sorted(words, key=str.lower))
-----------------------------------
['ABb','aBc','Xyz']
```

Η `key=str.lower` αγνοεί τη διάκριση πεζών-κεφαλαίων, μετατρέποντάς τα όλα σε πεζά, χωρίς να μεταφέρει την αλλαγή στο τέλος

```
words=['Xyz','aBc','ABb']
words.sort(key=str.lower)
print(words)
--------------------------------
['ABb','aBc','Xyz']
```

```
words=['apple','banana','ananas','melon']
print(words.sort(key=len))
-----------------------------------------
['apple', 'melon', 'ananas', 'banana']
```

Με την `key=len` γίνεται διάταξη δίνοντας προτεραιότητα στο μήκος των στοιχείων

Πρόσθεση Στοιχείων σε Λίστα

Έστω λίστα L.

L.insert(i,x) Προσθέτει το στοιχείο x μια θέση αριστερότερα της θέσης i.

! Αν i=0, τότε το προσθέτει στην αρχή της λίστας
! Δουλεύει και με αρνητικούς δείκτες

L.append(x) Προσθέτει το στοιχείο x στο τέλος της λίστας.

L.extend(x) Δουλεύει **μόνο αν το x είναι iterable.**
Τότε, "σπάει" το x σε άτομα και προεκτείνει την L με αυτά.

```
L=[1,2,3]
L+='spam'
-------------------------
L=[1,2,3,'s','p','a','m']
```

Αφαίρεση Στοιχείων από Λίστα

Έστω λίστα L.

L.clear() Επιστρέφει κενή λίστα

L.pop() Αφαιρεί το τελευταίο στοιχείο της λίστας

! Η εντολή **element=L.pop()** εκχωρεί το αφαιρεθέν στοιχείο στη μεταβλητή element

L.pop(i) Αφαιρεί το στοιχείο στη θέση i

L.remove(x) Αφαιρεί όλα τα στοιχεία με τιμή x

! Δεν αφαιρεί στοιχεία με βάση τη θέση τους, αλλά με βάση την τιμή τους

del L[i] Διαγράφει το στοιχείο στη θέση i

ΙΣΟΤΗΤΑ VS ΤΑΥΤΙΣΗ

```
L=[1,2,3]
M=L
L[2]=0
print(M)
---------
[1,0.3]
```

Δηλαδή αν αλλάξω τη λίστα L, αλλάζει ανάλογα και η λίστα M, παρόλο που η εκχώρηση M=L έγινε πριν την αλλαγή. Αυτό συμβαίνει, επειδή οι δύο λίστες αναφέρονται στο ίδιο αντικείμενο και η αλλαγή γίνεται σε αυτό!

```
L=[1,2,3]
M=L
L='spam'
print(M)
---------
[1,2,3]
```

Εδώ που εκχωρούμε str, δε συμβαίνει το παραπάνω, διότι οι str είναι αμετάβλητοι τύποι δεδομένων!

! ΓΙΑ ΝΑ ΕΞΑΣΦΑΛΙΣΟΥΜΕ ΟΤΙ ΔΥΟ ΛΙΣΤΕΣ ΑΝΑΦΕΡΟΝΤΑΙ ΣΕ ΔΥΟ ΔΙΑΦΟΡΕΤΙΚΑ ΑΝΤΙΚΕΙΜΕΝΑ ΜΕ ΤΟ ΙΔΙΟ ΠΕΡΙΕΧΟΜΕΝΟ, ΔΗΜΙΟΥΡΓΟΥΜΕ ΑΝΤΙΓΡΑΦΑ

1) Με τεμαχισμό:

```
L0=[1,2,3]
L1=L0[:]
print(L1==L0, L1 is L0)
-----------------------
(True, False)
```

2) Με τη list:

```
L0=[1,2,3]
L1=list(L0)
print(L1==L0, L1 is L0)
-----------------------
(True, False)
```

3) Με την copy:

```
L0=[1,2,3]
L1=L0.copy()
print(L1==L0, L1 is L0)
-----------------------
(True, False)
```

! Η copy δημιουργεί shallow copies (one level deep copying). Δηλαδή αντιγράφει το αρχικό αντικείμενο, αλλά όχι τα αντικείμενα τα οποία περιέχει. Το shallow copy περιέχει απλώς αναφορές σε αυτά τα αντικείμενα, οπότε, αν αλλάξουν εκτός, τότε αλλάζουν και εντός του shallow copy.

```
Y=['spam', [1,2,3], 42]
Z=Y.copy()
print(Y==Z, Y is Z)
-----------------------
(True, False)
```

```
Y=['spam', [1,2,3], 42]
Z=Y.copy()
Y[1][0]=99999
print(Z)
-------------------------
['spam', [99999,2,3], 42]
```

Η αλλαγή μεταφέρθηκε στην Z

Εντολές επανάληψης σε Λίστες

1) Με τη for και χρήση δεικτών:

```
L=list(range,10,16))
S=0
for i in range(len(L)):
    S+=L[i]**2
print(S)
-------------------------------
955 #το άθροισμα των τετραγώνων
```

! Ο τελεστής += προσαυξάνει ό,τι είναι αριστερά κατά αυτό που είναι δεξιά

2) Χωρίς χρήση δεικτών:

```
L=list(range(10,16))
p=1
for x in L:
p*=x
print(p)
--------------------
3603600
```

! Ο τελεστής *= πολλαπλασιάζει ό,τι είναι αριστερά με αυτό που είναι δεξιά

Αν δεν έχουμε ανάγκη από δείκτες, προτιμούμε το δεύτερο τρόπο.

TUPLES

Δημιουργία

Με παρενθέσεις: `(1,2,3,'spam',[4,5,6])`
Χωρίς παρενθέσεις: `P=1,2,3,'spam',[4,5,6]`

Κενή Tuple: `Empty_Tuple=()`

Tuple με 1 στοιχείο: `One_Element_Tuple=(10,)`

Το , είναι απαραίτητο για να δημιουργήσουμε tuple με 1 στοιχείο, αλλιώς θα ερμηνευτεί ως int

Μετατροπή Τύπων

Με τη συνάρτηση **tuple**

```
tuple('abcdef')
------------------------
('a','b','c','d','e','f')
```

! **Οι πλειάδες είναι διατεταγμένες ως προς τη θέση.**
Πρόκειται ουσιαστικά για εξειδικευμένο τύπο λιστών.

Διαχείριση με Δείκτες

Όπως και στους τύπους str και list.

! Σε αντίθεση με τις λίστες, οι πλειάδες είναι αμετάβλητοι τύποι. Δεν μπορώ να μεταβάλω τα στοιχεία μιας πλειάδας, ούτε το πλήθος τους

! Οι πράξεις με πλειάδες δημιουργούν νέα αντικείμενα

Πράξεις

Με τους τελεστές + και * κατά τα γνωστά

Έλεγχος Μέλους

Με τον τελεστή in

```
T='ham',2,45.6
print('ham' in T)
-----------------
True
```

Συναρτήσεις για πλειάδες

Ίδιες με εκείνες των λιστών (len, max, min, sum, ...)

! Η sorted επιστρέφει πάντα λίστα. Αν θέλουμε διατεταγμένη πλειάδα πρέπει να κάνουμε μετατροπή εκ νέου

```
P=3,1,2
SP=tuple(sorted(P))
```

Μέθοδοι για πλειάδες

Μας ενδιαφέρουν οι **count** και **index**

Επαναληπτικές Διαδικασίες

Όπως και στις λίστες

```
T=1,2,3
p=1
for x in T:
    p*=x
print(p)
----------
6
```

! Το πλεονέκτημα των πλειάδων είναι ότι δημιουργούν αντίγραφα με κάθε αλλαγή, κρατώντας τα αρχικά αντικείμενα αμετάβλητα. Συνεπώς, παρέχουν ασφάλεια όταν έχουμε κοινές αναφορές. Επίσης, έχουν μεγαλύτερη ταχύτητα εκτέλεσης κώδικα.

ΕΠΑΝΑΛΗΨΗ ΣΤΗΝ PYTHON

Τα αντικείμενα **list**, **str**, **range** είναι κάποια είδη αντικειμένων της Python που είναι **iterable**, δηλαδή **ακολουθούν το πρωτόκολλο επανάληψης** (μπορούν να δώσουν τα στοιχείο τους ένα ένα όταν τους ζητηθούν.

! Για να διατρέξουμε επαναληπτικά ένα iterable αντικείμενο, πρέπει συνήθως να δημιουργήσουμε πρώτα έναν iterator για αυτό, μέσω τη συνάρτησης iter

```
L=[0,1,3.14,2.71]
it=iter(L)
print(type(it))
-----------------
list_iterator
```

```
s='Python'
x=iter(s)
print(type(x))
--------------
str_iterator
```

```
R=range(5)
I=iter(R)
print(type(I))
--------------
range_iterator
```

Ας κρατήσουε τις μεταβλητές L=[0,1,3.14,2,71], s='Python' και R=range(5) και τα αντικείμενα it=iter(L), x=iter(s), I=iter(R) για περαιτέρω χρήση.

```
print(next(x))
--------------
P
```

Υπάρχει η συνάρτηση next που μας διαθέτει τα στοιχεία ένα-ένα τη φορά

```
print(next(x))
--------------
y
```

```
print(next(x),next(x))
----------------------
t h
```

! Όταν εξαντληθούν τα στοιχεία, προκαλείται η εξαίρεση StopIteration και τελειώνουν οι επαναλήψεις

LIST SHORTCUTS

Είναι ένας σύντομος και συμπαγής **τρόπος κατασκευής μιας νέας λίστας** από τα στοιχεία κάποιου **iterable** αντικειμένου.

```
L=[1,2,3,4,5]
res=[]
for x in L:
    res.append(x+10)
print(res)
--------------------
[11,12,13,14,15]
```

```
L=[1,2,3,4,5]
res=[x+10 for x in L]
---------------------
[11,12,13,14,15]
```

Δουλεύει μόνο για iterable objects

Χρήσιμα Shortcuts (Συμπεριλήψεις)

```
L=[1,2,3,4,5]
res=[x+10 for x in L[:3]]
odd_squares=[x**2 for x in res if x%2==1]
print(odd_squares)
-----------------------------------------
[121,169]
```

Με for και if

```
L=[x+y for x in 'abc' for y in '123']
print(L)
-----------------------------------------------
['a1','a2','a3','b1','b2','b3','c1','c2','c3']
```

Κατασκευή του πίνακα A = | 0 1 2 3 | με shortcut:
| 4 5 6 7 |
| 8 9 10 11 |

```
m=3
n=4
i=1
r2=[n*i+j for j in range(n)] #κατασκευή 2ης γραμμής
A=[[n*i+j for j in range(n)] for i in range(m)] #θέλουμε m γραμμές
print(A, A[2], [A[i][2] for i in range(3)], sep='\n')
-----------------------------------------------------------------------
[[0,1,2,3],[4,5,6,7],[8,9,10,11]]
[8,9,10,11]   #3η γραμμή
[2,6,10]      #3η στήλη
```

Γενική Σημείωση: Για να γράψουμε εντολή σε πάνω από μία γραμμές, ανοίγουμε παρένθεση ή αγκύλη στην πρώτη γραμμή, την οποία δεν κλείνουμε, παρά μόνο στην τελευταία γραμμή την εντολής, όταν αυτή ολοκληρωθεί. Αυτό η Python το "καταλαβαίνει" και δεν προχωράει παρακάτω αν δεν κλείσει η παρένθεση που ανοίξαμε.

FUNCTIONS

Έχουμε 3 είδη: i) **Built-in** (π.χ. sorted())
ii) **Defined in Modules** (π.χ. math.sqrt())
iii) **User defined**

Ορισμός

```
def name(parameter1,...,parameterN):
    statements
    return value
```

Η return μπορεί και να μην υπάρχει, δηλαδή η συνάρτησή μας να μην επιστρέφει τίποτα

Κλήση

```
result=name(argument1,...,argumentN)
```

Πολύ σπάνιο, αλλά υπάρχουν και συναρτήσεις που ούτε παραμέτρους έχουν ούτε επιστρέφουν κάτι

! **Συναρτήσεις χωρίς return επιστρέφουν την ειδική τιμή None.**
H print είναι μια τέτοια συνάρτηση:

```
res=print('Hello')   #εδώ τυπώνει Hello
print(res)   #η print της print δηλαδή
------------------
Hello
None
```

! **Πολλές επιστρεφόμενες τιμές:**

```
def everything(data):
    m=min(data)
    M=max(data)
    mo=sum(data)/len(data)
    return m,M,mo   #Επιστρέφει μια tuple, την (m,M,mo)
items=[1,2,3,4,5]
a,b,c=everything(items)
print('Minimum is ', a, '\nMaximum is ', b, '\nMean value is ', c)
----------------------------------------------------------------
Minimum is  1
Maximum is  5
Mean value is  3.0
```

! ΚΑΤΑ ΤΗΝ ΚΛΗΣΗ ΜΙΑΣ ΣΥΝΑΡΤΗΣΗΣ, ΤΑ ΟΡΙΣΜΑΤΑ (ARGUMENTS) ΕΙΝΑΙ ΑΝΑΦΟΡΕΣ ΣΕ ΑΝΤΙΚΕΙΜΕΝΑ ΤΗΣ PYTHON

Αν τα ορίσματα είναι **αμετάβλητου τύπου** (int, str κλπ), τότε οι αλλαγές τους μέσα στη συνάρτηση **δεν επηρεάζουν** τις τιμές κλήσης.

Αν τα ορίσματα είναι **μεταβλητού τύπου** (lists, dictionaries κλπ), τότε αλλαγές τους μέσα στη συνάρτηση **επηρεάζουν** τις τιμές κλήσης.

```
def func(x,L1,L2):
    print('\n***In func before changes***')
    print(x)
    print(L1)
    print(L2)
    x=2*x
    L1=L1*2
    for i in range(len(L2)):
        L2[i]=L2[i]***2
    print('\n***In func after changes***')
    print(x)
    print(L1)
    print(L2)   #τέλος συναρτήσης
t=500
P1=[11,12,13]
P2=[10,20,30]
func(t,P1,P2)
print('\n***What has happened***')
print(t)
print(P1)
print(P2)
```

```
-----------------------------------------
***In func before changes***
500
[11, 12, 13]
[10, 20, 30]

***In func after changes***
1000
[11, 12, 13, 11, 12, 13]
[100, 400, 900]

*** What has happened***
500
[11, 12, 13]
[100, 400, 900]
```

Δηλαδή οι αλλαγές στα αντικείμενα εντός της συνάρτησης παραμένουν και μετά την κλήση της!

Οι Συναρτήσεις ως Αντικείμενα

- **Μπορούν να εκχωρηθούν σε μεταβλητές:**

```
a=times(20,50)
print(a)
--------------
1000

mult=times
b=mult(2,5)
--------------
10
```

- **Μπορούν να είναι ορίσματα άλλων συναρτήσεων:**

```
def deriv(f,x,h):
    return(f(x+h)-f(x))/h
def g(x):
    return 2*x**2-3*x+5
fd=deriv(g,1,0.01) #Η παράγωγος g'(1)=1 με ακρίβεια h=0.01
import math
sqd=deriv(math.sqrt, 4, 1e-4)
print(fd)
print(sqd)
-----------------------------------------------------
1.019999999999932
0.2499984375203823
```

! Κάποιες παράμετροι μιας συνάρτησης μπορεί να έχουν default τιμές, τις οποίες θα πάρουν αυτόματα αν παραλείψουμε να δώσουμε όρισμα κατά την κλήση της συνάρτησης

```
def deriv(f,x,h=1e-4):
    return (f(x+h)-f(x))/h
#Η τιμή h=1e-4 είναι πλέον default και θα
#χρησιμοποιηθεί αν καλέσω την deriv με 2 ορίσματα
def g(x):
    return 2*x**2-3*x+5
deriv(g,1)
-------------------------------------------------
1.000199999996454
```

Modules

Οι συναρτήσεις γράφονται σε αρχεία. Το σύνολο των συναρτήσεων ενός αρχείου μαζί με τις άλλες εντολές που περιέχονται σε αυτό αποτελούν ένα **module**. Ένα module περιέχει συνήθως τις συναρτήσεις που χρειάζεται ένα πρόγραμμα για να τρέξει και συναρτήσεις με κοινό αντικείμενο. Έχει την έννοια της *βιβλιοθήκης* και αποτελεί ένα *namespace*.

! Η εκτέλεση ενός προγράμματος ξεκινάει από μία κύρια συνάρτηση, που παραδοσιακά ονομάζεται *main*.

Δημιουργία & χρήση Module

Έστω το αρχείο **calculus.py** που περιέχει την **deriv** όπως ορίστηκε παραπάνω, και μία συνάρτηση main. Αυτό αποτελεί ένα module.

```
def deriv(f,x,h=1-4):
    return(f(x+h)-f(x)/h)
def p(x):
    y=x**2-5*x+2
    return y
def main():
    x=float(input('X = '))
    y=p(x)
    dp=deriv(p,x)
    print('P(', x, ') =', y)
    print('DP(', x, ') =', dp)
main()
-------------------------------
X = 10 #εκχωρώ X=10
P( 10.0 ) = 52.0
DP( 10.0 ) = 33.33333333333333
```

! Οι μεταβλητές x,y που ορίζονται στην g είναι τοπικές μεταβλητές για τη g και διαφορετικές από τις αντίστοιχες τοπικές μεταβλητές της main

```
#Πολυώνυμο P βαθμού n με συντελεστές στη λίστα coeff=[cn,…,c0]
def p(x,coeff):
    value=0
    xpower=1
    for c in coeff[::-1]:
        value+=c*xpower
        xpower*=x
    return value
#Μπορώ να καλέσω την p με ορισμένο x και λίστα συντελεστών
```

! Τώρα το πρόβλημα είναι ότι δεν μπορώ να χρησιμοποιήσω την deriv, η οποία περιμένει η συνάρτηση f να έχει μόνο μία παράμετρο

! Η λύση είναι να βγάλουμε έξω την coeff, δηλαδή να την κάνουμε καθολική μεταβλητή

! Οι τοπικές μεταβλητές δεν είναι ορατές εκτός της συνάρτησης. Αντίθετα, οι καθολικές ορίζονται εκτός των ορίων τής συνάρτησης και είναι ορατές και εντός της

```
def deriv(f,x,h=1e-4):
     return (f(x+h)-f(x))/h

COEFF=[]

def p(x):
    value=0
    xpower=1
    for c in COEFF[::-1]:
        value+=c*xpower
        xpower*=x
    return value

def main():
    n=int(input('Degree of polynomial? '))
    COEFF.clear()
    for i in range(n,-1,-1):
        message='C'+str(i)+'? '
        c=float(input(message))
        COEFF.append(c)

main()

x=float(input('X = '))
y=p(x)
dp=deriv(p, x)
print('P(',x,') =',y)
print('DP(',x,') =',dp)
-----------------------------------------------
C4? 2
C3? 3
C2? -1
C1? 2
C0? 8
X = 3
P( 3.0 ) = 248.0
DP( 3.0 ) = 293.01340027075184
```

SETS

- Πρόκειται για συλλογή στοιχείων **διακριτών** μεταξύ τους και **όχι διατεταγμένων** ως προς τη θέση τους.

- Μπορεί να περιέχουν στοιχεία **διαφορετικού τύπου**.

! **Πρέπει να έχουν στοιχεία αμετάβλητου τύπου** (π.χ. int, float, str, tuple). **Αν προσπαθήσω να φτιάξω set που να περιέχει στοιχείο μεταβλητού τύπου (π.χ. list), τότε θα προκύψει error.**

Δημιουργία Συνόλων

Με άγκιστρα: {42,'spam',3.14, (1.5, 2)}

! Από τη στιγμή που τα στοιχεία εντός των αγκίστρων δεν είναι διατεταγμένα, η Python θα τα αποθηκεύσει με τον πιο αποδοτικό γι αυτήν τρόπο και όχι απαραίτητα με τη σειρά με την οποία τα γράψαμε.

Μετατροπή τύπων

```
S1,S2,S3=set('abcdef'), set([10,20,30,40]), set(range(5))
print(S1,S2,S3)
-----------------------------------------------------------
{‘d','c','a','e','b','f'} {40, 10, 20, 30} {0,1,2,3,4}
```

Κενό Set

```
Empty_Set=set()
```

Δημιουργείται με τη συνάρτηση set() και όχι με { }

Συναρτήσεις για Sets

```
len()
min()
max()
sum()
```

/ Μπορούν να χρησιμοποιηθούν μόνο αν
\ τα στοιχεία του συνόλου είναι συγκρίσιμα
Μόνο αν ορίζεται άθροισμα

Μέθοδοι για Sets

.add()	πρόσθεση στοιχείου
.clear()	άδειασμα συνόλου
.copy()	αντιγράφή συνόλου
.isdisjoint()	έχουν κοινά στοιχεία;
.issubset()	είναι υποσύνολο;
.issuperset()	είναι υπερσύνολο;
.pop()	αυθαίρετη αφαίρεση στοιχείου
.remove()	αφαίρεση συγκεκριμένου στοιχείου

```
.discard()                          αφαίρεση συγκεκριμένου στοιχείου
.union()                            ένωση συνόλων
.union_update()                     ένωση συνόλων και update
.intersection()                     τομή συνόλων
.intersectioon_update()             τομή συνόλων και update
.difference()                       διαφορά συνόλων
.difference_update()                διαφορά συνόλων και update
.symmetric_difference()             συμμετρική διαφορά συνόλων
.symmetric_difference_update()      συμμ. διαφορά συνόλων και update
```

```
S={1,2,3,'spam'}
S.add('abc')
print(S)
-------------------------
{1,2,3,'spam'}
```

```
S={1,2,3,'spam'}
S.discard('spam')
print(S)
-------------------------
{1,2,3}
```

```
S={1,2,3,'spam'}
S.remove('spam')
print(S)
-------------------------
{1,2,3}
```

! Ενώ η discard με την remove κάνουν την ίδια δουλειά, έχουν τη διαφορά ότι αν το στοιχείο που θέλουμε να αφαιρέσουμε δεν υπάρχει στο σύνολο, τότε η discard θα αγνοήσει τη μη-ύπαρξή του, ενώ η remove θα βγάλει error

```
S={1,2,3,'spam'}
el=S.pop()
print(el)
----------------
2
```

Η .pop() θα αφαιρέσει ΑΥΘΑΙΡΕΤΑ κάποιο στοιχείο.
Τα στοιχεία του συνόλου ΔΕΝ είναι διατεταγμένα.

```
S={1,2,3,'spam'}
T=S.copy()
print(T)
print(T is S)
----------------
{1,2,3,'spam'}
False
```

Έχουμε αντίγραφο της S
αλλά τα αντικείμενα δεν ταυτίζονται!

```
S={1,2,3,'spam'}
S.clear()
print(S)
----------------------
set() #το κενό σύνολο
```

```
S={1,2,3,'spam'}
IDJ={1,2,3}.isdisjoint(S)
print(IDJ)
-------------------------
False
```

```
P={1,2,3}
Q={1,2,3,'spam'}
SUB=P.issubset(Q)
print(SUB)
-----------------
True
```

```
P={1,2,3}
Q={1,2,3,'spam'}
SUP=P.issuperset(Q)
print(SUPER)
----------------------
False
```

Ούτε η .issubset() ούτε η .issuperset() μπορεί να ελέγξει τη γνησιότητα!

Λογικές Πράξεις

Με τους τελεστές in και == κατά τα γνωστά

```
P={1,2,3}
Q={1,2,3,5}
R={3,1,2}
print(10 in P)
print(P==Q)
print(P==R)
--------------
False
False
True
```

Πράξεις Συνολοθεωρίας

```
cs1={16101,17222,18102,19103}
cs2={19250,17222}
U=cs1|cs2  #ένωση
I=cs1&cs2  #τομή
D=cs1-cs2  #διαφορά
S=cs1^cs2  #συμμετροδιαφορά
print(U,I,D,S,sep='\n')
------------------------
{19250, 16101, 17222, 18102, 19103}
{17222}
{16101, 18102, 19103}
{19250, 16101, 18102, 19103}
```

! Οι παραπάνω πράξεις υλοποιούνται και με τη χρήση μεθόδων

```
cs1={16101,17222,18102,19103}
cs2={19250,17222}
new_cs2=cs2.union([19001,19002])  Θα μετατρέψει τη list σε set και θα ενώσει
print(new_cs2)
print(cs2)
--------------------------------
{19001, 19250, 19002, 17222}
{19250,17222}  Το αρχικό μας σύνολο δεν άλλαξε, παρόλο που του εφαρμόσαμε μέθοδο!
```

Για την τομή, διαφορά και συμμετροδιαφορά θα εφαρμόσουμε τις μεθόδους .intersection(), .difference() και .difference_symmetric() αντίστοιχα.

! Ούτε και αυτές θα αλλάξουν το αρχικό μας σύνολο

! Αν θέλουμε η αλλαγή να μεταφερθεί στο αρχικό σύνολο θα πρέπει να χρησιμοποιήσουμε τη μέθοδο γράφοντας _update

Δηλαδή: `.union_update()`
`.intersection_update()`
`.difference_update()`
`.difference_symmetric_update()`

```
cs1={16101,17222,18102,19103}
cs1.intersection_update({17222,19103})
print(cs1)
-------------------------
{17222,19103}
```

Χρήση Συνόλων σε εφαρμογές

- Φιλτράρισμα επαναλαμβανόμενων τιμών σε μία αλληλουχία

```
L=[2,2,3,3,1,1,1,3,3,3,2]
L=list(set(S))
print(L)
-------------------------
[1,2,3]   Η διάταξη άλλαξε!
```

Επαναληπτική επεξεργασία Συνόλων

Προφανώς χωρίς δείκτες!

```
n=int(input('How many elements? '))
S=set()
for i in range(n):
    x=input('Next element? ')
    S.add(x)
print(S)
-----------------------------------
How many elements? 3
Next element? Python
Next element? JavaScript
Next element? SQL
{'Python','SQL','JavaScript'}
```

DICTIONARIES

- Είναι μια **μη-διατεταγμένη** συλλογή αντικειμένων

- Η αποθήκευση γίνεται με **κλειδί αντί θέσης**

- Το κλειδί χρησιμοποιείται για την ανάκτηση του αποθηκευμένου αντικειμένου

- Τα στοιχεία του λεξικού είναι **ζεύγη κλειδιών/τιμών**

- Ο τύπος του λεξικού είναι **dict**

Δημιουργία Λεξικού

```
phone={"John":'6971234567','Mary':"6981122333",'Jax':'6947766555'}
print(phone['Jax'])
-------------------------------------------------------------------
6947766555
```

! Τα λεξικά είναι **απεικονίσεις**. Απεικονίζουν **κλειδιά σε τιμές.**

key:value

! **Τα κλειδιά του λεξικού πρέπει να είναι αμετάβλητου τύπου (str, int, float, tuple)**

! Η αναζήτηση τιμών ενός λεξικού είναι πιο γρήγορη από την αναζήτηση με θέση, όπως π.χ. στις λίστες:

```
names=['John','Mary','Jax']
numbers=['6971234567','6981122333','6947766555']
number[names.index('Jax')]
-----------------------------------------------
'6947766555'
```

! Τα στοιχεία ενός λεξικού μπορεί να είναι **διαφορετικού τύπου**

! Λόγω της μη-διάταξης των στοιχείων τους, τα λεξικά **δεν υποστηρίζουν πράξεις τεμαχισμού και συνένωσης**

! Τα λεξικά είναι **μεταβλητός τύπος δεδομένων: οι τιμές των στοιχείων τους μπορούν να αλλάξουν**

```
student={'name':'John','age':20,'mo':8.2}
student['mo']=9
print(student)
---------------------------------------
{'name':'John','age':20,'mo':9}
```

! Είναι **μεταβλητού μήκους: μπορούμε να εισάγουμε νέα στοιχεία στο τέλος** του λεξικού

```
student={'name':'John','age':20,'mo':8.2}
student['phone']='6971234567'   Δημιουργεί νέο κλειδί!
print(student)
-----------------------------------------------------
{'name':'John','age':20,'mo':8.2,'phone':'6971234567'}
```

Συναρτήσεις για Λεξικά

! Δύο λεξικά είναι ίσα αν και μόνο αν έχουν τα ίδια ζεύγη κλειδιών/τιμών, **ανεξάρτητα από τη σειρά αποθήκευσής τους**

```
D1={'a':1,'b':2}
D2={'b':2,'a':1}
print(D1==D2)
print(D1!=D2)
print(D1 is D2)
----------------
True
False
False
```

! Έλεγχος μέλους με χρήση του **κλειδιού**

```
D1={'a':1,'b':2}
print('a' in D1)      ΔΕΝ ΔΟΥΛΕΥΕΙ ΜΕ ΤΙΜΕΣ
----------------
True
```

Επανάληψη σε Λεξικά

```
ages=dict(Bob=20,Mary=31,Ann=20,John=42)
print(ages)
----------------------------------------
{'Bob':20, 'Mary':31,'Ann':20,'John':42}

ages=dict(Bob=20,Mary=31,Ann=20,John=42)

for k in ages:   #ισοδύναμα: ages.keys()
    print(k,'-->', ages[k])
----------------------------------------
Bob --> 20
Mary --> 31
Ann --> 20
John --> 42
```

```
ages=dict(Bob=20,Mary=31,Ann=20,John=42)
for k in sorted(ages):    #διατάσσει τα κλειδιά
    print(k,'-->', ages[k])
-----------------------------------------------
Ann --> 20
Bob --> 20
John --> 42
Mary --> 31


ages=dict(Bob=20,Mary=31,Ann=20,John=42)
for (name,age) in ages.items():  Παράλληλη επανάληψη με κλειδιά και τιμές
    print(name,'is',age,'years old.')
----------------------------------------
Bob is 20 years old.
Mary is 31 years old.
Ann is 20 years old.
John is 42 years old.
```

Λεξικά με στοιχεία αλληλουχίες, αλληλουχίες με στοιχεία λεξικά, λεξικά με στοιχεία λεξικά

```
employee={'name':'Bob','salary':1000,'age':25,'company':'ACME'}
employee['job']=['Programmer','Technician']
print(employee)
-------------------------------------------------------------------
{'name': 'Bob', 'salary': 1000, 'age': 25, 'company': 'ACME',
'job': ['Programmer', 'Technician']}


employee={'name':'Bob','salary':1000,'age':25,'company':'ACME'}
employee['job']=['Programmer','Technician']
(print(employee['name'],'has',len(employee['job']),'jobs:',
employee['job'][0], 'and',employee['job'][1]))
-----------------------------------------------------------
Bob has 2 jobs: Programmer and Technician


student_db=[]
for i in range(2):
    rec={}
    rec['name']=input('Student name? ')
    info={}
    info['year']=input('Year of study? ')
    info['AM']=input('AM? ')
    rec['details']=info
    student_db.append(rec)
print(student_db)
----------------------------------------
Student name? John
Year of study? 2024
AM? 1500067
```

```
Student name? Mary
Year of study? 2024
AM? 1800032
[{'name': 'John', 'details': {'year': '2024', 'AM': '1500067'}},
{'name': 'Mary', 'details': {'year': '2024', 'AM': '1800032'}}]
```

! Τα λεξικά προτιμώνται σε σχέση με τις λίστες για:
  i) δεδομένα που προσδιορίζονται με κάποιο **όνομα ή ετικέτα** (πιο γρήγορη αναζήτηση)
 ii) **αραιές δομές** (π.χ. πίνακες με πολλά μηδενικά)
iii) συλλογές αντικειμένων που αναπτύσσονται σε **τυχαίες θέσεις**

Παραδείγματα:

Πίνακες με πολλά μηδενικά - **Τα μηδενικά δεν αποθηκεύονται!**
Υπολογισμός Γινομένου αραιού πίνακα με διάνυσμα: y=Ax

```
                    |2 0 0 1|                    |y0|
          y=Ax=     |0 1 0 0|. (10 12 15 8)' =   |y1|
                    |0 0 4 5|                    |y2|
```

```
A={(0,0):2,(0,3):1,(1,1):1,(2,3):5,(2,2):4} #πετάμε τα μηδενικά
x=[10,12,15,8]
y=[0]*3
for ij in A: #τρέχει τα κλειδιά
    y[ij[0]]+=A[ij]*x[ij[1]]
print(y)
```

```
[28, 12, 100]
```

```
old={'a':1,'b':2,'c':3,'A':1,'B':2,'C':-1,'xyz':'abc'}
new={}
for k in old:
    if old[k] not in new:
        new[old[k]]=[k]
    else:
        new[old[k]].append(k)
print(new)
```

```
{1:['a','A'],2:['b','B'],3:['c'],-1:['C'],'abc': ['xyz']}
```

EXCEPTIONS

Πρόκειται για **σφάλματα** που ανιχνεύονται κατά την **εκτέλεση** ενός προγράμματος και τα οποία μπορούμε να **χειριστούμε**

Κάθε εξαίρεση έχει ένα **προκαθορισμένο μήνυμα** για το σφάλμα το οποίο την προκαλεί

Θα ασχοληθούμε με τις εξής εξαιρέσεις: **ZeroDivisionError**
IndexError
NameError
ValueError

Exception Handling

Όταν συμβεί μια εξαίρεση και δεν τη χειριστούμε, η εκτέλεση του προγράμματος τερματίζεται

```
passed=int(input('Students passed? '))
failed=int(input('Students failed? '))
ratio=passed/failed
print('Passed/Failed ratio is ',ratio)
print('Number of students tested is ', passed+failed)
```

```
Students passed? 100
Students failed?  0
Exception has occurred: ZeroDivisionError
```

! Μπορώ να χειριστώ μια εξαίρεση με τις εντολές try και except, τις οποίες χρησιμοποιώ τροποποιώντας τον υπάρχοντα κώδικα, ώστε να μεριμνήσω για τυχόν σφάλματα, όπου αυτά είναι πιθανά

```
#Exception Handling του παραπάνω κώδικα:
passed=int(input('Students passed? '))
failed=int(input('Students failed? '))
try:
    ratio=passed/failed
    print('Passed/Failed ratio is ',ratio)
except ZeroDivisionError:
    print('No failures!')
    print('Number of students tested is ',passed+failed)
```

```
Students passed? 100
Students failed? 0
No failures!
Number of students tested is  100
```

```
val1=int(input('Enter an integer: '))
print('The square of the number is ',val1**2)
-----------------------------------------
Enter an integer: spam
Exception has occurred:ValueError
```

```
#Exception Handling του παραπάνω κώδικα:
while True:
    val=input('Enter an integer: ')
    try:
        val=int(val)
        print('The square of the number is',val**2)
        break
    except ValueError:
        print(val,' is not an integer')
-------------------------------------------------
Enter an integer: spam
spam  is not an integer
Enter an integer: ok
ok  is not an integer
Enter an integer: yolo
yolo  is not an integer
Enter an integer: 8
The square of the number is 64
```

! Σε γενικότερη μορφή ως συνάρτηση εισόδου για διαφορετικούς τύπους δεδομένων:

```
def read_value(val_type,in_message,err_message):
    while True:
        val=input(in_message)
    try:
        return val_type(val)
    except ValueError:
        print(err_message)

#κλήση της παραπάνω συνάρτησης:
read_value(float,'Enter a real number: ','Not a real number')
-------------------------------------------------------------
Enter a real number: spam
Not a real number
Enter a real number: 100
```

! Μπορούμε να πιάσουμε πολλές εξαιρέσεις με μία try, δηλαδή να έχουμε πολλούς κλάδους except

! **Στη γενική μορφή της, η try με κλάδους except, else και finally, είναι η εξής:**

```
try:
    statements_to_test_for_exception
except ExceptionName1:
    statements_to_execute_when_ExceptionName1_is_raised
except ExceptionName2:
    statements_to_execute_when_ExceptionName2_is_raised
    ...
except ExceptionNameN:
    statements_to_execute_when_ExceptionNameN_is_raised
except:
    statements_to_execute_when_any_OTHER_exception_is_raised
else:
    statements_to_execute_when_NO_exception_is_raised
finally:
    statements_to_execute_ALWAYS  Ο κλάδος finally εκτελείται πάντα,
other_statements                  ανεξάρτητα από το αν προκλήθηκε εξαίρεση ή όχι
```

Πρόκληση εξαίρεσης

```
#Με τη συνάρτηση raise:
def get_month():
    month=int(input('Enter current month (1-12): '))
    if month<1 or month>12:
       raise ValueError('Invalid month value')
get_month()
-----------------------------------------------------
Enter current month (1-12): -1
Invalid month value   #τυπώνει το μήνυμα που θέσαμε


def get_month():
    month=int(input('Enter current month (1-12): '))
    if month<1 or month>12:
       raise ValueError('Invalid month value')
try:
    month=get_month()
except ValueError as msg:
    print(msg)
-----------------------------------------------------
Enter current month (1-12): 6.0        Η input δέχεται την str '6.0' και όχι τον float 6.0, άρα
invalid literal for int() with base 10: '6.0'  δεν μπορεί να την κάνει int, κι έτσι τυπώνει το
                                       προκαθορισμένο μήνυμα, αφού εμείς έχουμε
                                       προνοήσει μόνο για int < ή > και όχι για invalid type
```

! **Αν μια συνάρτηση δεν χειρίζεται μια εξαίρεση που προκαλείται σε αυτήν, μεταφέρει τον χειρισμό της στη συνάρτηση που την καλεί και είτε θα γίνει χειρισμός είτε θα crashαρει το πρόγραμμα.**

```
def A():
      print('>>> In A')
      print(5/0)
      print('>>> Leaving A')
def B():
      print('>>> In B')
      print('>>> Calling A')
      A()
      print('>>> Leaving B')
def C():
      print('>>> In C')
      print('>>> Calling B')
      B()
      print('>>> Leaving C')
C()
```

```
--------------------------------------
>>> In C
>>> Calling B
>>> In B
>>> Calling A
>>> In A
Exception has occurred: ZeroDivisionError
```

```
#ίδιο πρόγραμμα, αλλά με χειρισμό εξαίρεσης στην A
def A():
      print('>>> In A')
      try:
            print(5/0)
      except:  #αν προκληθεί οποιαδήποτε εξαίρεση, τότε:
            print('*** Caught exception in A')
      print('>>> Leaving A')
def B():
      print('>>> In B')
      print('>>> Calling A')
      A()
      print('>>> Leaving B')
def C():
      print('>>> In C')
      print('>>> Calling B')
      B()
      print('>>> Leaving C')
C()
```

```
------------------------------------------------------
>>> In C
>>> Calling B
>>> In B
>>> Calling A
>>> In A
*** Caught exception in A
>>> Leaving A
>>> Leaving B
>>> Leaving C
```

```
#ίδιο πρόγραμμα αλλά με χειρισμό εξαίρεσης στην B
def A():
      print('>>> In A')
      print(5/0)
      print('>>> Leaving A')
def B():
      print('>>> In B')
      print('>>> Calling A')
      try:
           A()
      except:
           print('*** Caught exception in B')
      print('>>> Leaving B')
def C():
      print('>>> In C')
      print('>>> Calling B')
      B()
      print('>>> Leaving C')
C()
```

```
---------------------------------
>>> In C
>>> Calling B
>>> In B
>>> Calling A
>>> In A
*** Caught exception in B
>>> Leaving B
>>> Leaving C
```

Η try ως εντολή ελέγχου:

```
def sum_digits(s):
    res=0
    for c in s:
        try:
            res+=int(c) #το int('a') προκαλεί εξαίρεση
        except:
            pass  #εντολή που δεν κάνει τίποτα - "απλά προχώρα"
    return res
print(sum_digits('aB1XX2cd4pP'))
```

```
-------------------------------------------------------------------
7
```

FILES

- Αποθηκεύουν δεδομένα για την είσοδο/έξοδο του σε προγράμματα Python

- Είναι built-in στην Python

Δημιουργία

Με την open: `file_object=open(filename,mode)`

Δημιουργούμε ένα αντικείμενο **file_object** για να διαχειριστούμε αυτό ένα **εξωτερικό αρχείο filename**.

! Η παράμετρος filename είναι προφανώς str - το όνομα του αρχείου που θέλω να διαχειριστώ

! Η παράμετρος mode είναι επίσης str και καθορίζει το είδος της επεξεργασίας. Μπορεί να πάρει 3 τιμές:
'r' for **read** (default value)
'w' for **write**
'a' for **append**

! Οτιδήποτε θέλουμε να εγγραφεί στο αρχείο, **πρέπει να είναι ήδη str**. Αλλιώς, η μορφοποίηση σε str δεν θα γίνει αυτόματα.

Η ΜΕΘΟΔΟΣ CLOSE

Την καλούμε όταν ολοκληρωθεί η επεξεργασία του αρχείου:
- **Τερματίζει τη σύνδεση** του προγράμματος με το εξωτερικό αρχείο στο δίσκο
- **Αποδίδει το αρχείο** στο λειτουργικό σύστημα
- **Μεταφέρει μη-καταχωρημένες εγγραφές** (λόγω buffering) από τη μνήμη στο δίσκο

```
f=open('myfile.txt','w')
f.close()
```

Η ΜΕΘΟΔΟΣ WRITE

```
f=open('myfile.txt','w')
f.write(5)
---------------------------------
Exception has occurred: TypeError
```

! Οτιδήποτε θέλω να εγγραφεί στο αρχείο μου πρέπει να είναι ήδη str. Η write δεν θα κάνει αυτόματη μορφοποίηση !

```
f=open('myfile.txt','w')
f.write('5')
```

<<< Ο σωστός τρόπος

```
f=open('myfile.txt','w')
f.write(str(5))
```

<<< Επίσης σωστό

! Τι επιστρέφει όμως η write;
Η write επιστρέφει το ΠΛΗΘΟΣ των δεδομένων που γράφω στο αρχείο.

```
f=open('myfile.txt','w')
print(f.write('5'))
-----------------------
1
```

```
f=open('myfile.txt','w')
for i in range(1,5):
    f.write('Line '+str(i))
f.close()
--------------------------
#Αν ανοίξω το αρχείο να το
#δω, θα έμφανιστεί το εξής:
Line 1Line 2Line 3Line 4
```

```
f=open('myfile.txt','w')
for i in range(1,5):
    f.write('Line '+str(i)+'\n')
f.close()
---------------------------------
#Αν ανοίξω το αρχείο να το
#δω, θα έμφανιστεί το εξής:
Line 1
Line 2
Line 3
Line 4
```

Η ΜΕΘΟΔΟΣ READ

```
f=open('myfile.txt')
s=f.read()
f.close()
print(s)
--------------------
Line 1
Line 2
Line 3
Line 4
```

Η ΜΕΘΟΔΟΣ READLINE

```
f=open('myfile.txt')
print(f.readline())
--------------------
Line 1 #η 1η γραμμή
```

```
f=open('myfile.txt')
for i in range(0,2)
    print(f.readline())
------------------------
Line 1 #τυπώνει σειριακά
Line 2 #τις γραμμές
```

```
f=open('myfile.txt')
for i in range(0,5)
    print(f.readline())
-----------------------
Line 1
Line 2
Line 3
Line 4
  #τυπώνει κενή str
```

! Η readline εσωτερικά αποθηκεύει την **str**. Δηλαδή για την 1η γραμμή αποθηκεύει **'Line 1\n'**, για τη 2η **'Line 2\n'** κ.ο.κ. Όταν εξαντληθούν οι γραμμές τότε έχει τιμή την **κενή str ''**.

```
f=open('myfile.txt')
for ln in f:
    print(ln[:-1],int(ln[-2])**3)   Το \n πιάνεται σαν ένας χαρακτήρας
f.close()
---------------------------------
Line 1 1
Line 2 8
Line 3 27
Line 4 64
```

Αρχεία και εξαιρέσεις

Η ανάγνωση και η εγγραφή μπορεί για διάφορους λόγους να προκαλέσουν εξαιρέσεις (το αρχείο να μην υπάρχει, να μην μπορεί να δημιουργηθεί κλπ). Μπορούμε να χειριστούμε **συνολικά** αυτές τις εξαιρέσεις με την **try** και το όνομα **OSError**.

Αποθήκευση δεδομένων σε Δυαδικά Αρχεία - Σειριοποίηση pickle

```
D={'name':'Bob','age':42}
L=[21,34,82,30,92,105]
F=open('datafile.pkl','wb') #γράφω b γιατι είναι δυαδικό αρχείο
import pickle    #άλλο ένα module
pickle.dump(D,F) #πέτα μέσα στο F το D
pickle.dump(L,F) #πέτα μέσα στο F την L
F.close()
F=open('datafile.pkl','rb') #άνοιξε το για ανάγνωση
DD=pickle.load(F) #φόρτωσε το 1ο στοιχείο που του πετάξαμε
LL=pickle.load(F) #φόρτωσε το επόμενο στοιχείο που του πετάξαμε
print(DD)
print(LL)
-------------------------------------------------------------
{'name':'Bob','age':42}
[21,34,82,30,92,105]
```

! Αν είχαμε βάλει άλλη μία εντολή pickle.load(...) θα είχαμε εξαίρεση **RAN OUT OF INPUT**, διότι το F δεν έχει άλλα στοιχεία.

- Ας υποθέσουμε ότι έχουμε το αρχείο grail.txt, όπως παρακάτω. Θα φτιάξουμε πρόγραμμα που θα τυπώνει το πλήθος γραμμών, λέξεων και χαρακτήρων του αρχείου αυτού.

```
print(open('grail.txt').read()) #πρώτα άνοιξε και μετά ανάγνωσε
----------------------------------------------------------------
We're Knights of the Round Table
We dance whene'er we are able
We do routines and chorus scenes        #περιέχει ένα τραγούδι
With footwork impeccable                #των Monty Python
We dine well here in Camelot
We eat ham and jam and spam a lot.
               -- Monty Python


def wc(filename):
    try:
        f=open(filename, 'r')
    except OSError as err:
        print(err)
    else:
        lines,words,chars=0,0,0
        for line in f:
            lines+=1
            words+=len(line.split())
            chars+=len(line)
    finally:
        f.close()
    return lines,words,chars
print(wc('grail.txt'))
-----------------------------------
(7, 39, 217)


#Κωδικοποίηση ROT13 (από το rotation 13)
def rot13(s):
    encs=''
    for c in s:
        n=ord(c)
        if 'a'<=c.lower()<='m':   #αν είναι στο [a,m], τότε βάλε
           encs+=chr(n+13)        #τον 13 θέσεις δεξιά chr
        elif 'n'<=c.lower()<='z': #αν είναι στο [n,z], τότε βάλε
           encs+=chr(n-13)        #τον 13 θέσεις αριστερά chr
        else:
           encs+=c
    return encs
print(rot13('Python 3.8'))
print(rot13(rot13('Python 3.8')))
----------------------------------------------------------------
Clguba 3.8 #κρυπτογραφημένα από τη rot13
Python 3.8 #αποκρυπτογραφημένο από τη rot13 της rot13
```

```
def main(): #έστω η παραπάνω rot13(s)
    infile=input('File to encode? ')
    outfile=input('Encoded file? ')
    try:
        fin=open(infile,'r')     #άνοιξέ το να το διαβάσω
        fout=open(outfile,'w')   #άνοιξέ το να το πειράξω
    except OSError as err:
        print(err)
    else:
        for i in fin:            #για κάθε χαρακτήρα
            fout.write(rot13(i)) #γράψε τον encoded του
    finally:
        fin.close()
        fout.close()
main()
```
```
-----------------------------------------------------------
File to encode? grail.txt
Encoded file? grail-enc.txt

#αν ανοίξουμε τώρα το αρχείο grail-enc.txt...
print(open('grail-enc.txt').read())
-------------------------------------------
Jr'er Xavtugf bs gur Ebhaq Gnoyr
Jr qnapr jurar're jr ner noyr
Jr qb ebhgvarf naq pubehf fprarf
Jvgu sbbgjbex vzcrppnoyr
Jr qvar jryy urer va Pnzrybg
Jr rng unz naq wnz naq fcnz n ybg.
                -- Zbagl Clguba    #κωδικοποιήθηκε ολόκληρο!
```

Πριν πάμε σε object-oriented programming...

Κάποιες συναρτήσεις στο module random

random() τυχαία επιλογή πραγματικού στο [0,1)
choice(L) τυχαία επιλογή στοιχείου από τη list L

Σαν module, χρειάζεται import για να χρησιμοποιήσουμε τις συναρτήσεις του, τις οποίες καλούμε μέσω **random.function()**

```
import random
a=random.choice(['c',2,True])
print(a)
-----------------------------
c
```

```
import random
a=random.random()
print(a)
------------------
0.6441390766607468
```

```
import random
a=random.random()
r=5+a*(9-5) #τυχαίος πραγματικός στο διάστημα [5,9)
print(r)
-----------------------------------------------------
7.319518611543522
```

OBJECT-ORIENTED PROGRAMMING

Αντικείμενα και κλάσεις

- Αντικείμενα του **ίδιου τύπου** ανήκουν στην **ίδια κλάση** και **κατασκευάζονται από αυτήν**
- Οι κλάσεις είναι **αφηρημένοι τύποι δεδομένων** για τον **ορισμό των αντικειμένων**
- Σε κάθε αντικείμενο ενσωματώνονται τα **δεδομένα** του και οι **συναρτήσεις** που δρουν πάνω σε αυτά.

  - Τα δεδομένα που αντιστοιχούν σε *ιδιοχαρακτηριστικά* τού αντικειμένου ονομάζονται **πεδία**.
  - Οι ενσωματωμένες συναρτήσεις λέγονται **μέθοδοι** και αντιστοιχούν στις ενέργειες που μπορούμε να κάνουμε με το αντικείμενο ή πάνω σε αυτό.

Δομή μιας κλάσης

```
import random
class Coin: #προσομοιώνω ένα νόμισμα
    def __init__(self):     ! Πρέπει πάντα να υπάρχει η __init__
        self.sideup='Heads' ! Αρχικοποιώ τα ιδιοχαρ/κά
    def toss(self): #προσομοιώνω τη ρίψη του νομίσματος σε μέθοδο
        self.sideup=random.choice(['Heads','Tails'])
    def get_sideup(self): #μέθοδος που μου δίνει την πάνω όψη
        return self.sideup
#Τέλος δημιουργίας της κλάσης Coin
#Οι συναρτήσεις που έχω ορίσει μέσα στην κλάση είναι μέθοδοι!
c1=Coin() #δημιουργώ το νόμισμα c1 - έχει το ρόλο τού self
c2=Coin() #δημιουργώ το νόμισμα c2 - έχει το ρόλο τού self
c1.toss() #ρίψη του c1
print(c1.get_sideup()) #τύπωσε την πάνω όψη του c1
print(c2.get_sideup()) #τύπωσε την πάνω όψη του c2
-------------------------------------------------------------
Tails #τυχαία επιλογή
Heads #όχι τυχαία επιλογη - δεν έγινε ρίψη του c2
```

```
class Employee: #κατασκευάζω έναν υπάλληλο
    def __init__(self,nam,sal):
        self.name=nam.upper() #αρχικοποιώ το ιδιο/κό name
        self.salary=sal #αρχικοποιώ το ιδιο/κό salary
    def give_raise(self,percent): #
        self.salary*=(1+percent)
e=Employee('bob',1000) #e for self, 'bob' for nam, 1000 for sal
e.give_raise(0.2) #δίνω στο e αύξηση
print(e.name, e.salary, sep=' ')
-------------------------------------------------------------
BOB 1200.0
```

! Το παραπάνω είναι ένα παράδειγμα ΚΑΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ, διότι μπορώ να πειράξω μόνο άμεσα (εκτός της κλάσης) τις τιμές των ιδιοχαρακτηριστικών self.name και self.salary, μέσω απλών εκχωρήσεων, κάτι που δεν μας είναι γενικά θεμιτό

! Μπορώ να διορθώσω εν μέρει το παραπάνω πρόβλημα "κρύβοντας" τα ονόματα των μεθόδων με δύο κάτω παύλες στην αρχή τους, δηλαδή self.__name και self.__salary, και τώρα δοκιμάζοντας e.__salary=9999, προκύπτει AttributeError

Note: Η Python, λοιπόν, έχει ορατά και αόρατα ονόματα

! Το πρόβλημα δεν έχει διορθωθεί εντελώς, διότι αυτή η απόκρυψη είναι ένα κοινό μυστικό και μπορώ να την παραβιάσω βάζοντας μπροστά από το όνομα της μεθόδου το όνομα της κλάσης με 1 κάτω παύλα, δηλαδή e._Employee__name. Τώρα το αντικείμενο αυτό είναι δεκτικό σε εκχωρήσεις, εκτυπώσεις κλπ

! Στον ΚΑΛΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ μπορώ να πειράξω (εκτός της κλάσης) έμμεσα τις τιμές των ιδιοχαρακτηριστικών, μέσω των λεγόμενων τροποποιητικών μεθόδων (των οποίων το όνομα συνήθως ξεκινάει με *set*), και να τις εκτυπώσω μέσω των λεγόμενων μεθόδων πρόσβασης (των οποίων το όνομα ξεκινάει συνήθως με *get*)

```
class Employee:
    def __init__(self,nam,sal):
        self.name=nam.upper()
        self.salary=sal
    # Τροποποιητικές
    def give_raise(self,percent):
        self.salary*=(1+percent)
    def set_salary(self,money):
        self.salary=money
    def set_name(self,newnam):
        self.name=newnam.upper()
    # Πρόσβασης
    def get_name(self): return self.name
    def get_salary(self): return self.salary
#Τέλος δημιουργίας της κλάσης Employee
e=Employee('Bob',2000)
print(e.get_name(),e.get_salary())
e.set_name('babis')
print(e.get_name())
-----------------------------------------
BOB 2000
BABIS
```

```
#Κατασκευή σημείων στο επίπεδο
import math
class Point:
    def __init__(self,x=0,y=0): #default point το (0,0)
        self.x=x
        self.y=y
    def getX(self): return self.x
    def getY(self): return self.y
    def dist(self,other):
        return math.sqrt((self.x-other.x)**2+(self.y-other.y)**2)
    def move(self,dx,dy):
        self.x+=dx
        self.y+=dy
    def equals(self,other):
        return self.x==other.x and self.y==other.y
    #Τέλος δημιουργίας της κλάσης Point
pt=Point(1/2,3)
print(pt.getX())
p1=Point(1,2)
p2=Point(2,1)
print(p1.equals(p2),p2.dist(p1),sep='\n')
```

```
0.5
False
1.4142135623730951
```

! Είναι εύλογο όμως να θέλω να τυπωθεί το σημείο και με τις 2 συντεταγμένες, κάτι που δεν θα γίνει μέσω της print(pt) στον παραπάνω κώδικα, όπως είναι γραμμένος

! Το παραπάνω θα το καταφέρω προσθέτοντας τις εξής γραμμές κώδικα εντός της κλάσης:

```
def __str__(self):
    return'('+str(self.x)+','+str(self.y)+')'
```

! Γενικά, η print(obj) ψάχνει εντός την κλάσης του obj τη μέθοδο __str__(self) και, αν αυτή υπάρχει, τότε την καλεί αυτόματα και επιστρέφει ό,τι επιστρέφει η __str__(self)

```
def incr_int(n):
    n+=1
    print('n = ',n)
def main():
    x=17
    incr_int(x)
    print('x = ',x)
main()
```

```
n = 18
x = 17 Η τιμή του x δεν άλλαξε, παρόλο που το πήρε η incr_int
```

```
class embedded_int:
    def __init__(self,n):
        self.value=n
    def set_value(self,n):
        self.value=n
    def get_value(self):
        return self.value
    def __str__(self):
        return str(self.value)
#Τέλος δημιουργίας της κλάσης embedded_int
def incr_embedded_int(N):
    N.set_value(N.get_value()+1)
    print('N = ',N)
def change_embedded_int(N):
    t=embedded_int(N.get_value()+1)
    N=t   Αναφορά - ΟΧΙ ΤΑΥΤΙΣΗ
    print('N = ',N)
def main():
    y=embedded_int(17)
    incr_embedded_int(y)
    print('y = ',y)
    change_embedded_int(y)
    print('y = ',y)
main()
-----------------------------------------
N =  18 #Η print της incr_embedded_int
y =  18 #Η 1η print της main
N =  19 #Η print της change_embedded_int
y =  18 #Η 2η print της main



import math
class Rational:
    def __init__(self,x=0,y=1):
        self.num=x
        if y==0:
            self.den=1
        else:
            self.den=abs(y)
            if y<0: self.num=-self.num
            self.simplify() #ορίζεται παρακάτω
    def get_num(self): return self.num
    def get_den(self): return self.den
    def __str__(self):
        if self.den==1:
            return str(self.num)
        else:
            return str(self.num) + '/' + str(self.den)
    def simplify(self):
        g=math.gcd(abs(self.num),abs(self.den)) #ο ΜΚΔ
        self.num//=g #κατ'αναλογία με τους τελεστές += και *=
        self.den//=g
```

```
    def add(self,other): #ορίζω την πρόσθεση
        return Rational(self.num*other.den+self.den*other.num,
        self.den*other.den)
    def times(self,other): #ορίζω τον πολλαπλασιασμό
        return Rational(self.num*other.num, self.den*other.den)
    def equals(self,other): #εξετάζω ισότητα
        return self.num==other.num and self.den==other.den
#Τέλος δημιουργίας της κλάσης Rational
x=Rational(10,30)
y=Rational(5)
z=Rational(1,2)
a=x.add(y)
b=x.add(y).add(z)
c=x.times(y)
print(x,'+',y,'=',a)
print(x,'+',y,'+',z,'=',b)
print(x,'*',y,'=',c)
```

```
-------------------------------------------------------------
1/3 + 5 = 16/3
1/3 + 5 + 1/2 = 35/6
1/3 * 5 = 5/3
```

! Εδώ έχω ορίσει τις *add*, *times* και *equals*, ώστε να κάνουν αυτό που θέλω. Θα ήταν όμως θεμιτό να επιστρέφεται το άθροισμα των x και y μέσω *x+y*, το γινόμενό τους μέσω *x*y* και ο έλεγχος ισότητας μέσω *x==y*, αντί μέσω user-defined συναρτήσεων. Αυτό μπορεί να επιτευχθεί με τη λεγόμενη υπερφόρτωση τελεστών, κατά την οποία ορίζω εκ νέου τη λειτουργία των τελεστών που με ενδιαφέρουν, επαναορίζοντας τις built-in συναρτήσεις που τους αντιστοιχούν.

```
  def __add__(self,other): #επαναορίζω τη λειτουργία του +
          ...
  def __mul__(self,other): #επαναορίζω τη λειτουργία του *
          ...
  def __eq__(self,other):  #επαναορίζω τη λειτουργία του ==
          ...
```

Παράδειγμα

```
class Killer:
    def __init__(self,X):
        self.name=X
    def get_name(self):
        return self.name
    def set_name(self,X):
        self.name=X
    def __add__(self,other):
        return 'See you in your dreams!'
K1=Killer('Freddy Krueger')
K2=Killer('Pennywise The Clown')
print(K1+K2)
```

```
---------------------------------------
See you in your dreams!
```

! Η υπερφόρτωση τελεστών γίνεται ΕΝΤΟΣ της κλάσης και αφορά ΜΟΝΟ τα αντικείμενα που ανήκουν σε αυτή. Η αλλαγή της λειτουργίας ενός τελεστή ΔΕΝ είναι global.

```
print(5+2)
----------------------
7 #σε όλα τα σύμπαντα
```

Κληρονομικότητα

Έστω Β **υποκλάση** της κλάσης Α (**υπερκλάση**).
- Η Β κληρονομεί τα ιδιοχαρακτηριστικά και τις μεθόδους της Α.
- Η Β μπορεί να προσθέσει **νέα ιδιοχαρακτηριστικά** και μεθόδους που την **εξειδικεύουν** σε σχέση με την Α
- Η Β μπορεί να **υπερκαλύψει** τις μεθόδους της Α: να έχει μεθόδους με το ίδιο όνομα, οι οποίες χρησιμοποιούνται για εξειδικευμένες ανάγκες της.

! Η υπερκάλυψη (overriding) είναι μια διαδικασία όπου μια υποκλάση παρέχει τη δική της υλοποίηση για μια μέθοδο που έχει ήδη οριστεί στην υπερκλάση. Αυτό επιτρέπει στις υποκλάσεις να προσαρμόζουν ή να επεκτείνουν τη συμπεριφορά της υπερκλάσης.

```
class Person:
    def __init__(self,nam):
        self.name=nam
    def get_name(self):
        return self.name
    def __str__(self):
        return self.name
class Student(Person):  Στην παρένθεση το όνομα της υπερκλάσης
    def __init__(self,nam,AM):
        Person.__init__(self,AM) Αναφέρομαι στην __init__ της υπερκλάσης
        self.AM=str(AM)
        self.courses={}
    def __str__(self):
        return Person.__str__(self)+' ('+self.AM+')'
    def get_AM(self):
        return self.AM
    def add_course_grade(self,course,grade):
        self.courses[course]=int(grade)
    def show_grades(self):
        for course in self.courses:
            print(course,self.courses[course])
    def get_MO(self):
        return sum(self.courses.values())/len(self.courses)
s1=Student('Mary',1112201900035)
s1.add_course_grade('251',8)
s1.add_course_grade('141',9)
s1.add_course_grade('617',6)
print(s1.get_name())
s1.show_grades()
print(s1.get_MO())
-------------------------------------------------------------
```

```
251 8
141 9
617 6
7.666666666666667
```

Πολυμορφισμός

Επιτρέπει σε αντικείμενα διαφορετικών κλάσεων να χρησιμοποιούνται μέσω της ίδιας διεπαφής. Αυτό σημαίνει ότι μια μέθοδος μπορεί να συμπεριφέρεται διαφορετικά ανάλογα με το αντικείμενο που την καλεί.

Έστω η μέθοδος *method* που έχει οριστεί και στην υπερκλάση και στην υποκλάση.

```
subclass_obj.method()     #call subclass method

superclass_obj.method()   #call superclass method
```

```
class Player:
    def __init__(self,nam):
        self.name=nam
    def get_name(self):
        return self.name
    def job(self):
        print('I play games')
class FootballPlayer(Player):
    def __init__(self,nam,goals):
        Player.__init__(self,nam)
        self.total_goals=goals
    def job(self):
        print('I play football')
class BasketballPlayer(Player):
    def __init__(self,nam,points):
        Player.__init__(self,nam)
        self.total_points=points
    def get_points(self):
        return self.total_goals
    def job(self):
        print('I play basketball')
def player_sport(x):
    print('I am',x.get_name(),end=': ')
    x.job()
def main():
    p=Player('John Doe')
    f=FootballPlayer('Messi',800)
    b=BasketballPlayer('Gallis',24805)
    player_sport(p)
    player_sport(f)
    player_sport(b)
main()
---------------------------------------
```

```
I am John Doe: I play games
I am Messi: I play football
I am Gallis: I play basketball
```

! **Η player_sport δουλεύει μόνο με αντικείμενα που έχουν μεθόδους get_name και job**

Την ορίζω διαφορετικά, με έλεγχο μέσω της isinstance(obj,class)

```
def player_sport(x):
    if isinstance(x,Player):
       print('I am',x.get_name(),end=':')
       x.job()
    else:
       print('Not a player')
```

Η isinstance(x,Player) θα εξετάσει **αν το αντικείμενο x ανήκει στην κλάση Player ή σε οποιαδήποτε υποκλάση της** και θα επιστρέψει τιμή True ή False αναλόγως.

! Την isinstance μπορώ επίσης να τη χρησιμοποιήσω για να ελέγξω αν ένα στοιχείο είναι συγκεκριμένου τύπου (str, list, int κλπ), καθώς στην Python όλα είναι αντικείμενα και οι τύποι είναι οι κλάσεις αυτών των αντικειμένων!

```
print(isinstance([1,2,3], list))
print(isinstance((1,2,3), tuple))
print(isinstance(3,int))
print(isinstance(3,float))
print(isinstance('a1b2c3',str))
-----------------------------------
True
True
True
False
True
```

```
print('That's all folks! Please code responsibly.')
-------------------------------------------------
SyntaxError: unterminated string literal
```