1η εσπερίδα - Τρίτη 5 Νοεμβρίου 2024, 18:00-20:00	Χρόνος	Εισηγητές
Προγραμματισμός Υπολογιστικών Συστημάτων Μεταβλητές, Δομές Επανάληψη, Δομή Επιλογής στο Scratch, Χειρισμός Γεγονότων	60’	Βραχνός Ευριπίδης
Τμηματικός Προγραμματισμός Υποπρόγραμματα στο Scratch, Ρεαλιστική κίνηση	20’	Αθανασάκου Πηνελόπη
Δομές Δεδομένων - Λίστες στο Scratch	20’	Κολεγά Ευαγγελία

2η εσπερίδα - Τρίτη 12 Νοεμβρίου 2024, 18:00-20:30	Χρόνος	Εισηγητές
Προγραμματισμός με τη γλώσσα Python για εκπαιδευτικούς	120’	Βραχνός Ευριπίδης – Παπαδάκης Σπυρίδων

3η εσπερίδα - Τετάρτη 20 Νοεμβρίου 2024, 18:00-20:30	Χρόνος	Εισηγητές
Τεχνητή Νοημοσύνη Β Τάξης	60’	Αθανάσακου Πηνελόπη – Παντελοπούλου Σταυρούλα
Τεχνητή Νοημοσύνη Γ Τάξης	60’	Τριανταφύλλου Χρήστος – Τζελέπη Σοφία – Παντελοπούλου Σταυρούλα

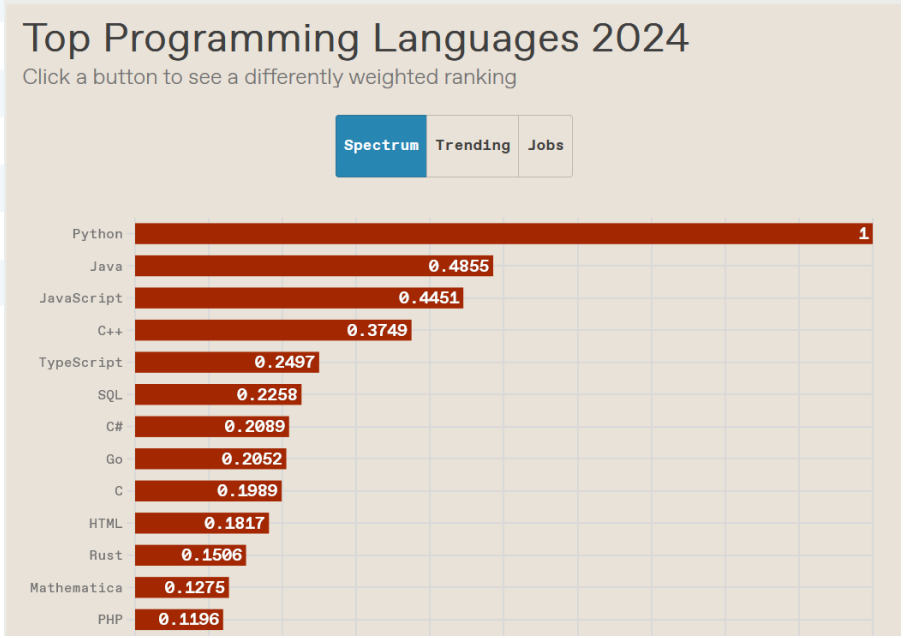
4η εσπερίδα - Πέμπτη 28 Νοεμβρίου 2024, 18:00-20:30	Χρόνος	Εισηγητής
Διδασκαλία του προγραμματισμού με τη γλώσσα Python Διδακτικές προσεγγίσεις, παρανοήσεις και δυσκολίες των μαθητών	120’	Βραχνός Ευριπίδης – Γώγουλος Γεώργιος

5η εσπερίδα - Τετάρτη 4 Δεκεμβρίου 2024, 16:00-18:30	Χρόνος	Εισηγητές
Αλγοριθμική Α’ τάξη	20’	Βραχνός Ευριπίδης
Αλγοριθμική Γ’ τάξη	20’	
Επιστημονικός Προγραμματισμός	20’	
Φυσική Υπολογιστική / Ρομποτικές Διατάξεις	60’	Κολεγά Ευαγγελία - Τριανταφύλλου Χρήστος – Αθανασάκου Πηνελόπη

Οι πιο δημοφιλείς γλώσσες

Programming Language	2024	2019	2014	2009	2004	1999	1994	1989
Python	1	3	8	7	7	24	22	-
C	2	2	1	2	2	1	1	1
C++	3	4	4	3	3	2	2	2
Java	4	1	2	1	1	4	-	-
C#	5	6	5	6	8	19	-	-
JavaScript	6	7	9	9	10	15	-	-
Visual Basic	7	20	235	-	-			
SQL	8	9	-	-	92			
Go	9	16	36	-	-			
Fortran	10	28	29	26	13			
Objective-C	34	10	3	28	39			
Lisp	35	32	18	18	14			
(Visual) Basic	-	-	6	5	4			

TIOBE index:



Πηγή: https://spectrum.ieee.org/top-programming-languages/

- **Εισαγωγή στην Python για εκπαιδευτικούς ΠΕ86**:
  1. Βασικά στοιχεία της γλώσσας
  2. Τύποι – Τελεστές – Μεταβλητές
  3. Συναρτήσεις
  4. Δομές επιλογής – Δομές Επανάληψης
  5. Λίστες
  6. Πλειάδες – Σύνολα - Λεξικά
  7. Αλφαριθμητικά
  8. Υλοποίηση δομών δεδομένων (Στοίβα - Ουρά)

ΕΠΙΜΟΡΦΩΤΙΚΟ ΥΛΙΚΟ - ΥΛΙΚΟ ΜΕΛΕΤΗΣ ΓΙΑ Python (6+6)

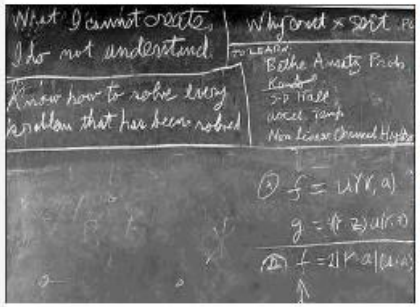
- ΒΙΒΛΙΑ

1. Guido van Rossum and the Python Development Team**. Python Tutorial**. (En) Python Software Foundation https://docs.python.org/3/tutorial/index.html
2. Αγγελιδάκης, Ν.Α., 2015. **Εισαγωγή στον προγραμματισμό με την Python**, Διαθέσιμο στο σύνδεσμο: http://aggelid.mysch.gr/pythonbook/
3. Swaroop, C. H. 2013. **A Byte of Python**, Επιμ. μετάφρασης **Ubuntu community,** https://archive.org/details/AByteOfPythonEl Πρωτότυπο (En) στο σύνδεσμο: https://open.umn.edu/opentextbooks/textbooks/581
4. Μανής, Γ., 2015. **Εισαγωγή στον Προγραμματισμό με αρωγό τη γλώσσα Python**. Αθήνα: Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών. http://hdl.handle.net/11419/2745
5. Allen B. Downey. **Think Python,** https://greenteapress.com/wp/think-python-2e/
6. Al Sweigart, **Automate the Boring Stuff with Python,** https://automatetheboringstuff.com/

- MOOC's

1. **Edx - Introduction to Computer Science and Programming Using Python** https://learning.edx.org/course/course-v1:MITx+6.00.1x_7+3T2015/home
2. **Harvard CS50's Introduction to Programming with Python** https://cs50.harvard.edu/python/2022/
3. Coursera Charles Severanc **Python for Everybody -** https://www.py4e.com/
4. **Mathesis**. Αβούρης Ν. **Εισαγωγή στην Python** https://mathesis.cup.gr/courses/course-v1:ComputerScience+CS1.1+23D/about
5. ΑΠΘ Δημητριάδης Σ. **Προγραμματισμός σε Python για μη-Προγραμματιστές** https://colmooc.gunet.gr/info/?course=COLMOOC113
6. Mathesis. Αβούρης Ν. **Προχωρημένος προγραμματισμός με Python** **https://mathesis.cup.gr/courses/course-v1:ComputerScience+CS1.2+24A/about**

https://evripides.mysch.gr/



What I cannot create I do not understand. --- Richard Feynman

The greatest teacher, failure is. --- Master Yoda

http://evripides.mysch.gr -- **Google Scholar** --

Σύμβουλος Εκπαίδευσης Πληροφορικής

Ιστοσελίδα Συμβούλων Εκπαίδευσης Πληροφορικής

- Αρχική
- Σύντομο Βιογραφικό
- Έρευνα
- Δημοσιεύσεις
- Πληροφορική
- Όμιλος Αλγοριθμικής
- Διαγωνισμός Πληροφορικής
- Δράσεις στο Ζάννειο Πρότυπο Γυμνάσιο
- Επικοινωνία

Τελευταία Νέα :

Συνέδριο CIE'23: Η πληροφορική στην

Σημειώσεις για προγραμματισμό Scratch και Python και φύλλα εργασίας με βάση τα νέα βιβλία του Γυμνασίου

Python **Scratch**

Σύνδεσμοι:

Project Euler

Python Challenge

CoderByte

English

Αλγοριθμική (aka Προγραμματισμός στα Ελληνικά)

```
Πρόγραμμα Πίνακες_σε_Υποπρογράμματα
Μεταβλητές
   Ακέραιες: ω, Σ, Α[8]
Αρχή
     ……..
     Κάλεσε Άθροισμα( Α, 8, Σ)
     Γράψε Σ
Τέλος_Προγράμματος
```

```
Διαδικασία Άθροισμα( Β, Ν, Σ )
Μεταβλητές
   Ακέραιες: Β[10], δ, Ν, Σ
Αρχή
     Σ ← 0
     Για δ από 1 μέχρι Ν
          Σ ← Σ + Β[δ]
     Τέλος_Επανάληψης
Τέλος_Διαδικασίας
```

Έλεγχος υποσυνόλου

```
def isSubset(A,B) :
    for element in A:
        if element not in B:
            return False
    return True
```

- Δυναμικό σύστημα τύπων
- Lazy Evaluation
- ***Μην κάνεις σήμερα αυτό που μπορείς να αναβάλλεις για αύριο*** ☺ (*Προσοχή! Δεν ισχύει για το διάβασμα στις εξετάσεις*)
- Πολύ υψηλού επιπέδου (ψευδογλώσσα)

```
>>> isSubset([3,10,1], [2,1,4,3,10])
True
>>> isSubset(["A", "C"], ["A", "B", "C", "E"])
True
>>> isSubset("cie", "cie2024")
True
```

Java ή Python ?

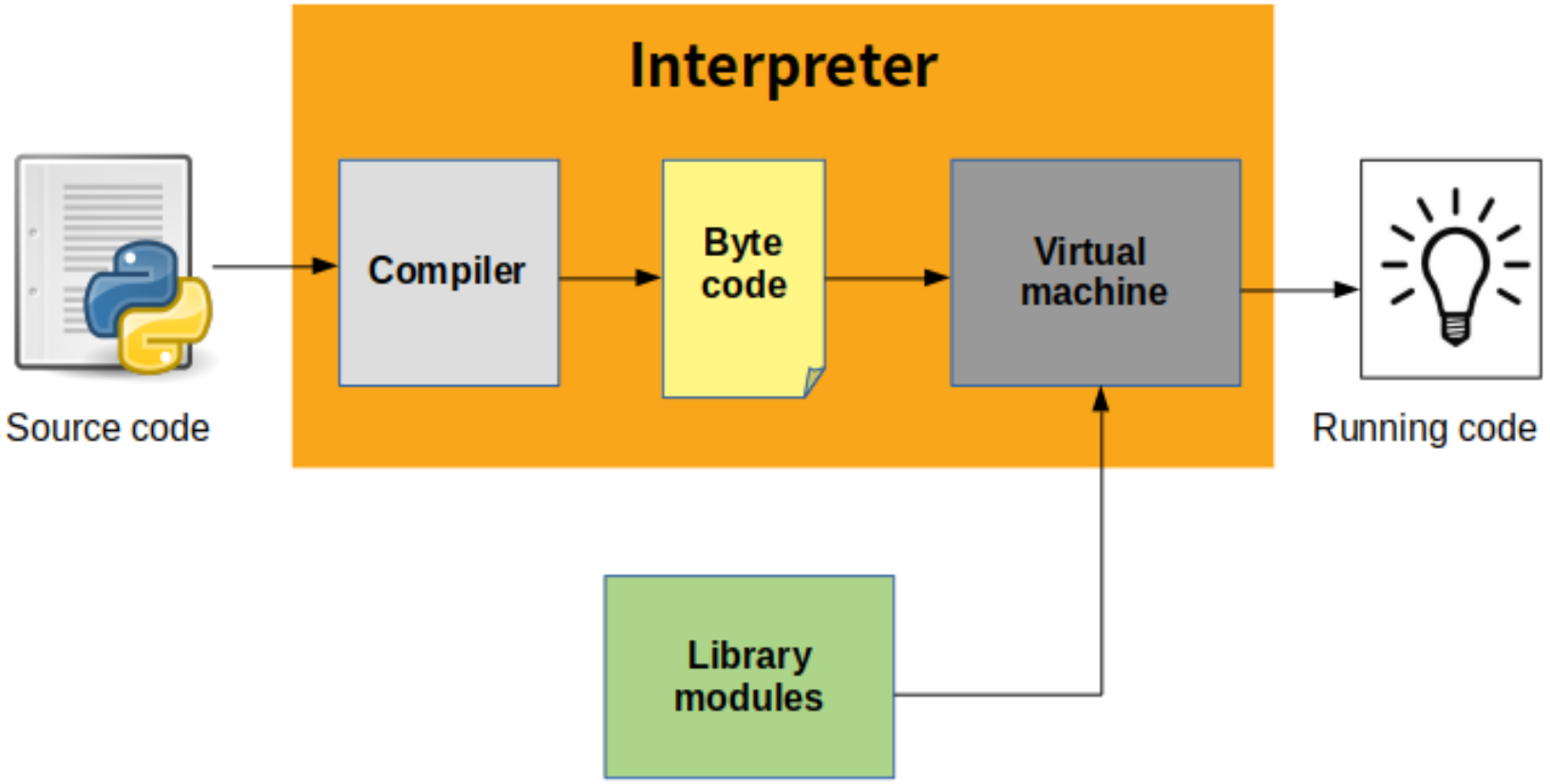
```
public class HelloWorld {

    public static void main(String args[])
    {
            System.out.println("Hello World");

            return 0;
    }
}
```

```
print ( "Hello world" )
```

Περιβάλλον εκτέλεσης της Python



Python ByteCode

```
>>> import dis
>>> def add(a, b):
        c = a + b
        return c

>>> dis.dis(add)
  2           0 LOAD_FAST                0 (a)
              2 LOAD_FAST                1 (b)
              4 BINARY_ADD
              6 STORE_FAST               2 (c)

  3           8 LOAD_FAST                2 (c)
             10 RETURN_VALUE
>>>
```

Συστήματα Τύπων Γλωσσών Προγραμματισμού

Python

```
>>> 1 + 2 + “3”
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Javascript

```
>>> 4 + 5 + “9”
99
```

```
>>> “9” + 4 + 5
“945”
```

```
>>> “” + 945
“945”
```

Εναλλακτικά περιβάλλοντα συγγραφής κώδικα (IDE) σε Python

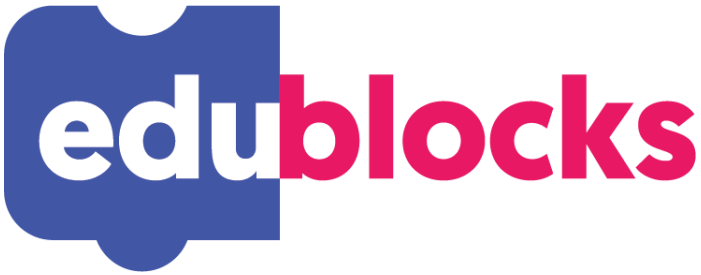
Εκτός από το IDLE (Integrated Development and Learning Environment) :

1. PyCharm.
2. Visual Studio Code
3. Sublime Text 3
4. Atom
5. Jupyter
6. Spyder
7. PyDev
8. Thonny (*For Beginners*)
9. http://pythontutor.com/

Ένα IDE επιτρέπει στους προγραμματιστές να συνδυάζουν τις διαφορετικές πτυχές της συγγραφής ενός προγράμματος υπολογιστή και αυξάνει την παραγωγικότητα του προγραμματιστή εισάγοντας λειτουργίες όπως η επεξεργασία του πηγαίου κώδικα, η δημιουργία εκτελέσιμων αρχείων και ο εντοπισμός σφαλμάτων.

10. https://www.programiz.com/python-programming/online-compiler/

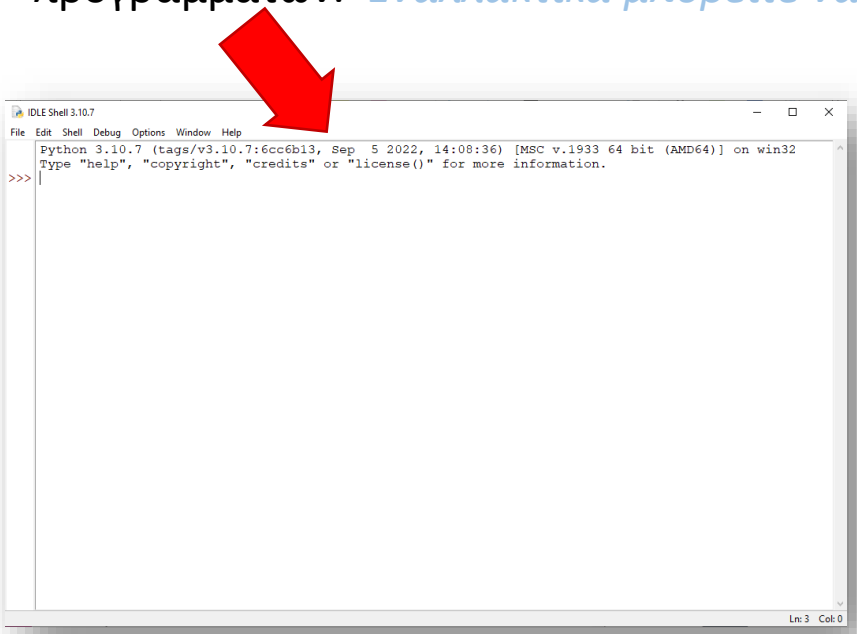
Περιβάλλοντα Προγραμματισμού

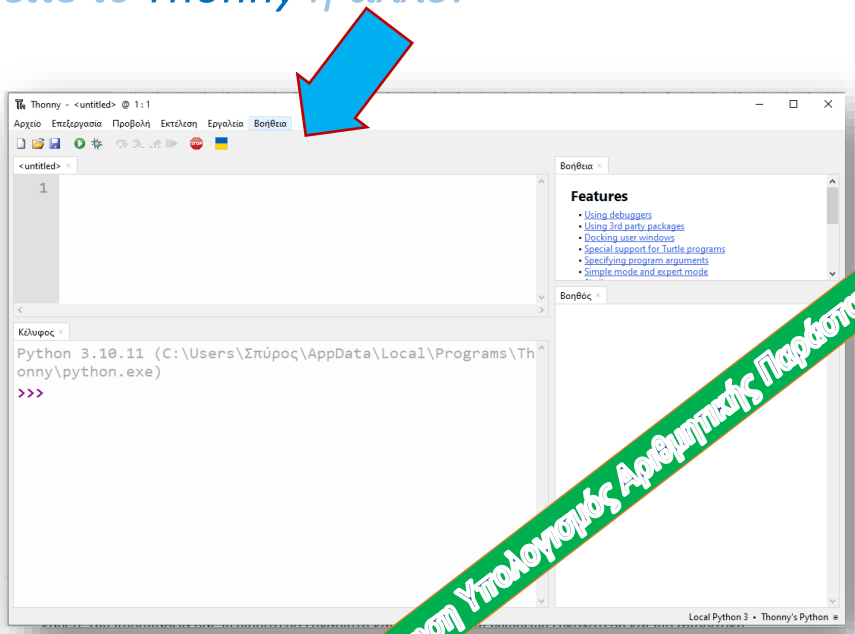


IDLE

Εγκατάσταση περιβάλλοντος Python

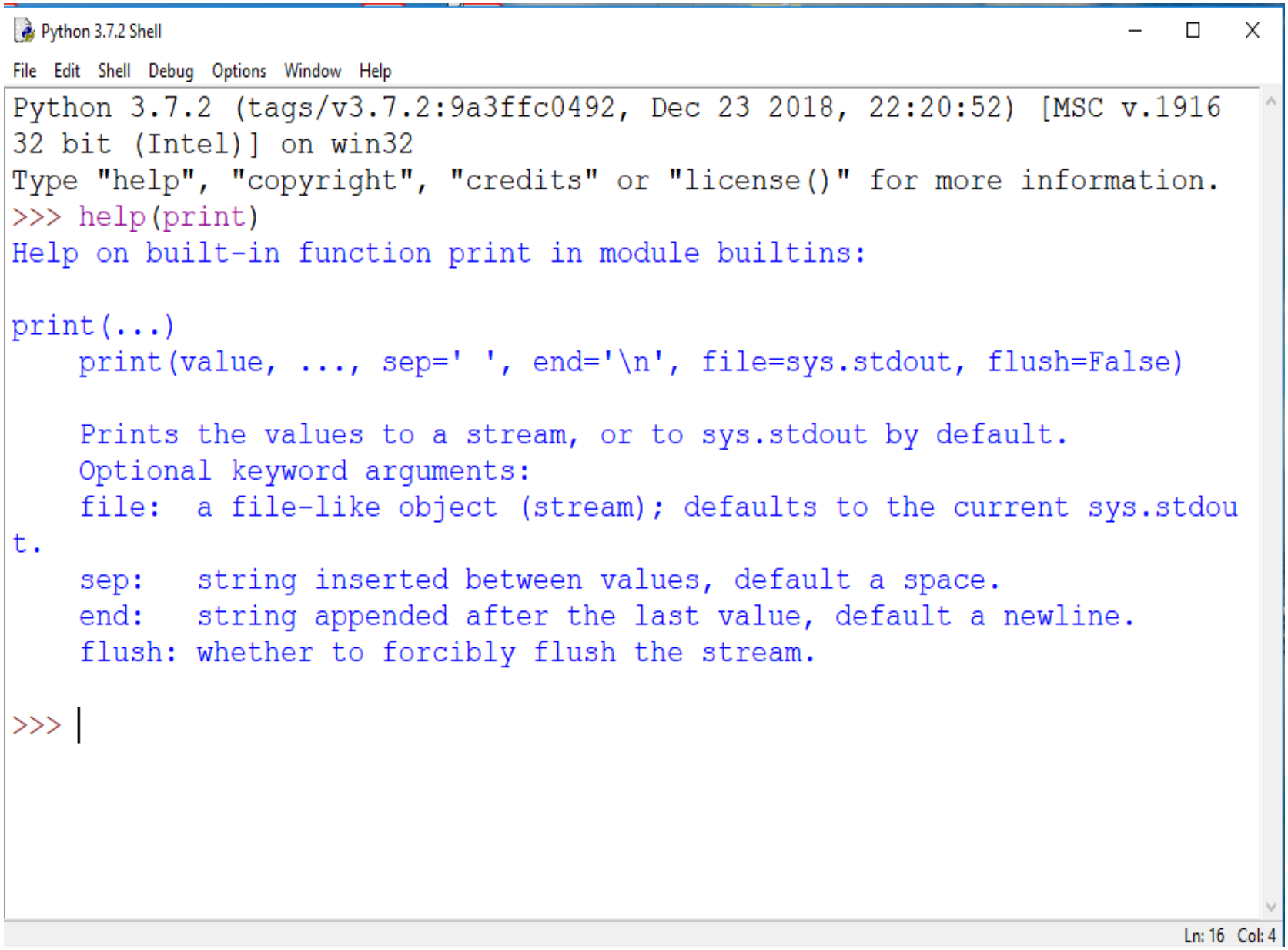
Μετά την εγκατάσταση, εκτέλεση του **IDLE** εμφανίζεται το **κέλυφος Python (Python Shell)** που επιτρέπει την διαδραστική εκτέλεση εντολών Python και εμφανίζει το αποτέλεσμα προγραμμάτων. *Εναλλακτικά μπορείτε να δείτε το Thonny ή άλλο.*



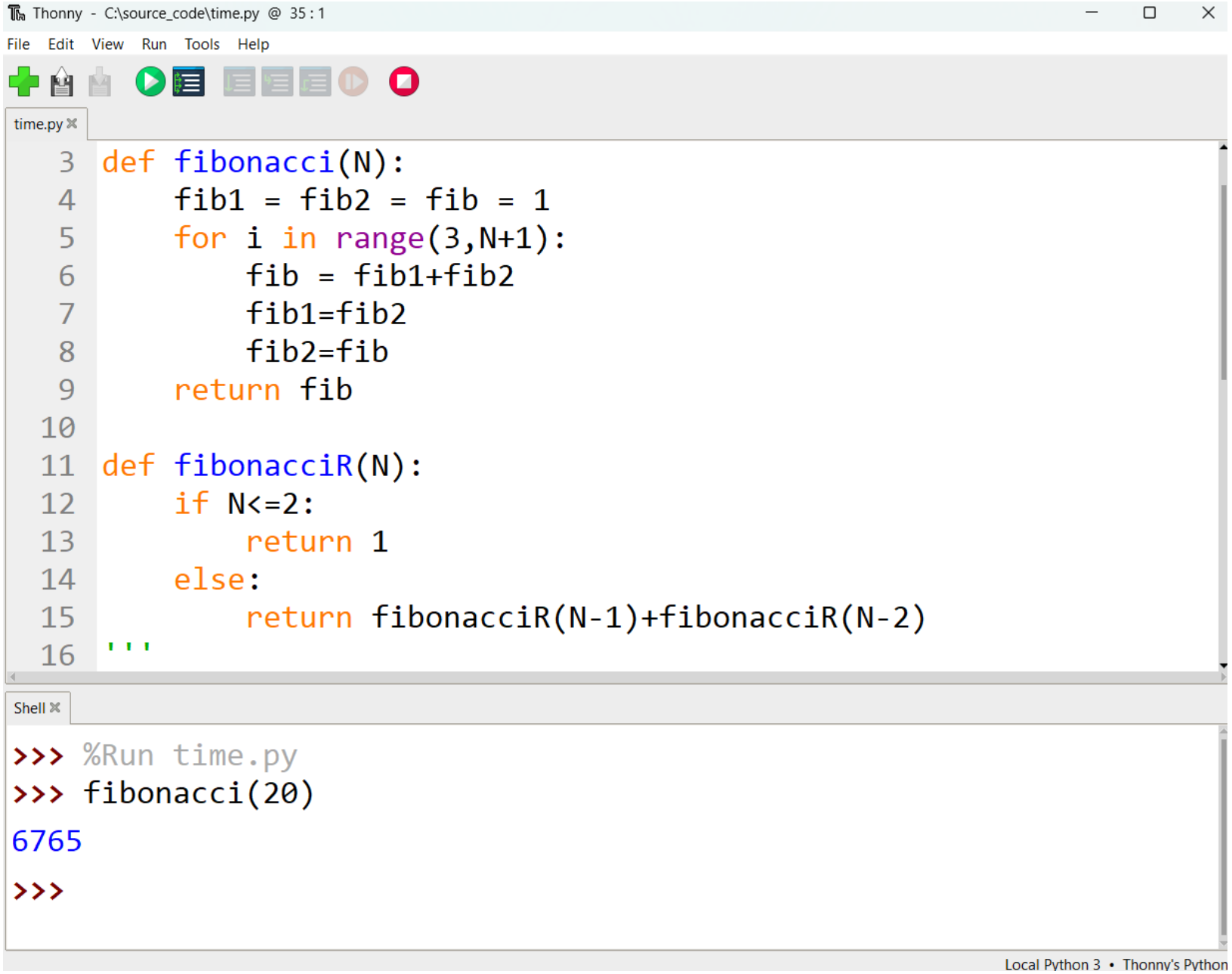


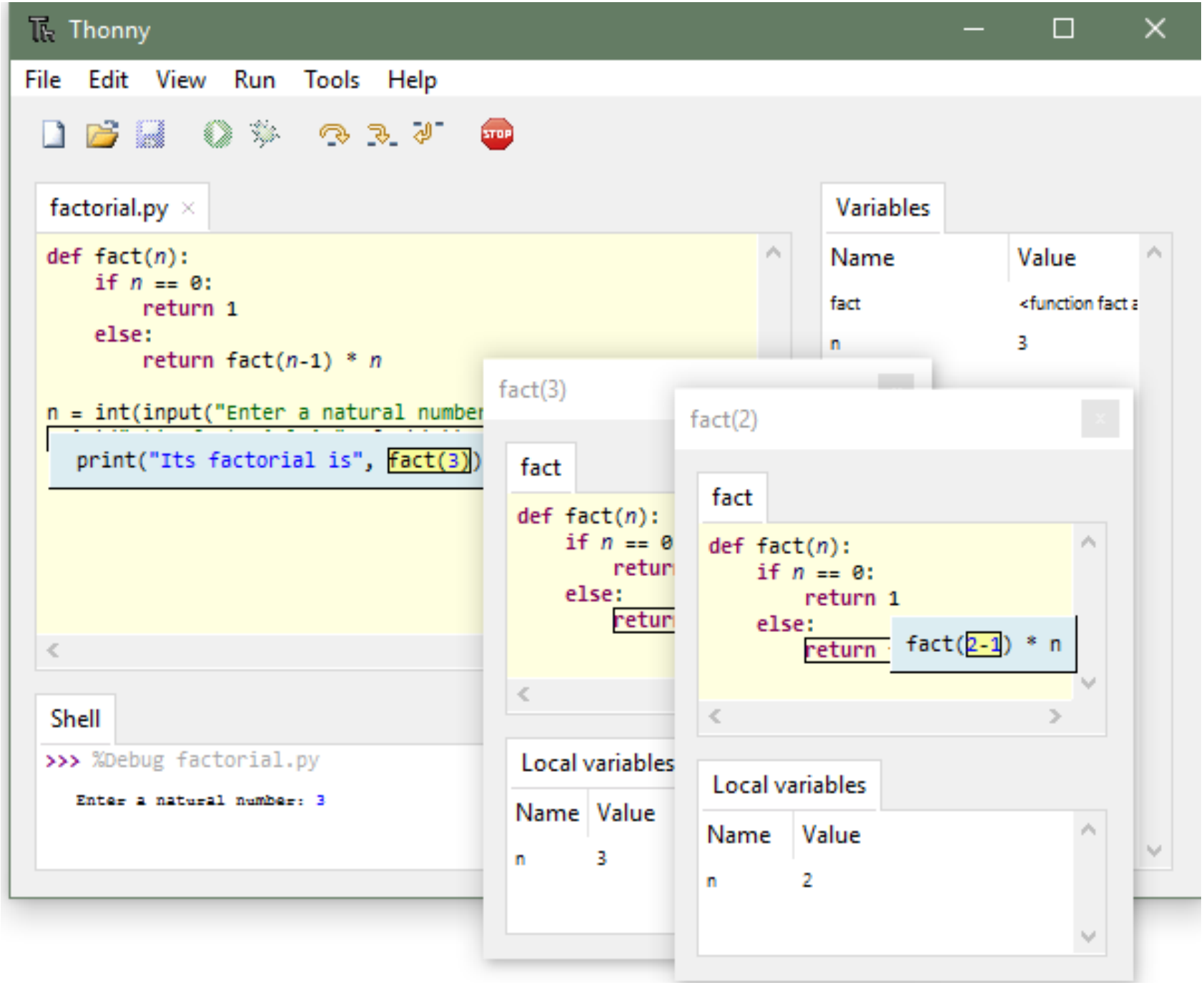
Άσκηση Υπολογισμός Αριθμητικής Παράστασης

Το περιβάλλον IDLE της Python

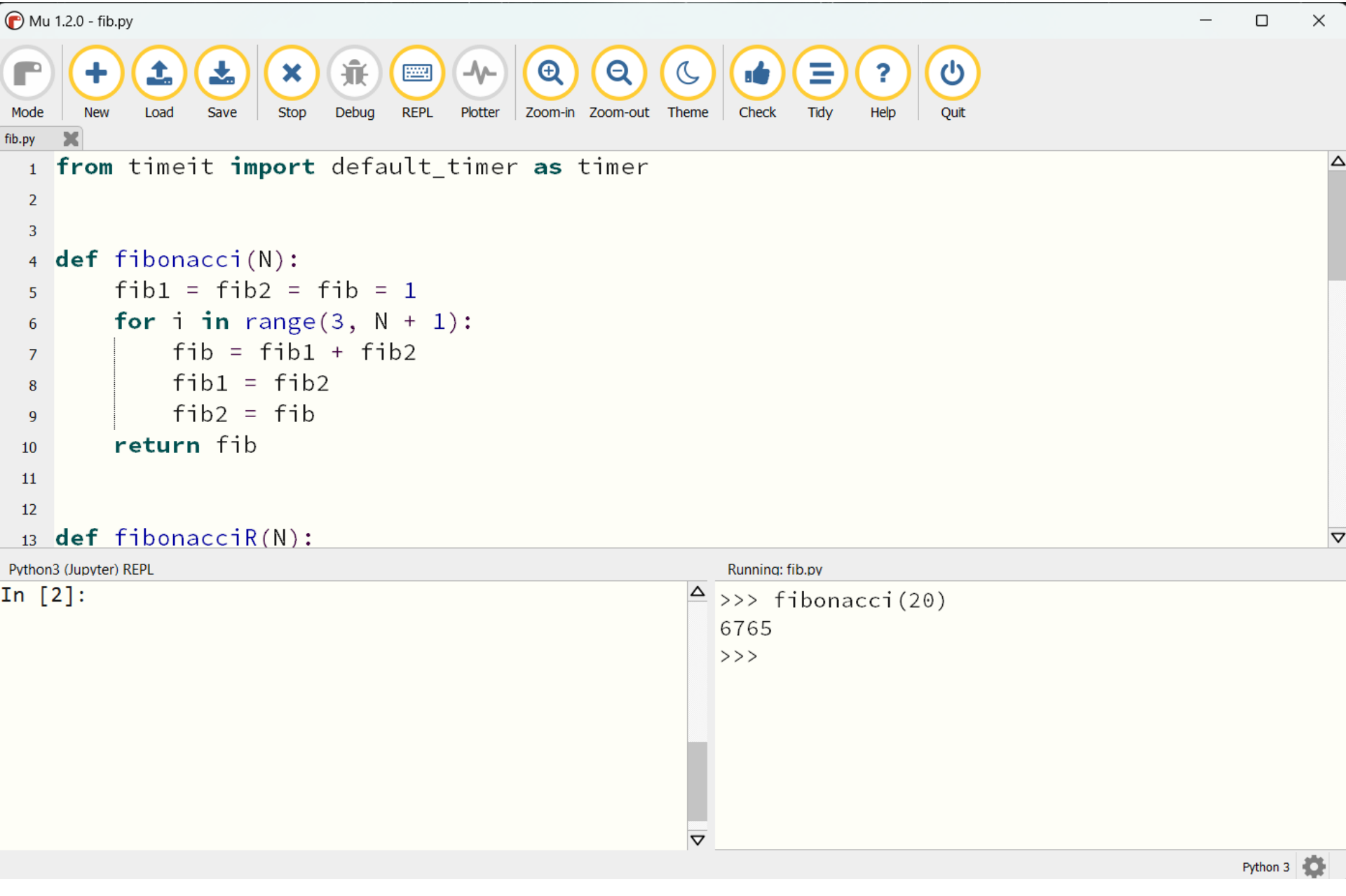


https://thonny.org/

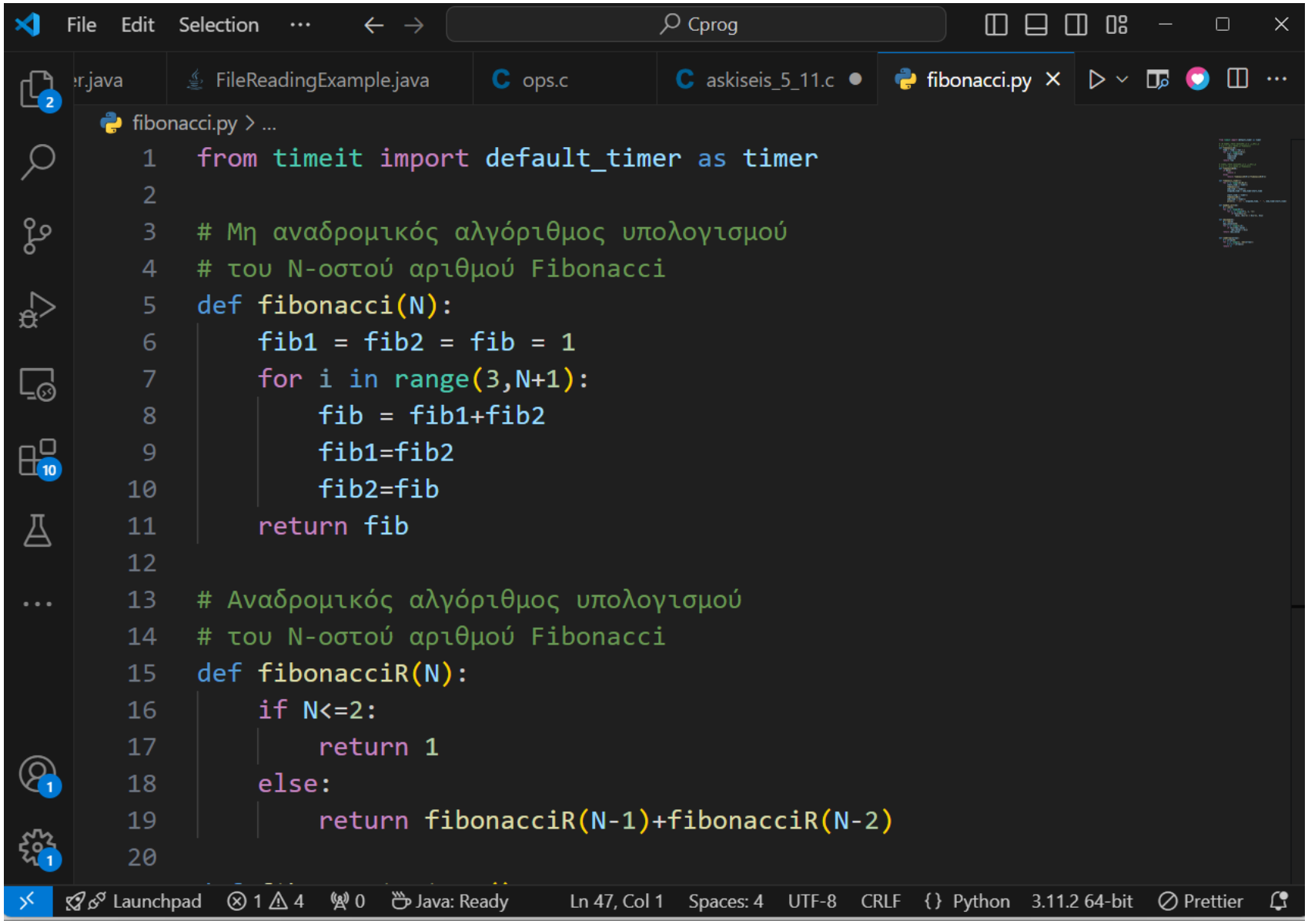



Thonny
File Edit View Run Tools Help
factorial.py
def fact(n):
if n == 0:
return 1
else:
return fact(n-1) * n
n = int(input("Enter a natural number
print("Its factorial is", fact(3))
Variables
Name Value
fact <function fact a
n 3
Shell
>>> %Debug factorial.py
Enter a natural number: 3
fact(3)
fact
def fact(n):
if n == 0
retur
else:
retur
Local variables
Name Value
n 3
fact(2)
fact
def fact(n):
if n == 0:
return 1
else:
return fact(2-1) * n
Local variables
Name Value
n 2

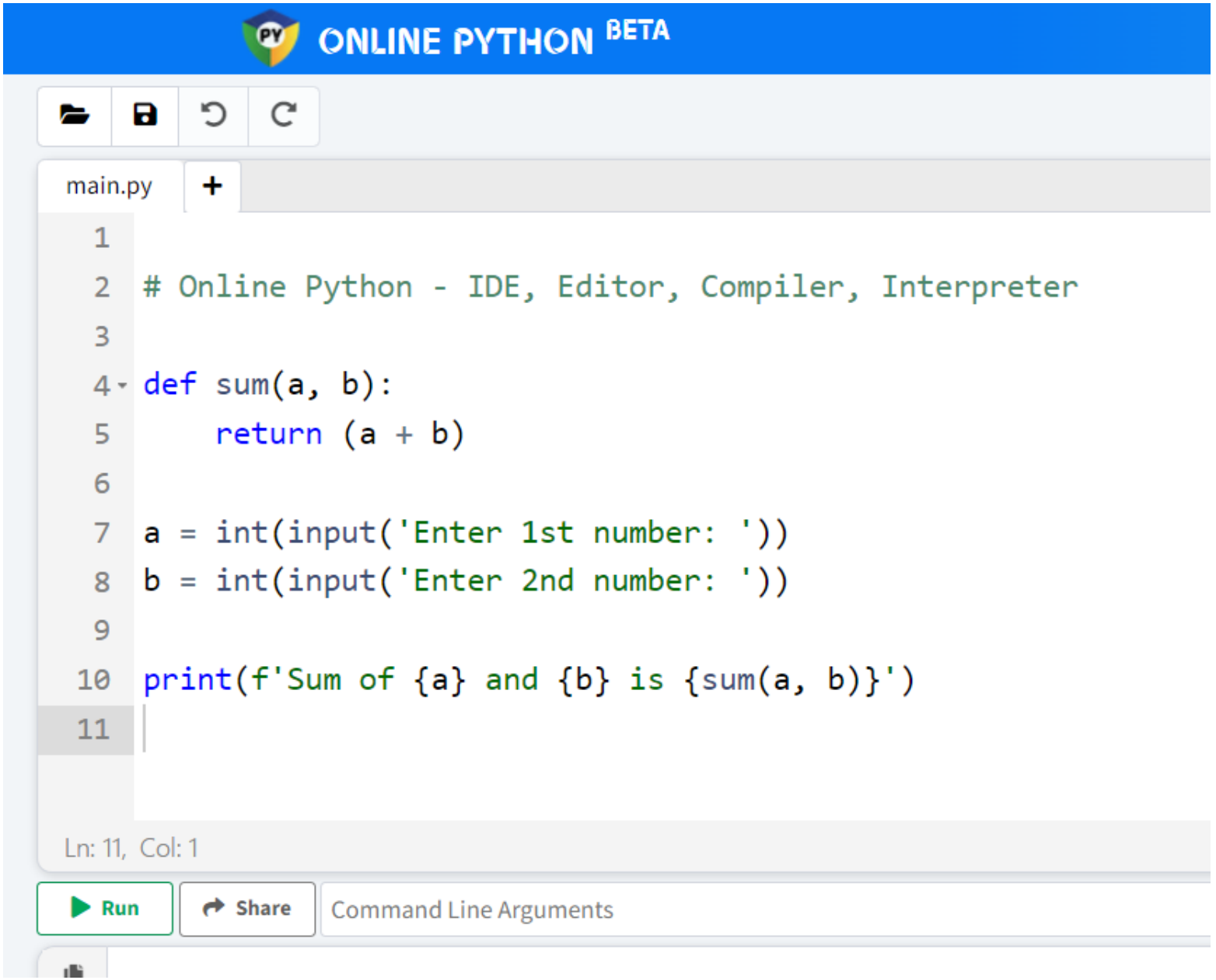
https://codewith.mu/



https://code.visualstudio.com/

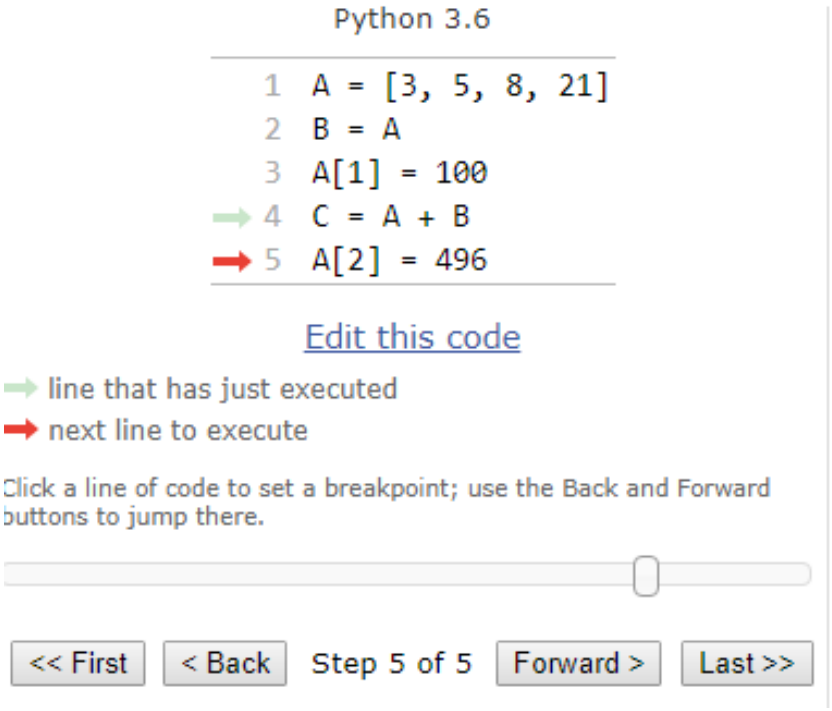


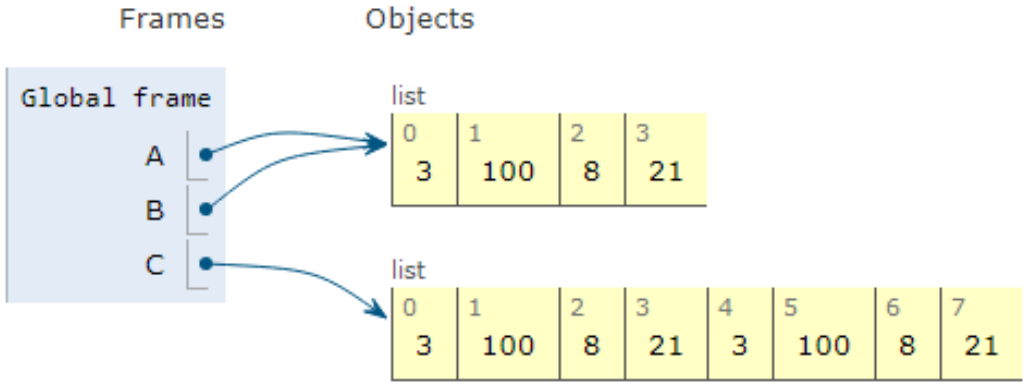
https://www.online-python.com/



Python Tutor (http://pythontutor.com/)

Θα σας βοηθήσει πολύ να καταλάβετε την αναπαράσταση των δομών δεδομένων στη μνήμη





Κύρια χαρακτηριστικά της Python

- Δωρεάν και άμεσα προσβάσιμη
- Τρέχει σε οποιοδήποτε Λειτουργικό Σύστημα (Windows, MacOSX, Linux, iOS, Android, κτλ.)
- Ανοικτός κώδικας
- Ευκολία εκμάθησης και χρήσης
- Πληθώρα βιβλιοθηκών και έτοιμου κώδικα
- Ύπαρξη διερμηνευτή εντολών για άμεση εκτέλεση εντολών

Η Python

Γκίντο βαν Ρόσσου
Guido van Rossum

- Διαθέσιμη σε 2 εκδόσεις
  - **Python 2.x** (πρώτη έκδοση Οκτώβριος 2000) με τελευταία έκδοση 2.7
  - **Python 3.x** (πρώτη έκδοση Δεκέμβριος 2008): ιστορικά η πρώτη γλώσσα προγραμματισμού που σπάει την προς τα πίσω συμβατότητα
- Οι Python 2.x και Python 3.x είναι μη συμβατές. Έχουν μικρές αλλά **ουσιαστικές διαφορές**.
- Η 2.x συνεχίζει να υπάρχει για όσους ήδη έχουν γράψει κώδικα σε 2.x και στα ΕΠΑΛ
- Πλέον μαθαίνουμε και χρησιμοποιούμε την **3.x.**

Υποστήριξη Προγραμματιστικών Υποδειγμάτων:

- **Προστακτικός προγραμματισμός**
  - Imperative programming
- **Δομημένος προγραμματισμός**
  - Structured programming
- Διαδικαστικός προγραμματισμός
  - Procedural programming
- Συναρτησιακός προγραμματισμός
  - Functional programming
- **Αντικειμενοστρεφής προγραμματισμός**
  - Objected oriented programming

Εγκατάσταση περιβάλλοντος Python

- www.python.org
  - Εγκατάσταση της πιο πρόσφατης έκδοσης **Python 3**.x.y (3.13.0 ή όποια τρέχει ο

Τα πάντα είναι αντικείμενα

- Και όταν λέμε τα πάντα, εννοούμε τα πάντα
- Δεν είναι υβριδική γλώσσα (βλέπε Java, C++)
- Πολύ κοντά σε Ruby
- Χρησιμοποιούμε dot notation

```
>>> ( 128 ).bit_length()
8
>>> “python”.upper()
‘PYTHON’
>>> “python”[1]
’y’
```

```
>>> True.bit_length()
1
>>> (2**1000000).bit_length()
1000001
>>> (1.5).as_integer_ratio()
 (3, 2)
```

Πρόγραμμα ...

- … ακολουθία εντολών που καθορίζει το τρόπο εκτέλεσης ενός υπολογισμού.

Ο υπολογισμός μπορεί να είναι

- **μαθηματικός**, όπως η επίλυση ενός συστήματος εξισώσεων ή η εύρεση των ριζών ενός πολυωνύμου,
  - **συμβολικός**, όπως η αναζήτηση και η αντικατάσταση λέξεων σε ένα κείμενο, ή κάτι
  - **γραφικός**, όπως η επεξεργασία μιας εικόνας ή η αναπαραγωγή ενός video.

- Σενάριο (script) – πηγαίος κώδικας

Python

- Ονόματα, εκφράσεις, τελεστές, δεσμευμένες λέξεις
- Ονόματα μεταβλητών, τιμές, δεδομένα (αντικείμενα)
- **Τύποι Δεδομένων**: Ακέραιοι (int), πραγματικοί (float), συμβολοσειρές/αλφαριθητικά (str)
  - Τιμές (values), Τελεστές (operators) / Πράξεις,
  - Αναπαράσταση στη μνήμη, Συναρτήσεις / Μεθόδους

- Είσοδος, Επεξεργασία, Έξοδος
- Εισαγωγή δεδομένων από το πληκτρολόγιο, Εμφάνιση στην οθόνη
- Μνήμη Υπολογιστή (*όνομα, τιμή, σύνδεση - δείκτης*)

Βασικές Έννοιες Δομημένου Προγραμματισμού

- Αλγόριθμος
- Πρόγραμμα
- Δομή δεδομένων
- Δομή επιλογής
- Δομή επανάληψης
- Συνάρτηση
- …

Πώς τα κάνουμε όλα αυτά στην Python;

Εμφάνιση στην οθόνη

Η ενσωματωμένη συνάρτηση **print()** χρησιμοποιείται για την **εμφάνιση** των **ορισμάτων** της στην **οθόνη**.

```
>>> print("Γεια σας ")
Γεια σας

>>> print(«Καλώς","ήρθατε")
Καλώς ήρθατε

>>> print(«Σχολικό Έτος","2024-2026", sep='---’)
Σχολικό Έτος---2024-2025

>>> print("Σχολικό Έτος","\n","2024-2025")
Σχολικό Έτος-
2024-2025

>>> print(" Σχολικό Έτος","\t","2024-2025")
1η ΟΣΣ                  2024-2025
```

Στο IDLE shell η επανεμφάνιση της τελευταίας εντολής που δώσατε επιτυγχάνεται με Alt+P στα Windows ή Control+P στο MacOS.

Δεν υπάρχουν μόνο σταθερές τιμές...

```
>>> s1 = «Καλώς"
>>> s2 = «Ήρθατε"
>>> print(s1, s2)
      Καλώς ήρθατε

>>> s2 = "2024-2025"
>>>print(s1, s2)
     Καλώς 2024-2025
```

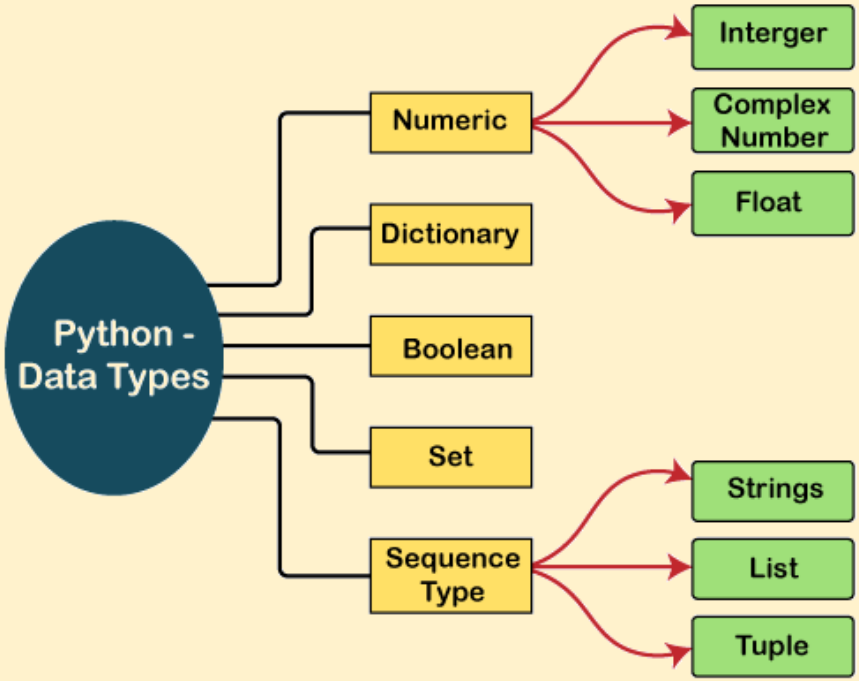
Η s2 έχει ένα όνομα (το s2) και μια τιμή η οποία άλλαξε (από "Ήρθατε" σε "2024-2025")

Πρόκειται για μια **μεταβλητή**. Μεταβλητή είναι και η s1.

Χρησιμοποιείται το σύμβολο '=' που αντιστοιχεί στην εντολή **εκχώρησης (ανάθεσης)** τιμής σε μεταβλητή.

Οι μεταβλητές έχουν επιπλέον έναν τύπο (χρήση της type()) και οι τιμές τους αποθηκεύονται στη μνήμη με έναν αριθμό ταυτότητας (χρήση της id()).

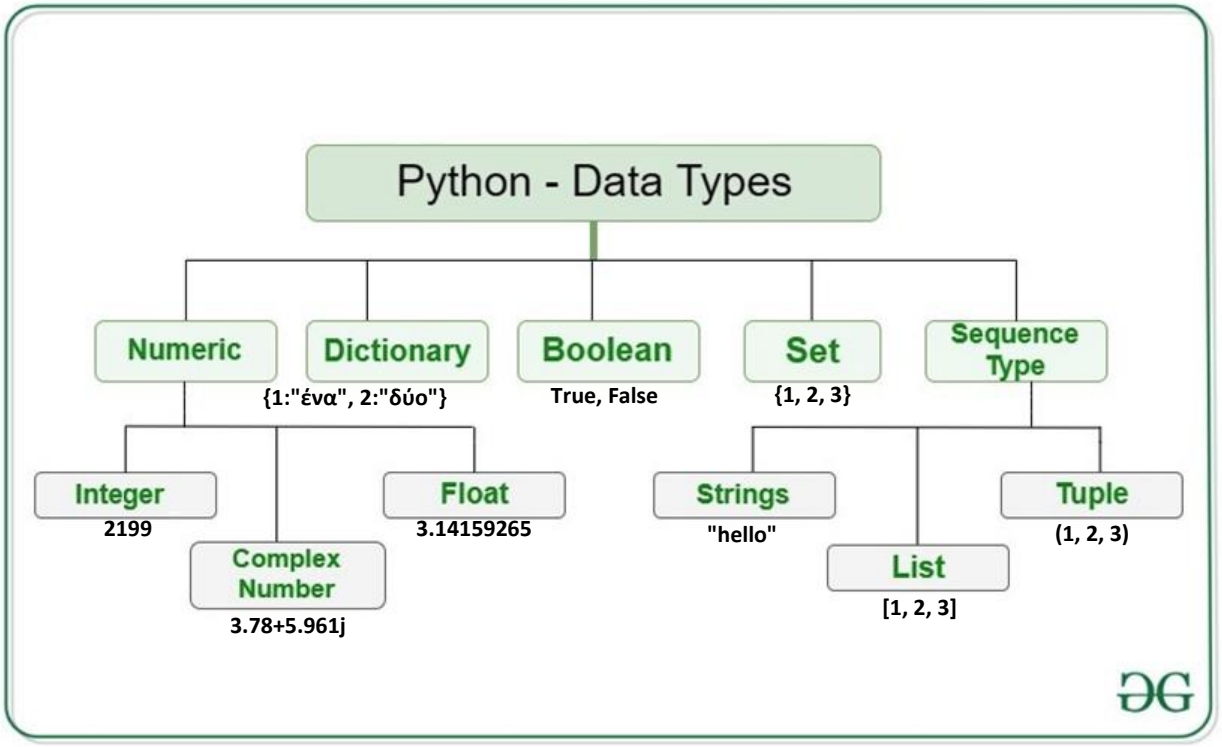
```
>>> type(s1)
<class 'str'>
>>> id(s1)
140473931225104
```



Τύποι δεδομένων - μεταβλητές

Τύποι δεδομένων στη γλώσσα Python

Αριθμητικοί τύποι (Numeric), Λογικοί τύποι (Boolean), Συμβολοσειρές (Strings) Λίστες (Lists), Πλειάδες (Tuples), Λεξικά (Dictionaries), Σύνολα (Sets) .



Οι τύποι στην Python

Τύπος	Ονομασία	Παράδειγμα
Ακέραιοι	int	1, 0, –1, 496
Κινητής Υποδιαστολής	float	3.14159, 0.5, 3.0
Λογικές	bool	True, False
Αλφαριθμητικά	str	“Babylon5”, “Star Trek”
Λίστες	list	[6, 28, 496], [“a”, “34w”, 45, True]
Πλειάδες	tuple	(“Star”, “Trek”)
Απουσία τιμής	None	None

Τελεστές

Αριθμητικοί

+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Υπόλοιπο (mod)
**	Δύναμη
//	Ακέραια διαίρεση

Συγκριτικοί

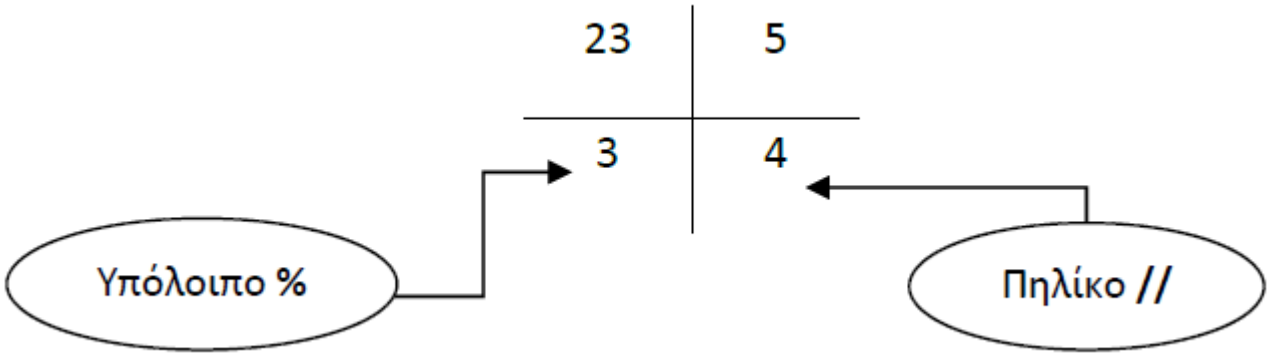
==	ισότητα
!=	≠
<	<
>	>
<=	≤
>=	≥

Λογικοί

and	και
or	ή
not	όχι

παντός τύπου

+ *



Μεγάλοι… αριθμοί

- Οι ακέραιοι είναι οσοδήποτε μεγάλοι
- Οι κινητής υποδιαστολής όχι

```
>>> 2.0 ** 1024

>>> 2.0 ** 1023

>>> 2**1024
```

Οι τύποι στην Python

- Οι ακέραιοι είναι οσοδήποτε μεγάλοι
- Οι κινητής υποδιαστολής όχι

```
>>> 2.0 ** 1024
    OverflowError: (34, 'Result too large')
>>> 2.0 ** 1023
8.98846567431158e+307
>>> 2**1024
179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
```

Οι τύποι στην Python

```
>>> 2 + 2.0
4.0
>>> type( 5 )
<class 'int'>
>>> type( 3.14159 )
<class 'float'>
>>> type( 2/2 )
<class 'float'>
>>> 2 / 2
1.0
```

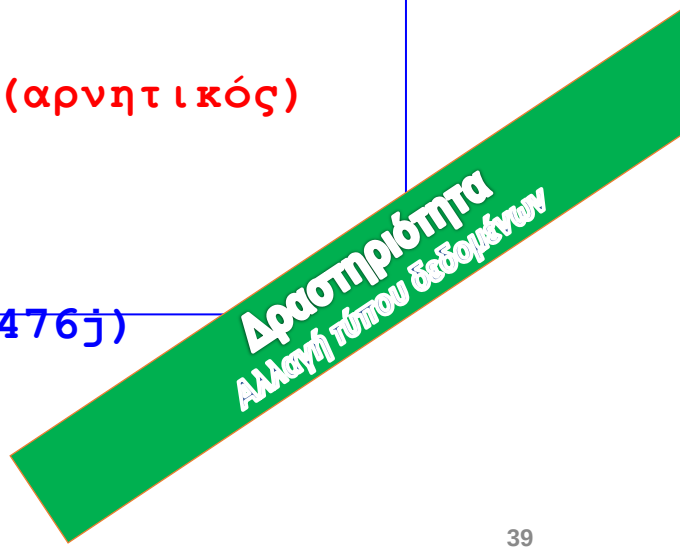
```
>>> 2 + 2
4
>>> 7 % 3
1
>>> 7 // 3
2
>>> 7.0 / 3
2.33333
```

```
>>> type( True )
<class 'bool'>
>>> type( "Python" )
<class 'str'>
>>> "Monty" + "Python"
'MontyPython'
>>> 3 * "Python"
'PythonPythonPython'
```

Αυτόματη αλλαγή τύπου

Η python **αλλάζει ΑΥΤΟΜΑΤΑ τον τύπο δεδομένων (*δείχνει σε άλλη τιμή*)** των ορισμάτων προκειμένου να γίνει εφικτή η αποτίμηση της παράστασης (*ένα πιο εντυπωσιακό παράδειγμα*):

```
>>> x = 1          # ακέραιος
>>> x = x/2        # έγινε πραγματικός
>>> print(x)
0.5
>>> x = -x
>>> print(x)       # επίσης πραγματικός (αρνητικός)
-0.5
>>> x = x**(1/2)   # τετραγωνική ρίζα
>>> print(x)       # έγινε μιγαδικός
(4.329780281177467e-17+0.7071067811865476j)
```



Προτεραιότητα πράξεων

Η προτεραιότητα των αριθμητικών πράξεων ακολουθεί τη φυσιολογική ροή της άλγεβρας:

Παρενθέσεις -> ύψωση σε δύναμη -> πολλαπλασιασμός και διαίρεση -> πρόσθεση και αφαίρεση.

Σε περίπτωση ίδιας προτεραιότητας, οι πράξεις εκτελούνται από τα αριστερά προς τα δεξιά (με εξαίρεση την ύψωση σε δύναμη).

2 * (3-1) δίνει **4** και **(1+1)**(5-2)** δίνει **8**.

2**1+1 δίνει 3 (και όχι 4)

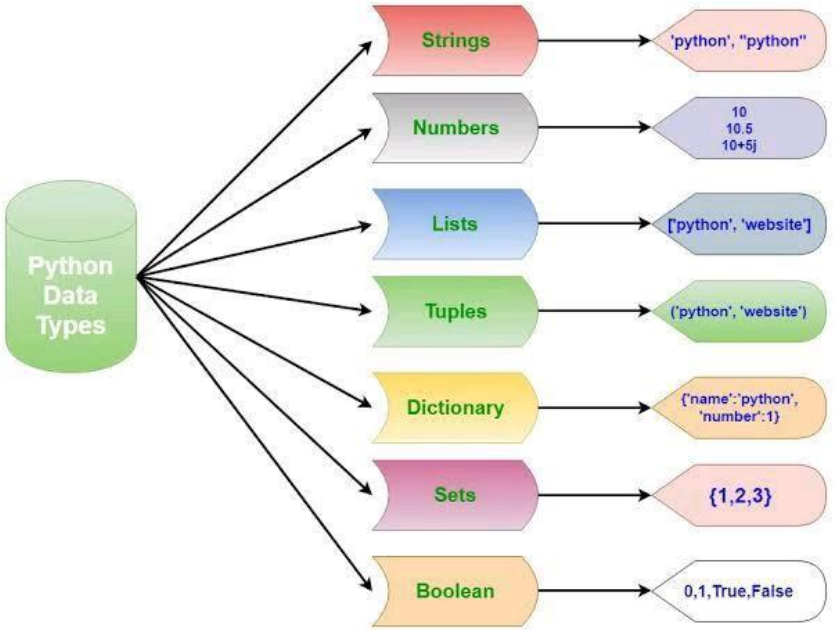
2*3-1 δίνει **5**, 6-3+2 δίνει **5** (και όχι 1)

Ωστόσο, το **2**1**2** δίνει **2** (και όχι 4)

Αλλαγή τύπου μεταβλητής

Οι μεταβλητές μπορούν να **αλλάζουν τύπο** εντός του ίδιου προγράμματος:

```
>>> x=5
>>> type(5)
    <class 'int'>
>>> type(x)
    <class 'int'>
>>> x=5.4
>>> type(x)
    <class 'float'>
>>> x='Συμβολοσειρά'
>>> type(x)
    <class 'str'>
```



Μετατροπές μεταξύ τύπων

- **type(<expression>)**: Επιστρέφει τον τύπο της τιμής της έκφρασης ή αν είναι μεταβλητή την τιμή στην οποία αναφέρεται. Χρησιμοποιείται το όνομα class επειδή η Python είναι αντικειμενοστρεφής γλώσσα
- Επειδή η Python έχει ισχυρό σύστημα τύπων (strongly typed) δεν γίνονται έμμεσες μετατροπές, π.χ. αν γράψουμε 1 + “2” θα θεωρηθεί συντακτικό λάθος.
- Πολλές φορές χρειάζεται να προβούμε άμεσα σε μετατροπή μιας τιμής από έναν τύπο σε έναν άλλο.
- Αυτό γίνεται με το όνομα του τύπου και την τιμή που θέλουμε σε παρενθέσεις:

<όνομα τύπου>(τιμή)

Μετατροπές μεταξύ τύπων: *Παραδείγματα*

```
>>> float( “2.71828” )
2.71828
>>> int( “2.71828” )
Value error: Invalid literal for int()..
>>> int( float( “2.71828” ) )
2
>>> 1 + 0 + 1
2
>>> str( 1 ) + str( 0 ) + str( 1 )
“101”
```

```
>>> int( “1024” ) + 6
1030
>>> str( 1024 * 4)
“4096”
>>> str( 1024 )*4
“10241024102410241024”
>>> str( 1024 )*4  + str(8)
“102410241024102410248”
```

Μεταβλητές

```
>>> x = 2**30
1073741824
>>> type( x )
<class 'int'>
>>> x = x + 0.0
<class ‘float'>
```

```
>>> x = “Monty Python”
>>> type( x )
<class ‘str'>
>>> x = False
>>> type( x )
<class ‘bool'>
```

- Μια μεταβλητή μπορεί να αλλάζει τύπο !!! (ή μήπως δεν είναι ακριβώς έτσι;)
- Οι τύποι δεν συνδέονται με τις μεταβλητές αλλά με τις τιμές
- Δεν δηλώνουμε μεταβλητές
- Η μεταβλητή ορίζεται με την πρώτη εκχώρηση!!!

Τύποι δεδομένων στη γλώσσα Python

Μεταβλητές

- Τα ονόματα μεταβλητών **ξεκινούν** πάντα **με γράμμα** (ή underscore _)
- Στα ονόματα μεταβλητών υπάρχει **διάκριση πεζών/κεφαλαίων**
- Επιτρέπονται και ελληνικοί χαρακτήρες στα ονόματα μεταβλητών (Προσοχή! Δεν είναι πάντα επιθυμητό.)
- Τα ονόματα μεταβλητών μπορούν να περιέχουν γράμματα, ψηφία και κάτω παύλα (underscore)
- Δεν απαιτείται ρητή δήλωση του τύπου δεδομένων μιας μεταβλητής
- Μια μεταβλητή ορίζεται κατά την εκχώρηση τιμής σε αυτή ('=') και ο τύπος της προκύπτει από τον τύπο του αντικειμένου που προκύπτει από την εντολή εκχώρησης.

Μεταβλητές: Εντολή ανάθεσης

>>> <Όνομα Μεταβλητής> = <έκφραση>

- Μια μεταβλητή λέμε ότι "*δένεται*" (*bind*) με μια τιμή
- Οι τύποι δεν συνδέονται με τις μεταβλητές αλλά με τις τιμές
- Δεν δηλώνουμε μεταβλητές
- Η μεταβλητή ορίζεται με την πρώτη εκχώρηση!!!
- Δεν επιτρέπεται αριστερά του = να υπάρχει έκφραση
- Το όνομα κάθε μεταβλητής αρχίζει με γράμμα ή _ .

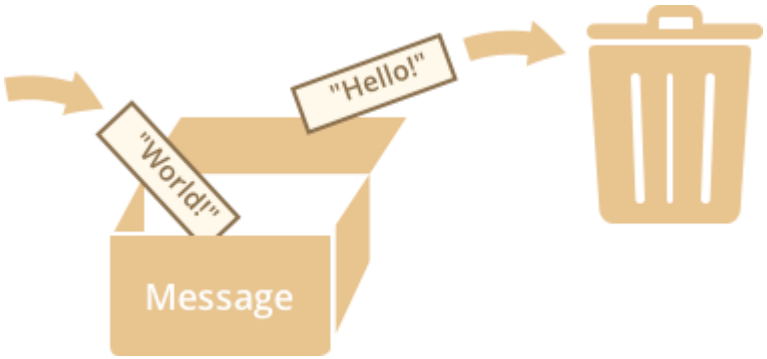
Μεταβλητές: Εντολή Ανάθεσης

>>> Message = “Hello!”

Μεταβλητές: Εντολή Ανάθεσης

>>> Message = “Hello!”

>>> Message = “World!”

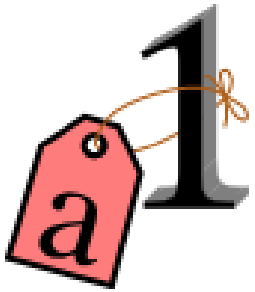
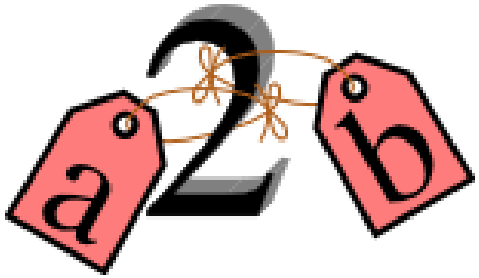


Μεταβλητές και μνήμη

Η συνάρτηση **id** επιστρέφει έναν μοναδικό αριθμό για κάθε αντικείμενο (μεταβλητή ή σταθερά) (μνήμη??)

```
>>> x = 28; y = 28; z = 28
>>> id( 28 ), id( x ), id( y ), id( z )
(10479276, 10479276, 10479276, 10479276)
>>> x = 10
(10479276, 10479492, 10479276, 10479276)
>>> id( 10 ), id( 11 ), id( 120 ), id( 121 )
(10479492, 10479480, 10480156, 10480144)
```

Αναπαράσταση μεταβλητών στην Python

Εντολές	Αποτέλεσμα στη μνήμη	
	Άλλες γλώσσες	Python
a = 1		
a = 2		
b = a		

Ιδιώματα που κάνουν τη ζωή μας εύκολη

```
>>> x = y = z = 28     # αυτό είναι ένα σχόλιο γραμμής
>>> x, y, z = 28, 6, 496     # πολλές αναθέσεις τιμής
>>> x = 100
>>> x, y = y, x          # αντιμετάθεση με πλειάδες
>>> z > w > x
False
>>> x += 10              # x = x + 10
'''
Σχόλιο που εκτείνεται σε πολλές γραμμές
'''
```

Ιδιώματα που κάνουν τη ζωή μας δύσκολη

```
>>> x = y = z = []
>>> print(x, y, z)
[] [] []
>>> x.append(10)
>>> y.append(20)
>>> z.append(30)
>>> print(x, y, z)
[10, 20, 30] [10, 20, 30] [10, 20, 30]
```

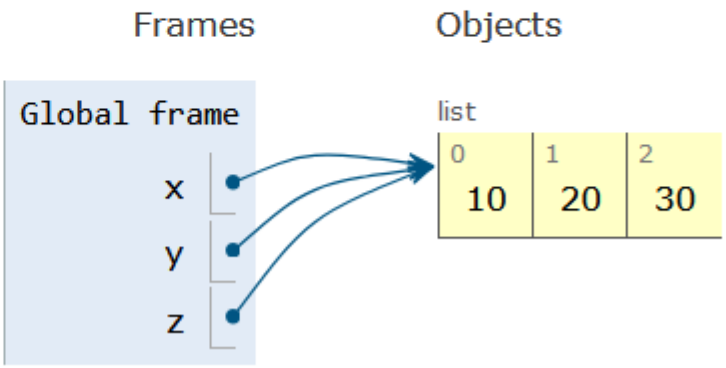
Python 3.6
known limitations

```
1 x = y = z = []
2 print(x, y, z)
3 x.append(10)
4 y.append(20)
5 z.append(30)
6 print(x, y, z)
```

Edit this code

Print output (drag lower right corner to resize)

```
[] [] []
[10, 20, 30] [10, 20, 30] [10, 20, 30]
```



Εισαγωγή δεδομένων από τον χρήστη

>>> < Όνομα Μεταβλητής > = input (< μήνυμα >)

```
>>> name = input("Πως σε λένε; ")
Πως σε λένε; Οδυσσέα
>>> print ( "Καλημέρα ", name )
Καλημέρα Οδυσσέα
```

Εισαγωγή δεδομένων από τον χρήστη ΙΙ

```
>>> x = input(" x =  ")
x = 1
>>> y = input(" y =  ")
y = 2
>>> z = input(" z =  ")
z = 3
>>> s = x + y + z
>>> print ( s )
122333
>>> s
'122333'
```

Προσοχή! *Η input επιστρέφει αλφαριθμητικό.*

Εισαγωγή δεδομένων από τον χρήστη ΙΙΙ

```
>>> x = int( input(“ x =  ”) )
x = 1
>>> y = int( input(“ y =  ”) )
y = 2
>>> z = int( input(“ z =  ”) )
z = 3
>>> s = x + y + z
>>> print ( s )
6
```

Χρήση συναρτήσεων από βιβλιοθήκες (modules)

- Ενσωμάτωση όλης τη βιβλιοθήκης ή
- Ενσωμάτωση μιας μόνο συνάρτησης

```
>>> import math
>>> math.sqrt(64)
8
>>> math.sqrt(–1)
Value error: math domain error
>>> math.floor(–4.1)
–5
>>> math.pi
3.14159265359
```

```
>>> import random
>>> random.randint(1,6)
5
>>> from cmath import sqrt
>>> sqrt(–1 )
1j
```

Ορισμός των δικών μας συναρτήσεων

Οι συναρτήσεις επιστρέφουν αποτέλεσμα με την εντολή **return** αλλιώς επιστρέφουν **None**
Ενσωμάτωση μιας μόνο συνάρτησης

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

Ποιο είναι το αποτέλεσμα των διπλανών εντολών?

```
def  sqr( a ):
   return  a * a
```

```
>>> sqr( 3 )
>>> square9 = sqr( 9 )
>>> sqr(sqr( 3 ) ) == 81
```

Προσοχή**!!!** Το σώμα της συνάρτησης οριοθετείται από τις εσοχές!!!

Οι συναρτήσεις είναι αντικείμενα

- Τι επιστρέφουν οι συναρτήσεις που δεν περιέχουν την εντολή return στο σώμα τους?

```
def printPython3():
    print ( 'Python'*3 )
>>> var = printPython3()
pythonpythonpython
>>> var
>>> print ( var )
None
```

```
>>> print ( printPython3() )
pythonpythonpython
None
>>> var = printPython3
>>> var
<function p3 at 0x013688B0>
>>> var()
pythonpythonpython
```

- Το **None** δηλώνει απουσία τιμής, όπως το **Null** στις ΒΔ.

Γνωριμία με το IDLE: Ο συντάκτης κώδικα

Ανοίξτε τον συντάκτη κώδικα επιλέγοντας από το μενού

File → New File και γράψτε το παρακάτω πρόγραμμα. Αποθηκεύστε το στον φάκελό σας ως test.py και εκτελέστε το με F5 ή με Run → Run Module. Προσοχή στις στοιχίσεις!!!

```
for i in range(1,10):
    for j in range(1,10):
        print ("%2i" % (i*j), i*j , end = "  ")
    print ( )
```

Options → Configure IDLE (ρυθμίσεις συντάκτη)

Format → Indent region (Ctrl +])

Options → Comment out region (Alt + 3)

Μεταφορά κώδικα ένα μπλοκ μέσα…

Επιλέγουμε όλο τον κώδικα και μετά Indent region μια φορά ή Ctrl +]

```
for i in range(1,10):
    for j in range(1,10):
        print ("%2i" % (i*j), i*j , end = "  ")
    print ( )
```

Format → Indent region (Ctrl +])

Το πρώτο μου υποπρόγραμμα ☺

Και ονομάζουμε το τμήμα κώδικα όπως θέλουμε π.χ. propaideia().
Μόλις τώρα ορίσαμε ένα υποπρόγραμμα !!!

```
def propaideia( )
    for i in range(1,10):
        for j in range(1,10):
            print (“%2i” % (i*j), i*j , end = “  ”)
        print ( )
```

Στη συνέχεια με F5 φορτώνουμε το υποπρόγραμμα στον διερμηνευτή και στη γραμμή εντολών δίνουμε

```
>>> propaideia( )
```

Καλή Πρακτική: *όλα σε 1 νοικοκυρεμένα* ☺

- Όταν τελειώσουμε την 1η άσκηση και ξεκινήσουμε την 2η ανοίγουμε νέο αρχείο ή ορίζουμε κάθε άσκηση σαν συνάρτηση
- Έχουμε όλες τις ασκήσεις σε ένα αρχείο, το φορτώνουμε μια φορά με F5 και εκτελούμε όποια θέλουμε από τη γραμμή εντολών.

```
def askisi_1( ):
    for i in range (1,10):
        print ( i*i  )
def askisi_2( ):
    for i in range (1,10):
        print ( i*i*i )
```

```
def askisi_3( ):
    sum = 0
    for i in range (1,10):
        sum = sum + i
    return   sum
```

```
>>> askisi_2( )
??
>>> askisi_1( )
??
>>> askisi_3( )
??
```

Η ώρα των αποφάσεων

Απλής Επιλογής

```
if <συνθήκη> :
    <Εντολές>
```

Σύνθετης Επιλογής

```
if <συνθήκη> :
    <Εντολές_1>
else :
    <Εντολές_2>
```

Πολλαπλής επιλογής

```
if <συνθήκη> :
    <Εντολές_1>
elif <συνθήκη2> :
    <Εντολές_2>
.....
else :
    <Εντολές_3>
```

Προσοχή**!!!** Οι εσοχές παίζουν ρόλο. Η στοίχιση μας δείχνει σε ποιο κλάδο ανήκει κάθε εντολή

Επιλέγουμε.... Δομή Επιλογής ☺

Σύνθετη Δομή Επιλογής

```
if  x  < 0 :
    print ( “αρνητικός” )
else :
    if  x < 10:
        print ( “μονοψήφιος” )
    else :
        if  x < 100:
            print ( “διψήφιος” )
 print ( “end if” )
```

Πολλαπλή Δομή Επιλογής

```
if  x  < 0 :
    print ( “αρνητικός” )
elif  x < 10:
    print ( “μονοψήφιος” )
elif  x < 100:
    print ( “διψήφιος” )
print ( “end if” )
```

Επιλέγουμε…. Δομή Επιλογής ☺

```
if  x  <= 0 :
    print ( “αρνητικός” )
    print ( “ή μηδέν” )
else :
    print ( “θετικός” )
print ( “αριθμός” )
```

```
if  x  <= 0 :
    print ( “αρνητικός” )
    print ( “ή μηδέν ” )
else :
    print ( “θετικός” )
    print ( “αριθμός” )
```

- **Προσοχή**!!! Η εσοχή του μπλοκ εντολών της if καθορίζει ποιες εντολές θα εκτελεστούν
- (**εσοχή = εμφωλευμένη δομή = 4 κενά ή tab**)
- **Κλασικό λάθος**: Ξεχνάμε το **“ : ”**
- Τι θα εμφανίσουν τα παραπάνω τμήματα κώδικα για x=0;

Υψηλού επιπέδου δυναμική γλώσσα

```
def maximum(a,b) :
    if a>b :
        return a
    else :
        return b
```

- Δυναμικό σύστημα τύπων
- Lazy Evaluation
- *Μην κάνεις σήμερα αυτό που μπορείς να αναβάλλεις για αύριο* ☺ (*Προσοχή! Δεν ισχύει για το διάβασμα στις εξετάσεις*)
- Πολύ υψηλού επιπέδου (ψευδογλώσσα)

```
>>> maximum(40,67)
67
>>> maximum(3.14, 2.718)
3.14
>>> maximum("Monty Python", "Monte Carlo")
'Monty Python'
```

Εντολή εισόδου

Εντολή εισόδου

Η ενσωματωμένη συνάρτηση **input()** χρησιμοποιείται για την **είσοδο δεδομένων** από το **πληκτρολόγιο**. Επιστρέφει την είσοδο σε μορφή **string**:

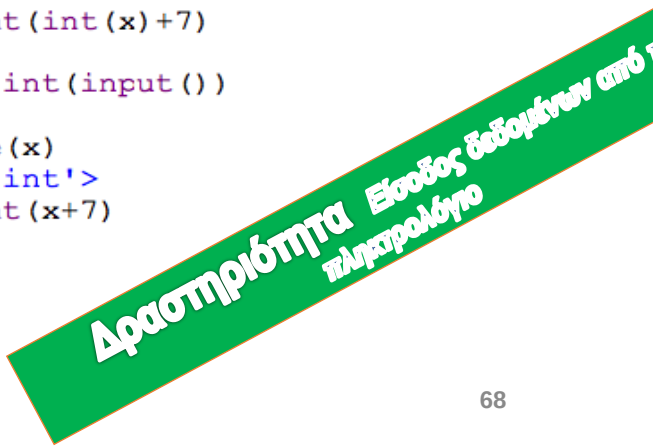
```
>>> x = input()
5
>>> print(x+7)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print(x+7)
TypeError: can only concatenate str (not "int") to str
```

Επειδή το x είναι string, δεν είναι δυνατή η πρόσθεση '5' + 7.
Εδώ θα αλλάξουμε τύπο, είτε διατηρώντας τον αρχικό τύπο της x:

```
>>> x = input()
5
>>> print(int(x)+7)
12
>>>
```

είτε αλλάζοντάς τον κατά την εκχώρηση:

```
>>> x = input()
5
>>> print(int(x)+7)
12
>>> x = int(input())
5
>>> type(x)
<class 'int'>
>>> print(x+7)
12
```



Εντολή εισόδου

Η input() μπορεί να συμπεριλαμβάνει ως όρισμα και ένα μήνυμα προς τον χρήστη:

```
# Ένα πρόγραμμα για υπολογισμούς σε κύκλο ακτίνας R
pi = 3.14  # Η τιμή του π

r = float(input("Δώστε την ακτίνα r του κύκλου:"))

perifereia = 2*pi*r
emvadon = pi*r**2

print("Η περιφέρεια του κύκλου με ακτίνα", r, "είναι:", perifereia)
print("Το εμβαδόν του κύκλου με ακτίνα", r, "είναι:", emvadon)
```

```
Δώστε την ακτίνα r του κύκλου:5
Η περιφέρεια του κύκλου με ακτίνα 5.0 είναι: 31.400000000000002
Το εμβαδόν του κύκλου με ακτίνα 5.0 είναι: 78.5
>>>
```

Εκτυπώσεις… (συνέχεια)

Είναι δυνατόν να καθορίσουμε τη **μορφοποίηση μεταβλητών εξόδου**, όπως για παράδειγμα τον **αριθμό των δεκαδικών ψηφίων** ενός πραγματικού αριθμού:

```
# Ένα πρόγραμμα για τον υπολογισμό της γεωμετρίας του κύκλου ακτίνας R
pi = 3.14 # η τιμή του π
r = float(input("Δώστε την ακτίνα r του κύκλου:"))
perifereia = 2*pi*r
emvadon = pi*r**2

print("Περιφέρεια", perifereia, end=" ") # Δεν αλλάζει η γραμμή
print("Εμβαδόν", emvadon)

# Μορφοποιημένη εκτύπωση
print("Ακτίνα:%5d, περιφέρεια:%5.2f και εμβαδόν: %4.1f" % (r, perifereia, emvadon))

print("Εναλλακτικά με fstring:")
print(f"Ακτίνα:{r:5.0f}, περιφέρεια:{perifereia:5.2f}, και εμβαδόν:{emvadon:4.1f}")
```

```
Δώστε την ακτίνα r του κύκλου:5
Περιφέρεια 31.400000000000002 Εμβαδόν 78.5
Ακτίνα:    5, περιφέρεια:31.40 και εμβαδόν: 78.5
Εναλλακτικά με fstring:
Ακτίνα:    5, περιφέρεια:31.40, και εμβαδόν: 78.5
```

Τύποι μορφοποίησης εμφάνισης

```
>>> the_name = "Petros"
>>> the_age = 20
```

- Μορφοποίηση με το σύμβολο %

```
>>> "Name: %s, Age: %s"%(the_name,the_age)
'Name: Petros, Age: 20'
```

- **Μορφοποίηση με την μέθοδο . format() της συμβολοσειράς**

```
>>> "Name: {}, Age: {}".format(the_name,the_age)
'Name: Petros, Age: 20'
```

- **Μορφοποίηση f-String**

```
>>> f"Name: {the_name}, Age: {the_age}"
'Name: Petros, Age: 20'
```

Είναι η προτεινόμενη μέθοδος για σύγχρονα προγράμματα

- **Με διαδοχική σύνδεση strings**

```
>>> "Name: " + the_name + ", Age: " + str(the_age)
'Name: Petros, Age: 20'
```

- **Με print**

```
>>> print("Name:", the_name, ", Age:", the_age)
Name: Petros , Age: 20
```

Συμβολοσειρές (str) - Τι είναι ...

- **Συμβολοσειρά** είναι μια ακολουθία χαρακτήρων που οριοθετείται από ζεύγη ειδικών συμβόλων.
- *Σύμβολα οριοθέτησης* συμβολοσειράς μπορεί να είναι: (α) τα μονά εισαγωγικά 'abc', (β) τα διπλά εισαγωγικά "abc", (γ) τρία διαδοχικά μονά εισαγωγικά '''abc''', (δ) τρία διαδοχικά διπλά εισαγωγικά """abc""".
- Η Python χρησιμοποιεί κωδικοποίηση Unicode.

```
# Μεταβλητή s αποκτά τύπο δεδομένων string
>>> s = 'This is an example'

# Τριπλά εισαγωγικά για συμβολοσειρές που
                        καταλαμβάνουν
#  πολλές γραμμές
>>> s2 = '''This is a
        multiline
        example'''
```

Συμβολοσειρές (str) - Οριοθέτηση ...

Διπλά και τριπλά εισαγωγικά. Παραδείγματα:

```
>>> str1 = 'abcd'
>>> str2 = "abcd"
>>> str3 = ''abcd''
SyntaxError: invalid syntax
>>> str4 = '''abcd'''
>>> str5 = """abcd"""
>>> str6 = '"abcd"'
>>> str7 = "'abcd'"
```

```
>>> print(str1)
abcd
>>> print(str2)
abcd
>>> print(str4)
abcd
>>> print(str5)
abcd
>>> print(str6)
"abcd"
>>> print(str7)
'abcd'
```

Έλεγχος/Απόφαση - Επανάληψη

1. **Έλεγχος και Απόφαση με** if
2. **Επανάληψη με** while
3. **Επανάληψη με** for … range

- **Μπλοκ εντολών - Στοίχιση**
- **Τελεστές σύγκρισης - Λογικές Τιμές - Λογικοί Τελεστές**
- **Σύνθετες Λογικές Εκφράσεις (Συνθήκες)**

ΑΠΛΟ ΟΛΟΚΛΗΡΩΜΕΝΟ ΠΡΟΓΡΑΜΜΑ ΜΕ ΔΟΜΗ ΕΛΕΓΧΟΥ - ΑΠΟΦΑΣΗ

```
# ΠΑΡΑΔΕΙΓΜΑ ΑΠΛΟΥ ΕΛΕΓΟΥ - ΣΥΝΘΗΚΕΣ (Conditions) με επανάληψη ερώτησης
while True:
    age=input('\nΠόσο χρονών είσαι; ')
    if len(age)==0:    # Αν ο χρήστης πατήσει Enter θα γίνει έξοδος από το πρόγραμμα
        print("\n --- Τέλος προγράμματος---")
        break                           # break or continue or fail
    if int(age)>=18:
        print('Είσαι ενήλικας')  # Αυτή γραμμή θα εμφανιστεί μόνο αν ισχύει η συνθήκη
    else:
        print('Είσαι ΑΝΗΛΙΚΟΣ !')
    print('='*20)  # Αυτή η γραμμή θα εκτελεστεί σε κάθε περίπτωση
```

- **Αμυντικός Προγραμματισμός**: Τι θα συμβεί αν … ο χρήστης δώσει … ένα μη θετικό ακέραιο αριθμό;

Επανάληψη - Αμυντικός προγραμματισμός

```
# Πρόγραμμα ελέγχου θετικού ακέραιου αριθμού από το χρήστη
print('\nΕΛΕΓΧΟΣ ΘΕΤΙΚΟΥ ΑΚΕΡΑΙΟΥ ΑΡΙΘΜΟΥ')
print('================================')
while True:
    try:
        x = int(input('\nΔώσε έναν θετικό ακέραιο αριθμό:'))
        if x<0:
            print('Ο αριθμός',x, 'δεν είναι θετικός\n')
        elif x>0:
            print('Ο αριθμός',x, 'ΕΙΝΑΙ θετικός')
        elif x==0:
            print('\n\nΔώσατε μηδέν (0) \nΕΞΟΔΟΣ ΑΠΟ ΤΟ ΠΡΟΓΡΑΜΜΑ')
            print('------------------------')
            break
    except:
        print('Το δεδομένο που εισαγάγατε δεν είναι ακέραιος αριθμός')
```

Αμυντικός προγραμματισμός με while

```
while True:
    a_str = input("αριθμητής = ")
    b_str = input("παρονομαστής = ")

    # Ελέγχουμε αν και οι δύο εισόδοι αποτελούνται μόνο από αριθμούς
    if a_str.isdigit() and b_str.isdigit():
        a = int(a_str)
        b = int(b_str)

        if b != 0:
            print("πηλίκο =", a/b)
            break
        else:
            print("Ο παρονομαστής δεν μπορεί να είναι μηδέν. Προσπάθησε ξανά.")
    else:
        print("Παρακαλώ εισάγετε ακέραιους αριθμούς.")
```

- Μπορεί να χρησιμοποιηθεί εναλλακτικά το .isdecimal διότι αποδέχεται η συνάρτηση int και μόνο αυτά.
- Δοκιμάστε το!

Δομή try - except

```
try:
    a = int(input("αριθμητής = "))
    b = int(input("παρονομαστής = "))
    print("πηλίκο =", a/b)
except:
    print("Ο υπολογισμός δεν μπορεί να γίνει.")
print("Τέλος υπολογισμού.")
```

```
αριθμητής = 4
παρονομαστής = 0
Ο υπολογισμός δεν μπορεί να γίνει.
Τέλος υπολογισμού.
```

```
αριθμητής = 3
παρονομαστής = k
Ο υπολογισμός δεν μπορεί να γίνει.
Τέλος υπολογισμού.
```

```
αριθμητής = 5
παρονομαστής = 2
πηλίκο = 2.5
Τέλος υπολογισμού.
```

Η try προσπαθεί να εκτελέσει τον κώδικα στο block και αν συμβεί σφάλμα εκτελεί την except.

Αμυντικός προγραμματισμός
(*defensive programming*)

Δραστηριότητα
Αμυντικός Προγραμματισμός

```python
from timeit import default_timer as timer

# Μη αναδρομικός αλγόριθμος υπολογισμού
# του Ν-οστού αριθμού Fibonacci
def fibonacci(N):
    fib1 = fib2 = fib = 1
    for i in range(3,N+1):
        fib = fib1+fib2
        fib1=fib2
        fib2=fib
    return fib
```

Η εντολή επανάληψης while

Οι Εντολές εκτελούνται για όσο ισχύει (=True) η Συνθήκη.

while <Συνθήκη> :

<Εντολές>

Η συνθήκη είναι μια λογική έκφραση η οποία έχει τιμή True ή False

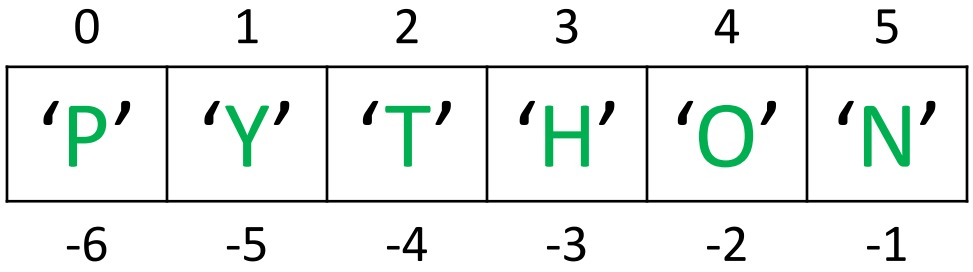
Ακολουθία βημάτων εκτέλεσης της εντολής while:

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

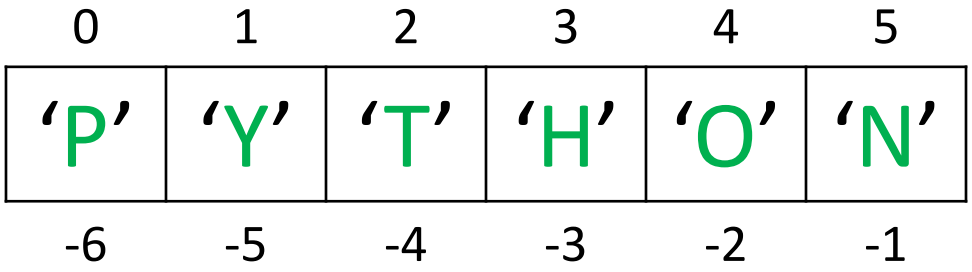
Αλφαριθμητικά




```
>>> w = "Python"
>>> len( w )
?
>>> print (w[5]+w[4]+w[2])
?
>>> w[5]+w[4]+w[2]
?
```

```
>>> print  ( w[-1]+w[-6] )
?
>>> "Python"[0]
?
>>>  str( 123 ) == '123' and int( '123' ) == 123
?
```

Αλφαριθμητικά




```
>>> w = “Python”
>>> len( w )
6
>>> print (w[5]+w[4]+w[2])
NOT
>>> w[5]+w[4]+w[2]
‘NOT’
```

```
>>> print  ( w[-1]+w[-6] )
NP
>>> “Python”[0]
‘P’
>>>  str( 123 ) == ‘123’ and int( ‘123’ ) == 123
True
```

Αλφαριθμητικά - Strings

```
>>> a = “one” ; b = “two”
>>> a < b
True
>>> 3 * (a + “ ” +b)
'one twoone twoone two‘
>>> lang = “python”
>>> lang[0] = “J”
TypeError: 'str' object does not
support item assignment
```

```
>>> len( a + b )
6
>>> “py” in “python”
True
>>> “tron” not in “python”
True
>>>  “python” == ‘python’
True
```

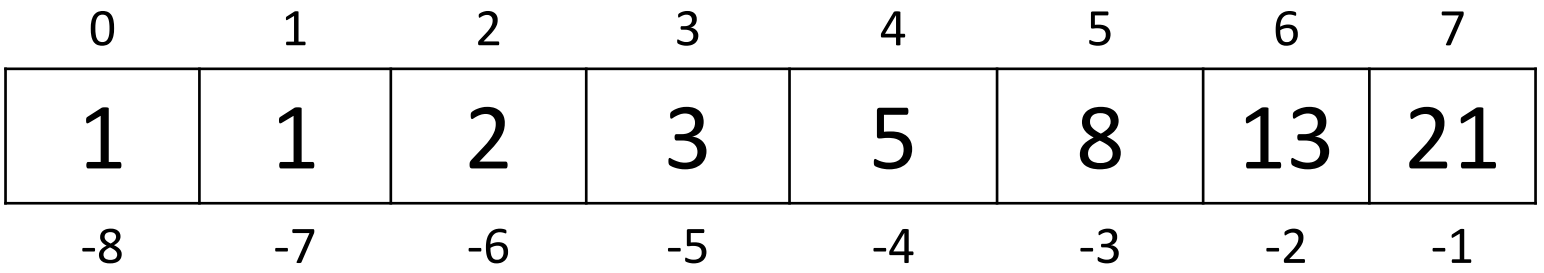
- Η προσπέλαση σε κάθε γράμμα του string γίνεται όπως σε έναν πίνακα, μόνο που δεν μπορούμε να τροποποιήσουμε τα περιεχόμενά του.

Δραστηριότητα: Έλεγχος email

- Να γράψετε μια συνάρτηση isEmail(email) η οποία θα δέχεται μια συμβολοσειρά και θα ελέγχει αν αποτελεί ελληνική διεύθυνση ηλεκτρονικής αλληλογραφίας, δηλαδή πρέπει να περιέχει τον χαρακτήρα '@' και να έχει κατάληξη '.gr'

```
def isEmail( email ):
    m = len ( email )
    return   email[m-3] + email[m-2] + email[m-1] == '.gr'  \
             and  '@'  in  email
```

Η δομή δεδομένων Λίστα




```
>>> fib = [1, 1, 2, 3, 5, 8, 13, 21]
>>> len( fib )
8
>>> print  ( fib[-1]+fib[-2] )
34
>>> fib = fib + [ fib[6] + fib[7] ]
>>> str( fib )
'[1, 1, 2, 3, 5, 8, 13, 21, 34]'
```

```
>>> len( fib )
9
>>> 34 in fib
True
>>> list( "abc" )
['a', 'b', 'c']
```

Η εντολή επανάληψης for για διάσχιση λίστας

Οι Εντολές εκτελούνται για κάθε στοιχείο της λίστας.

```
for item in List :
    <Εντολές>
```

Οι Εντολές εκτελούνται τόσες φορές όσες είναι και τα στοιχεία της λίστας

```
strangeList = [6, False, "False", [ 0,1 ], [ ], [ [ ] ] ]
for element in strangeList:
    print( 3*element, end = "  " )

18  0  FalseFalseFalse  [0,1,0,1,0,1]  [ ]  [ [ ], [ ], [ ] ]
```

Ο συναρτησιακός τελεστής list

Ο τελεστής list δέχεται ένα αντικείμενο και το μετατρέπει σε λίστα εφόσον η μετατροπή αυτή ορίζεται.

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

Η συνάρτηση range

Η συνάρτηση *range(A, M, B)* επιστρέφει μια λίστα αριθμών από το Α μέχρι το Μ με βήμα Β. Το Μ δεν συμπεριλαμβάνεται στη λίστα. Πρέπει όμως να εφαρμόσουμε τον τελεστή *list* για να δημιουργηθεί η λίστα.

```
>>> range( 5 )
range(0,5)
>>> list( range(5) ) )
[0, 1, 2, 3, 4]
>>> list( range(1, 10, 2) )
[1, 3, 5, 7, 9]
>>> list( range(10, 1, -2) )
[10, 8, 6, 4, 2]
```

```
>>> list( range(0) )
[ ]
>>> list( range(1) )
[ 0 ]
>>> list( range(–1) )
[ ]
>>> list( range(–100) )
[ ]
```

Η εντολή επανάληψης for

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

Οι Εντολές εκτελούνται για δεδομένο αριθμό επαναλήψεων .

Η εντολή αυτή χρησιμοποιείται όταν γνωρίζουμε εκ των προτέρων το πλήθος των επαναλήψεων

```
for index in range( A, M, B) :
    print(index)
```

```
index = A
while index < M :
    print(index)
    index = index + 1
```

Ισοδυναμία δομών for / while

Οι Εντολές εκτελούνται και στις δυο περιπτώσεις για τον ίδιο αριθμό επαναλήψεων. Όμως η τελική τιμή του index μετά το τέλος της επανάληψης δεν είναι η ίδια.

```
for  index  in range( 1, 10, 4)  :
      print(index, end = "  ")

print("τελική τιμή = ", index )
```

```
1  5  9
τελική τιμή =  9
```

```
index = 1
while  index < 10 :
     print(index, end = "  ")
     index = index + 4

print("τελική τιμή = ", index )
```

```
1  5  9
τελική τιμή =  13
```

Αλγόριθμος εύρεσης πρώτων αριθμών

Όλοι οι αριθμοί μικρότεροι του Ν είναι πιθανοί διαιρέτες του.

```
def isPrime(n):
    for j in range(2,n):
        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
```

Η **return** διακόπτει άμεσα την εκτέλεση της συνάρτησης και επιστρέφει την τιμή στο σημείο του προγράμματος από το οποίο κλήθηκε

Προσοχή στις στοιχίσεις

Αλγόριθμος εύρεσης πρώτων αριθμών ΙΙ

Αρκεί να ελέγξουμε μέχρι τη ρίζα του αριθμού! Μπορείτε να σκεφτείτε 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 η απόδοση της allPrimes βελτιώθηκε όμως η συνάρτηση δεν άλλαξε καθόλου!!!
Το χαρακτηριστικό αυτό λέγεται ανεξαρτησία των υποπρογραμμάτων

Εφαρμογή: Ταχυδρομικός Κώδικας

Να γράψετε μια συνάρτηση isPostalCode(pc) η οποία θα δέχεται μια συμβολοσειρά και θα ελέγχει αν αποτελεί πενταψήφιο ταχυδρομικό κώδικα.

```
def isPostalCode( pc ):
    digits = '0123456789'
    for symbol in pc:
        if symbol not in digits:
            return False
    return len( pc ) == 5
```

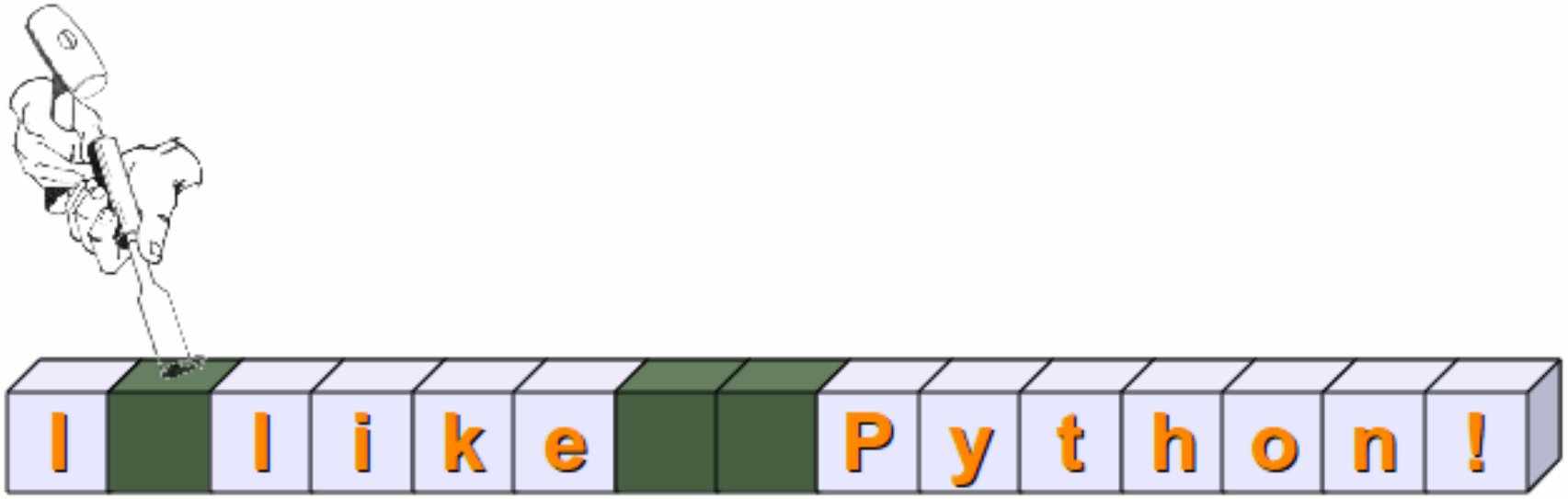
Εφαρμογή: Μετράμε φωνήεντα

Να γράψετε μια συνάρτηση count_vowels(word) η οποία θα δέχεται μια λέξη και θα μετράει και θα επιστρέφει το πλήθος των φωνηέντων της λέξης, π.χ.

count_vowels("Abracadabra Python") = 7

```
def count_vowels( word ):
    vowels = "AEIOUaeiou"
    count = 0
    for letter in word :          # παρατηρήστε τη χρήση του
        if letter in vowels:      # τελεστή in
            count += 1
    return count
```

Αλφαριθμητικό.split(Διαχωριστής)



Λίστες και αλφαριθμητικά

- Μπορούμε από ένα αλφαριθμητικό να πάρουμε μια λίστα από χαρακτήρες με τον τελεστή **list** ή μια λίστα από λέξεις με την μέθοδο **split**.

```
s = "python"
slist = list("python" )                  #  ['p', 'y', 't', 'h', 'o', 'n']
"a,b,c,d".split(',')                     #  ['a', 'b', 'c', 'd']
word = "ABRA CAD ABRA"
w_list = word.split()            #  ["ABRA", "CAD",  "ABRA"]
delimiter = "-"
w_str = w_list[0] +  '-' + w_list[1] +  '-' + w_list[2]
                                         #   "ABRA-CAD-ABRA"
w_str.split( delimiter )         #  ["ABRA", "CAD",  "ABRA"]
```

Λίστες και αλφαριθμητικά ΙΙ

Αντίστροφα μπορούμε να δημιουργήσουμε ένα αλφαριθμητικό από μια λίστα με την **join**.

```
>>> slist = [ ‘p’, ‘y’, ‘t’, ‘h’, ‘o’, ‘n’ ]
>>> delimiter = ‘’
>>> s = delimiter.join( slist )
>>> print( s )
python
>>> slist = [ ‘1’, ‘2’, ‘3’, ‘4’, ‘5’ ]
>>> s = ‘+’.join( slist )
‘1+2+3+4+5’
```

Πολλαπλή εισαγωγή δεδομένων

Όταν θέλουμε να εισάγουμε πολλά δεδομένα στην ίδια γραμμή σε μεταβλητές με μια εντολή χρησιμοποιούμε το παρακάτω ιδίωμα:

var1, var2, var3, … = ***input***().***split***()

```
>>> a, b, c, d = input().split()
3  5  8  13
>>> print( a + b, c + d )
'35'  '813'
>>> int( c ) + int( d )
21
```

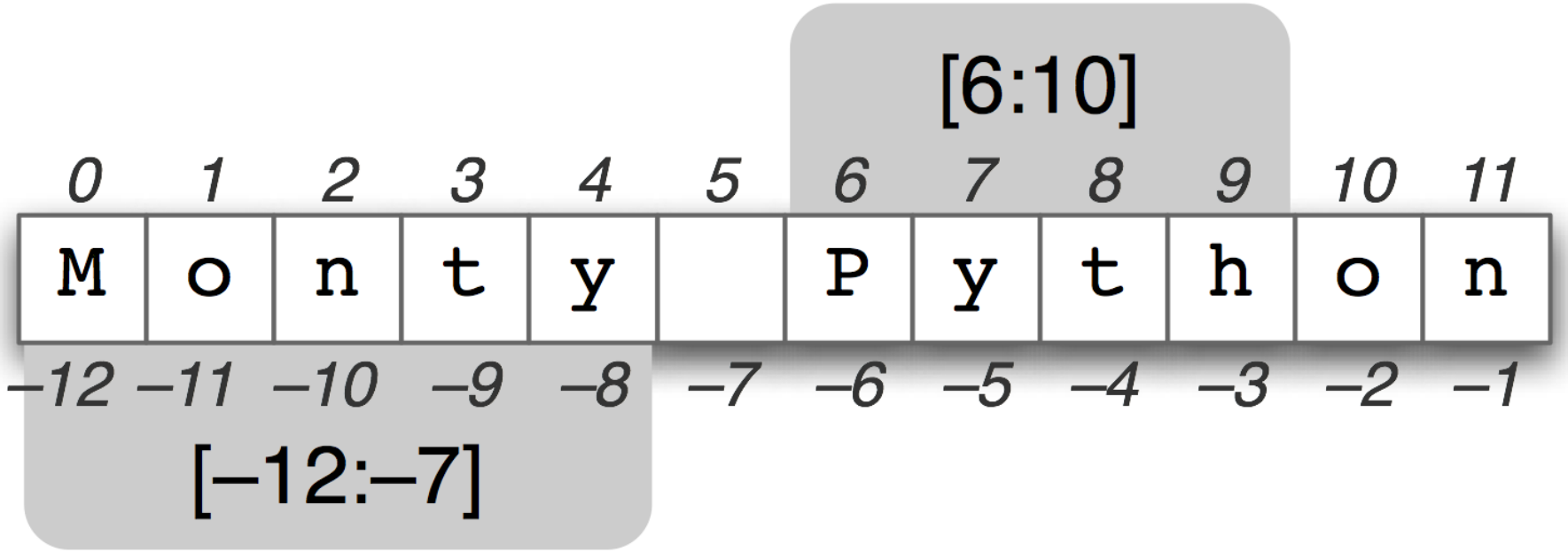
Εισαγωγή ημερομηνίας

Μια ημερομηνία εισάγεται ως αλφαριθμητικό. Αν θέλουμε να απομονώσουμε τον μήνα ή το έτος ως ακέραιο αριθμό σε μια μεταβλητή χρησιμοποιούμε την εξής εντολή

day, month, year = input().split()

```
>>> date = '07-05-2019'
>>> day, month, year = date.split("-")
>>> day + "/" + month + "/" + year
'07/05/2019'
>>> day, month, year = int(day), int(month), int(year)
>>> print(day, month, year)
7 5 2019
>>> print(day+1, month+1, year+1)
8 6 2020
```

Τελεστής κατάτμησης : για συμβολοσειρές



Πράξεις σε Λίστες

- Ισχύει και εδώ ο τελεστής κατάτμησης (:) όπως στα strings.
- Όμως μπορούμε να προσθέτουμε, και να αφαιρούμε στοιχεία
- Αν L = [ ], τότε L = L + [3] = [3]
- L + [4] = [3, 4]
- Ο τελεστής + συνενώνει λίστες σε μια **νέα λίστα**.

```
>>> fib = [1, 2, 3, 5, 8, 13, 21]
>>> fib[ : ] + fib[ : ]  == 2 * fib
True
>>> fib[0 : len (fib) : 2]
[1, 3, 8, 21]
>>>  fib  +=  [34]
```

```
>>> fib[-8: -4:1]
[1, 2, 3]
>>> fib[: : -1]
[21, 13, 8, 5, 3, 2, 1]
```

```
# fib ==[1, 2, 3, 5, 8, 13, 21, 34]
```

Αμυντικός προγραμματισμός

Σφάλματα χρόνου εκτέλεσης (runtime errors)

```
a = int(input("αριθμητής = "))
b = int(input("παρονομαστής = "))
print("πηλίκο =", a/b)
print("Τέλος υπολογισμού.")
```

Ανάλογα με την είσοδο, μπορεί να τρέξει σωστά:

```
αριθμητής = 5
παρονομαστής = 2
πηλίκο = 2.5
Τέλος υπολογισμού.
```

Πώς προστατεύουμε τον κώδικά μας από τέτοια σφάλματα;

… ή όχι:

```
αριθμητής = 4
παρονομαστής = 0
Traceback (most recent call last):
  File "/Users/eap/plipro/test.py", line 3, in
     <module>
     print("πηλίκο =", a/b)
ZeroDivisionError: division by zero
```

```
a = int(input("αριθμητής = "))
b = int(input("παρονομαστής = "))
if b==0:
    print("Ο υπολογισμός δεν μπορεί να γίνει.")
else:
    print("πηλίκο =", a/b)
print("Τέλος υπολογισμού.")
```

```
αριθμητής = 4
παρονομαστής = 0
Ο υπολογισμός δεν μπορεί να γίνει.
Τέλος υπολογισμού.
```

Σωστό!

```
αριθμητής = 3
παρονομαστής = k
Traceback (most recent call last):
  File "/Users/eap/plipro/test.py", line 2, in <module>
    b = int(input("παρονομαστής = "))
ValueError: invalid literal for int() with base 10: 'k'
```

Αμυντικός προγραμματισμός με while

```
while True:
    a_str = input("αριθμητής = ")
    b_str = input("παρονομαστής = ")

    # Ελέγχουμε αν και οι δύο εισόδοι αποτελούνται μόνο από αριθμούς
    if a_str.isdigit() and b_str.isdigit():
        a = int(a_str)
        b = int(b_str)

        if b != 0:
            print("πηλίκο =", a/b)
            break
        else:
            print("Ο παρονομαστής δεν μπορεί να είναι μηδέν. Προσπάθησε ξανά.")
    else:
        print("Παρακαλώ εισάγετε ακέραιους αριθμούς.")
```

- **Μπορεί να χρησιμοποιηθεί εναλλακτικά το .isdecimal διότι αποδέχεται η συνάρτηση int και μόνο αυτά.**
- **Δοκιμάστε το!**

Δομή try - except

```
try:
    a = int(input("αριθμητής = "))
    b = int(input("παρονομαστής = "))
    print("πηλίκο =", a/b)
except:
    print("Ο υπολογισμός δεν μπορεί να γίνει.")
print("Τέλος υπολογισμού.")
```

```
αριθμητής = 4
παρονομαστής = 0
Ο υπολογισμός δεν μπορεί να γίνει.
Τέλος υπολογισμού.
```

```
αριθμητής = 3
παρονομαστής = k
Ο υπολογισμός δεν μπορεί να γίνει.
Τέλος υπολογισμού.
```

```
αριθμητής = 5
παρονομαστής = 2
πηλίκο = 2.5
Τέλος υπολογισμού.
```

Η try προσπαθεί να εκτελέσει τον κώδικα στο block και αν συμβεί σφάλμα εκτελεί την except.

Αμυντικός προγραμματισμός
(*defensive programming*)

Προτεραιότητα σε σύνθετες προτάσεις

Η προτεραιότητα σε εκφράσεις όταν υπάρχουν πολλά είδη τελεστών είναι:

Αριθμητικοί -> Σύγκρισης -> Λογικοί

Σε τέτοιες εκφράσεις το αποτέλεσμα είναι πάντα λογικό (boolean).

Μία έκφραση με λογικό αποτέλεσμα λέγεται και συνθήκη.

```
>>> print(3+5<=8 and 2**2>10)
False
```

Πώς θα παραλείψουμε ένα κομμάτι του βρόγχου

- ```
  # ΠΑΡΑΔΕΙΓΜΑ ΑΤΕΡΜΟΝΗΣ ΕΠΑΝΑΛΗΨΗΣ ΜΕ ΠΑΡΑΛΕΙΨΗ ΕΝΤΟΛΩΝ
  # Η Continue χρησιμοιοποιείται αν θέλετε να πάτε στην επόμενη επανάληψη
  # αλλά χωρίς να εκτελεστούν κάποιες εντολές μέσα στο σώμα της επανάληψης

  i=0
  while i<100:
      i +=5
      if i > 20:
          continue
      print(i)
  print('\n--- Τέλος Επανάληψης ---')
  print(i)
  print('\nΕξοδος')
  ```
  ```

Επανάληψη μέσα σε επανάληψη (Φωλιασμένοι βρόχοι)

- Δημιουργήστε ένα πρόγραμμα που εκτυπώνει την προπαίδεια (1 X10 … 10 X 10)

```
# Επανάληψη μέσα στην επανάληψη (Φωλιασμένοι βρόχοι)
# Πίνακας Προπαίδειας

for i in range(1,11):
    for j in range (1,11):
    # Βάζω ένα στηλοθέτη για να τυπώσει μετά στην 8η στήλη
    # και το end='' για να παραμείνει σε αυτή τη θέση να μην αλλάξει γραμμή
        print (i*j,'\t',end='' )


    print('\n')
```

Παραδείγματα με την Εντολή επανάληψης for

```
s = 'The first example'
for c in s :
    print(c)
```

```
T
h
e

f
i
r
s
t

e
x
a
m
p
l
e
```

```
number = 1234
for digit in str(number) :
    print(digit)
```

```
1
2
3
4
```

Δομές Δεδομένων: Λίστες

Δομές Δεδομένων Λίστες

- **Οι λίστες είναι δομές δεδομένων οι οποίες μπορούν να περιέχουν οποιοδήποτε τύπο στοιχείων.**
  - `mylist = [1, 2, "a", 3.4]`
- **Αναφερόμαστε στα στοιχεία της λίστας μέσω της θέσης τους (οι δείκτες αρχίζουν από 0)**
- **Προσπέλαση τμημάτων λίστας με χρήση του τελεστή [start:end:step] (τεμαχισμός)**
- **Προσθήκη στοιχείων**

  Μέθοδος .append() : προσθέτει νέο στοιχείο στο τέλος της λίστας

  Μέθοδος .insert() : εισάγει νέο στοιχείο σε συγκεκριμένη θέση της λίστας.

  Μέθοδος .extend() : προσθέτει νέα λίστα στο τέλος μιας άλλης λίστας

  Τελεστής + : συνένωση δύο λιστών

  Τελεστής * ακέραιος : επανάληψη των στοιχείων λίστας
- **Αφαίρεση στοιχείων**

  Μέθοδος .pop() : αφαιρεί και επιστρέφει το τελευταίο στοιχείο της λίστας αν δεν δοθεί όρισμα ή το στοιχείο στη θέση που δίνεται ως όρισμα

  Μέθοδος .remove() : αφαιρεί συγκεκριμένο στοιχείο (την πρώτη του εμφάνιση εντός της λίστας), παίρνοντας ως όρισμα την τιμή του στοιχείου προς διαγραφή. Αν δεν υπάρχει το προς διαγραφή στοιχείο στη λίστα, προκαλείται σφάλμα.

  Τελεστής del li[θέση] : δίνοντας τη θέση του στοιχείου προς διαγραφή

Λίστες: Μέθοδοι και τελεστές

- **Προσθήκη** στοιχείων
  - Μέθοδος **.append() :** προσθέτει νέο στοιχείο στο *τέλος* της λίστας
  - Μέθοδος **.insert() :** εισάγει νέο στοιχείο σε *συγκεκριμένη θέση* της λίστας.
  - Μέθοδος **.extend() :** προσθέτει νέα λίστα στο τέλος μιας άλλης λίστας
  - Τελεστής **+** : συνένωση δύο λιστών
  - Τελεστής *** ακέραιος** : επανάληψη των στοιχείων λίστας

- **Αφαίρεση** στοιχείων
  - Μέθοδος **.pop() :** αφαιρεί και επιστρέφει το *τελευταίο στοιχείο* της λίστας αν δεν δοθεί όρισμα ή το στοιχείο στη θέση που δίνεται ως όρισμα
  - Μέθοδος **.remove() :** αφαιρεί *συγκεκριμένο στοιχείο* (την πρώτη του εμφάνιση εντός της λίστας), παίρνοντας ως όρισμα την τιμή του στοιχείου προς διαγραφή. Αν δεν υπάρχει το προς διαγραφή στοιχείο στη λίστα, προκαλείται σφάλμα.
  - Τελεστής **del** li[θέση] : δίνοντας τη *θέση του στοιχείου* προς διαγραφή

Πράξεις σε λίστες

```
>>> mylist = [“example”, 0, 1, 2]
>>> mylist = mylist + [2, 3]
>>> mylist
['example', 0, 1, 2, 2, 3]

>>> mylist.append(4)
>>> mylist
['example', 0, 1, 2, 2, 3, 4]

>>> mylist.extend( [4, 'test'] )
>>> mylist
['example', 0, 1, 2, 2, 3, 4, 4, 'test']

>>> mylist.remove(2)
>>> mylist
['example', 0, 1, 2, 3, 4, 4, 'test’]

>>> mylist.pop()
>>> mylist
['example', 0, 1, 2, 3, 4, 4]
```

Εμφωλευμένες λίστες

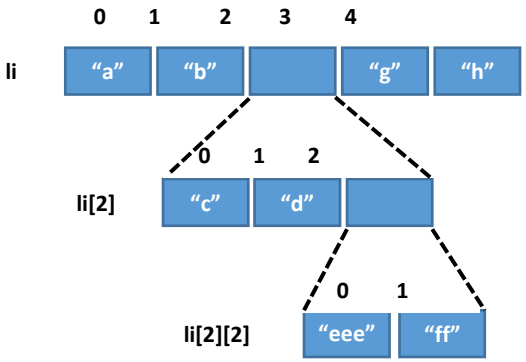
```
li = ['a', 'b', ['cc', 'dd', ['eee', 'fff']],
'g', 'h']

print(li[2])

print(li[2][2])

print(li[2][2][0])
```

```
['cc', 'dd', ['eee', 'fff']]
['eee', 'fff']
eee
```



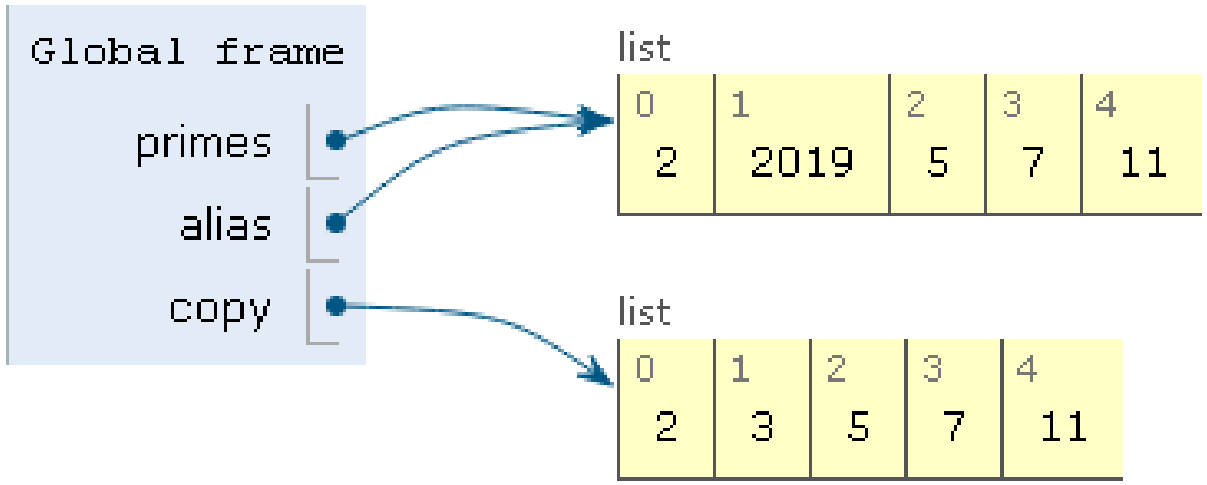
Δημιουργία αντίγραφου μιας λίστας

Η έκφραση primes[:] δημιουργεί ένα αντίγραφο της λίστας primes.

Ο τελεστής απόδοσης τιμής = , δεν δημιουργεί αντίγραφο. Αποτελεί αναφορά στην ίδια λίστα.

Python 3.6

```
1  primes = [2,3,5,7,11]
2  alias = primes
3  copy = primes[:]
4  alias[1] = 2019
5  print(alias)
6  print(primes)
7  print(copy)
```



Μέθοδος append:
Προσθήκη στοιχείου στο τέλος της λίστας

Λίστα.append(στοιχείο)

```
>>> numbers = [ ]                          # δημιουργία κενής λίστας
>>> numbers.append(8)                      # numbers == [ 8 ]
>>> numbers.append(13)                     # numbers == [ 8, 13 ]
>>> numbers.append( numbers[0] + numbers[1])
>>> print( numbers )
[ 8, 13, 21 ]
```

Μέθοδος append:

Προσθήκη στοιχείου στο τέλος της λίστας

Προσοχή!!!

Λίστα.append(στοιχείο) **≠** ***Λίστα = Λίστα + στοιχείο***

```
def add2List( List, item) :
    List = List + [ item ]


>>> List = [ 3, 5, 8 ]
>>> add2List( List, 13 )
>>> print( List )
[ 3, 5, 8 ]
```

```
def addToList( List, item) :
    List = List.append( item )


>>> List = [ 3, 5, 8 ]
>>> addToList( List, 13 )
>>> print( List )
[ 3, 5, 8, 13 ]
```

Μέθοδος append:
Προσθήκη στοιχείου στο τέλος της λίστας

```
def add2List( List, item) :
    List = List + [ item ]
# Δημιουργεί μια νέα λίστα
αντίγραφο της αρχικής και
προσθέτει το νέο στοιχείο item
στο τέλος της.

>>> List = [ 3, 5, 8 ]
>>> add2List( List, 13 )
>>> print( List )
[ 3, 5, 8 ]
```

```
def addToList( List, item) :
    List = List.append( item )

# Τροποποιεί τη λίστα
προσθέτοντας το νέο στοιχείο
item στο τέλος της.

>>> List = [ 3, 5, 8 ]
>>> addToList( List, 13 )
>>> print( List )
[ 3, 5, 8, 13 ]
```

Μέθοδος insert:
Προσθήκη στοιχείου σε συγκεκριμένη θέση στη λίστα

Λίστα.insert(θέση, στοιχείο)

Εισάγει το <**στοιχείο**> στη θέση <**θέση**> της λίστας

```
>>> numbers = [8, 21 ]
>>> numbers.insert(0, 5)      # numbers == [ 5, 8, 21 ]
>>> numbers.insert(2, 13)     # numbers == [ 5, 8, 13, 21 ]
>>> numbers.insert(len(numbers), 34)    # numbers.append(34)
```

fib.*insert*(len(fib), 34) === fib.*append*(34)

Μέθοδος pop:
Διαγραφή στοιχείου από τη λίστα

Διαγράφει το στοιχείο που βρίσκεται στη θέση **<θέση>** της λίστας και το επιστρέφει ως αποτέλεσμα

Λίστα.pop([**θέση**])

```
>>> fib = [3, 5, 8, 13, 21, 34, 55 ]
>>>  fib.pop( )     # Διαγραφή του τελευταίου στοιχείου (55)
>>>  fib.pop( 0 )  # Διαγραφή του πρώτου στοιχείου (3)
>>>  print( fib )
[5, 8, 13, 21, 34 ]
>>>  fib.pop( 3 )    # Διαγραφή του 4ου στοιχείου ( 21 )
>>>  fib.pop( len( fib )-1 )  # Διαγραφή του τελευταίου στοιχείου
```

Διαχωρισμός Λίστας

Να γράψετε ένα πρόγραμμα το οποίο δεδομένης μια λίστας ακέραιων αριθμών θα διαχωρίζει τους αριθμούς σε δυο νέες λίστες, μια για τους θετικούς και μια για τους αρνητικούς. Οι ομόσημοι αριθμοί δεν πρέπει να αλλάξουν σειρά μεταξύ τους.

```
numbers = [ 2, -1, 8, 4, -5, -7 ]
positives = negatives = [ ]   #bug
for  number in numbers :
    if  number > 0 :
        positives += [ number ]
    elif  number < 0 :
        negatives += [ number ]
```

Κανονικά θα έπρεπε μετά την εκτέλεση του προγράμματος οι λίστες να έχουν τις τιμές:

positives = [2, 8, 4,]

negatives = [-1, -5, -7]

όμως .....

και οι δυο είναι ίσες με την αρχική λίστα!

Διαχωρισμός Λίστας

```
numbers = [2, -1, 8, 4, -5, -7]
positives = negatives = []
for  number in numbers :
    if  number > 0 :
        positives += [ number ]
    elif  number < 0 :
        negatives += [ number ]
```

Και οι δυο μεταβλητές δείχνουν στην ίδια λίστα στη μνήμη λόγω της εντολής:

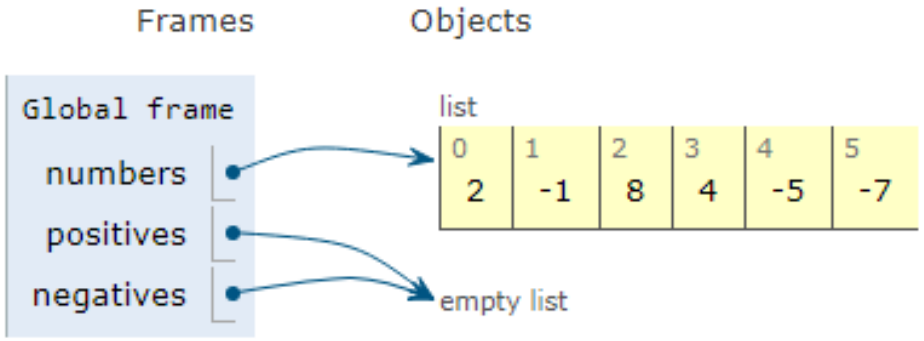
positives = negatives = [ ]

```
>>> print(positives)
[2, -1, 8, 4, -5, -7]
>>> print(negatives)
[2, -1, 8, 4, -5, -7]
>>> positives.pop()
-7
>>> positives.append(100)
>>> print(positives)
[2, -1, 8, 4, -5, 100]
>>> print(negatives)
[2, -1, 8, 4, -5, 100]
```

Διαχωρισμός Λίστας

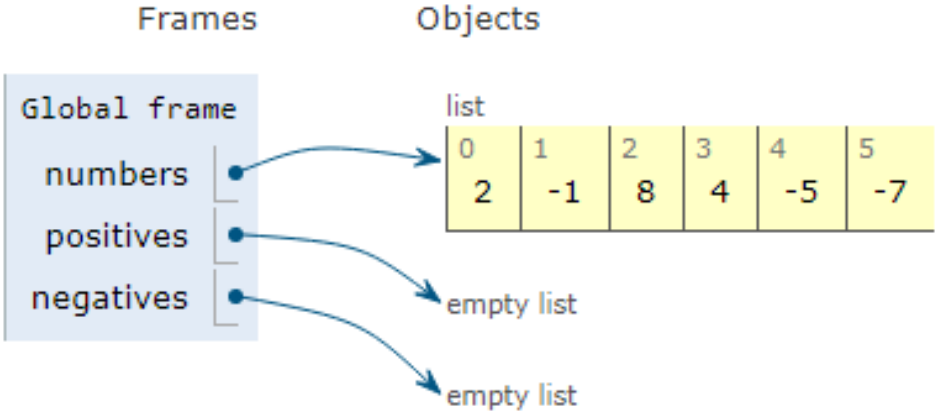
Python 3.6

```
1  numbers = [2, -1, 8, 4, -5, -7]
2  positives = negatives = []
3  for number in numbers:
4      if number > 0 :
5          positives += [ number ]
6      elif number < 0 :
7          negatives += [ number ]
```



Python 3.6

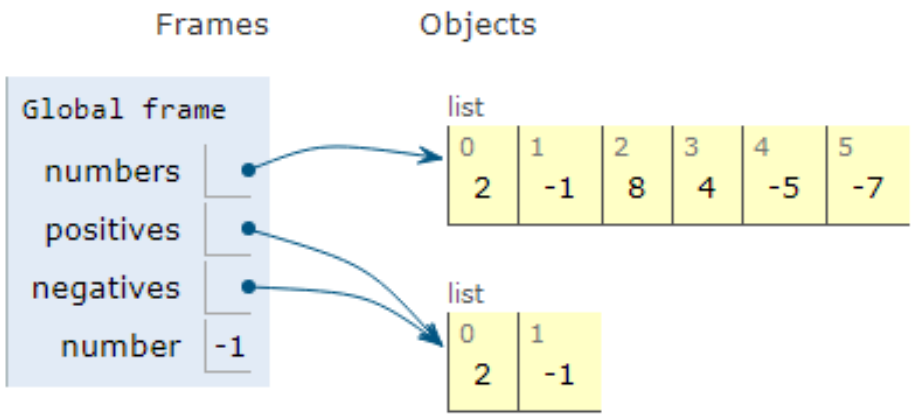
```
1  numbers = [2, -1, 8, 4, -5, -7]
2  positives = []
3  negatives = []
4  for number in numbers:
5      if number > 0 :
6          positives += [ number ]
7      elif number < 0 :
8          negatives += [ number ]
```



Διαχωρισμός Λίστας

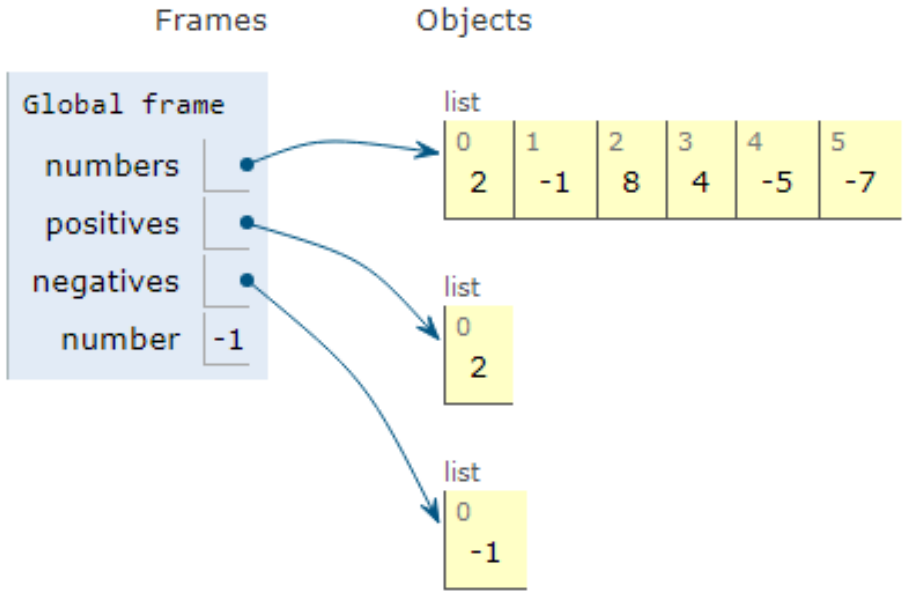
Python 3.6

```
1  numbers = [2, -1, 8, 4, -5, -7]
2  positives = negatives = []
3  for number in numbers:
4      if number > 0 :
5          positives += [ number ]
6      elif number < 0 :
7          negatives += [ number ]
```



Python 3.6

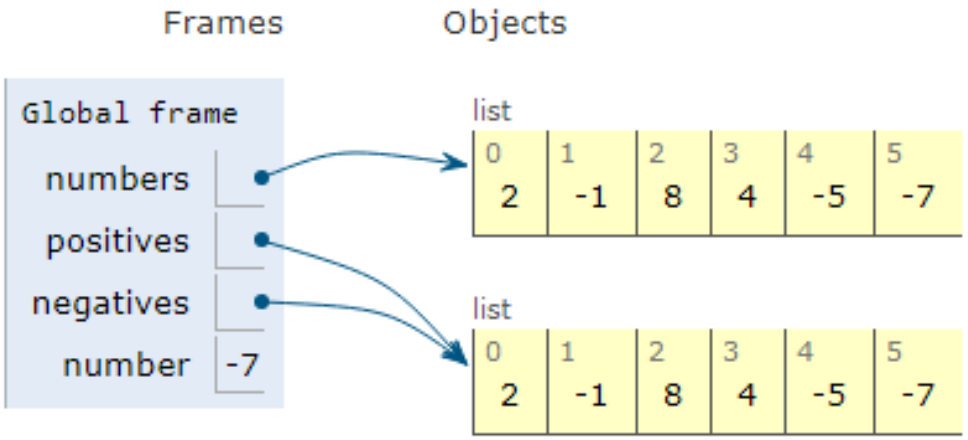
```
1  numbers = [2, -1, 8, 4, -5, -7]
2  positives = []
3  negatives = []
4  for number in numbers:
5      if number > 0 :
6          positives += [ number ]
7      elif number < 0 :
8          negatives += [ number ]
```



Διαχωρισμός Λίστας

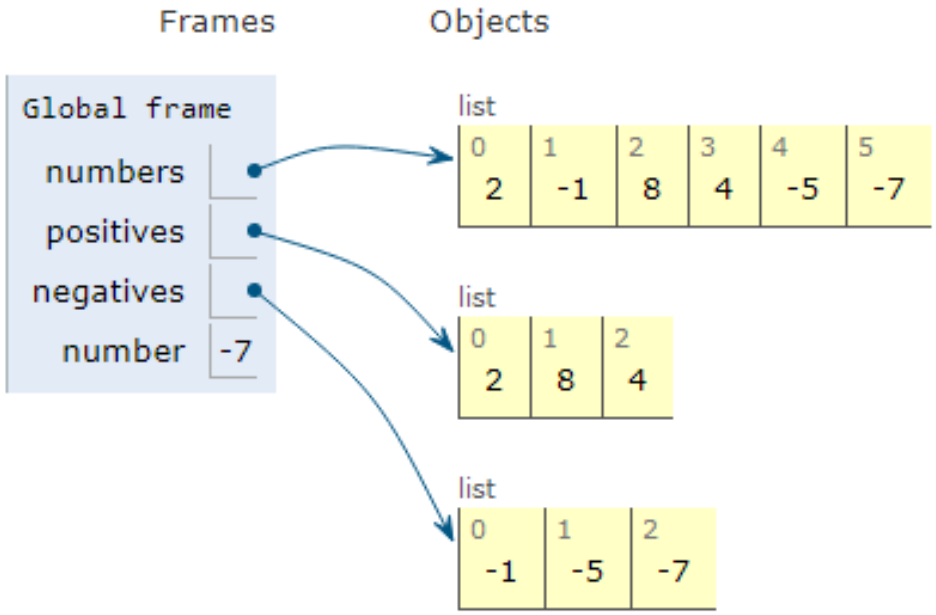
Python 3.6

```
1  numbers = [2, -1, 8, 4, -5, -7]
2  positives = negatives = []
3  for number in numbers:
4      if number > 0 :
5          positives += [ number ]
6      elif number < 0 :
7          negatives += [ number ]
```



Python 3.6

```
1  numbers = [2, -1, 8, 4, -5, -7]
2  positives = []
3  negatives = []
4  for number in numbers:
5      if number > 0 :
6          positives += [ number ]
7      elif number < 0 :
8          negatives += [ number ]
```

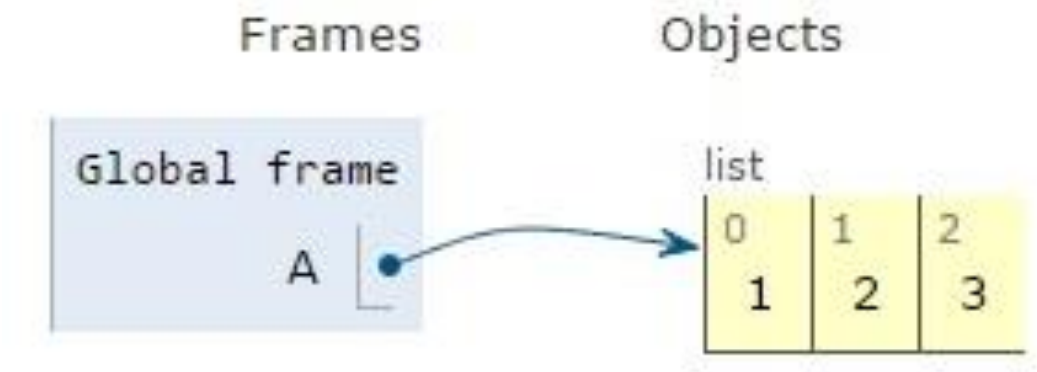
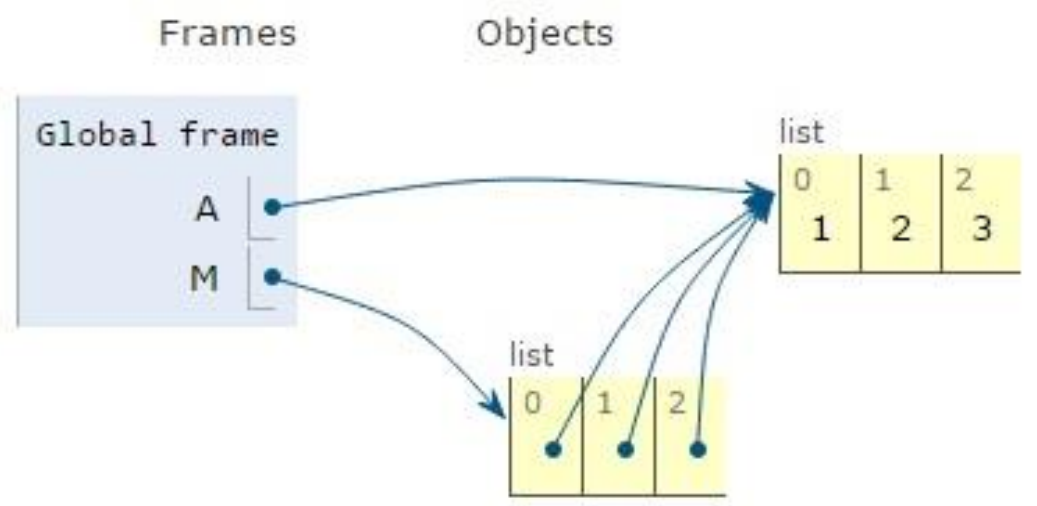


Διαχωρισμός Λίστας: Διόρθωση

```
numbers = [2, -1, 8, 4, -5, -7]
positives = []
negatives = []
for  number in numbers :
    if  number > 0 :
        positives += [ number ]
    elif  number < 0 :
        negatives += [ number ]
```

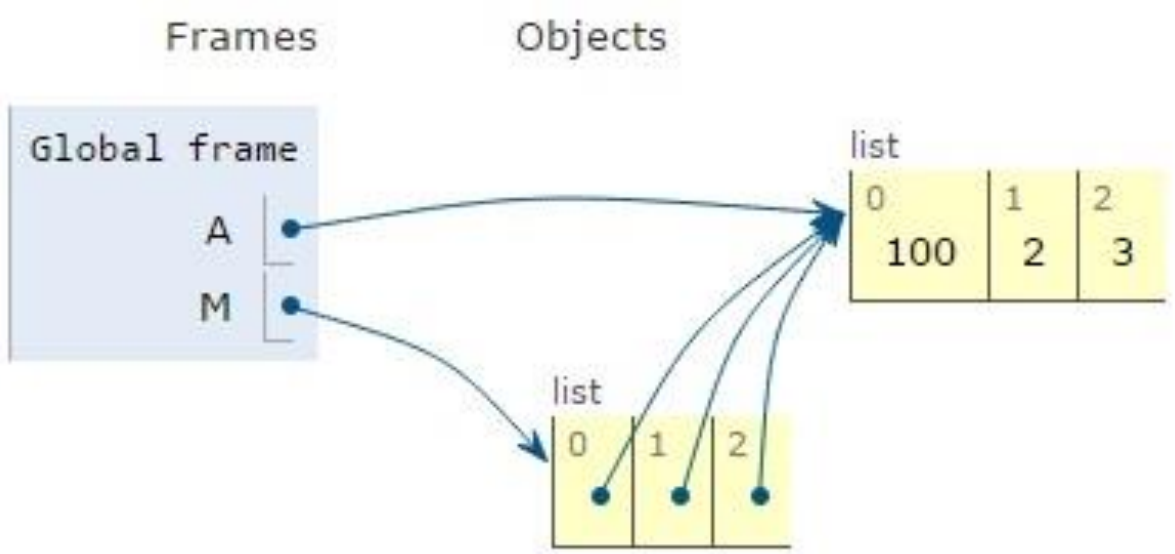
```
>>> print(positives)
[2, 8, 4]
>>> print(negatives)
[-1, -5, -7]
```

Λίστες και μνήμη I

A = [1, 2, 3]	
M = [A, A, A]	

Λίστες και μνήμη II

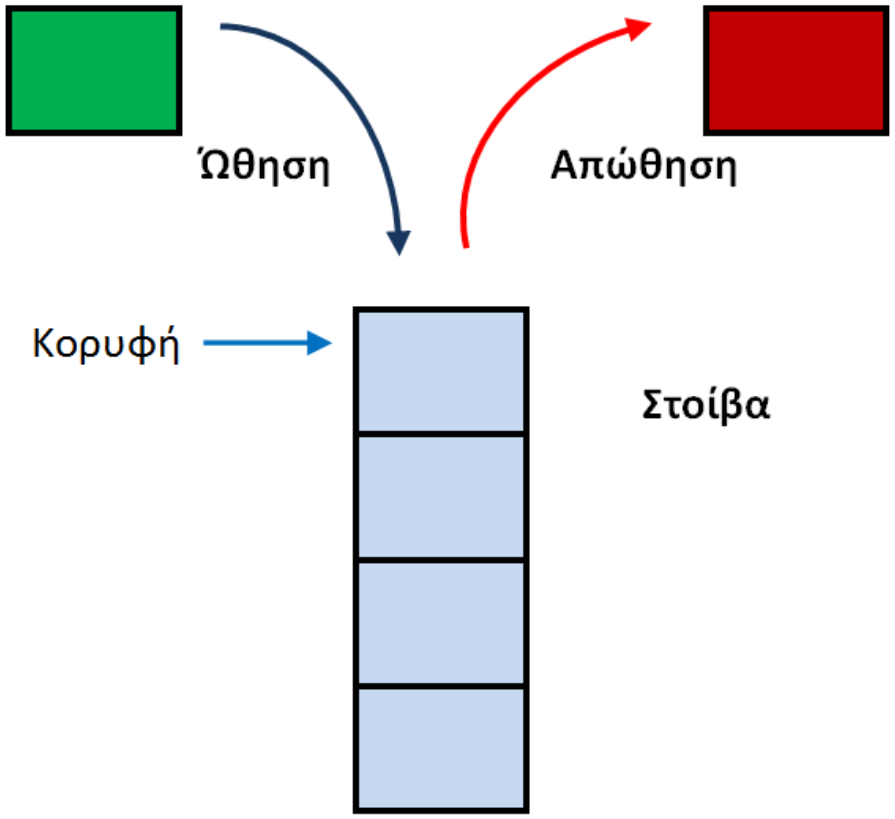
M[0][0] = 100




```
>>> A = [1, 2, 3]
>>> M = [A, A, A]
>>> M
[[1, 2, 3], [1, 2, 3], [1, 2, 3] ]
```

```
>>> A[0] = 100;  print( M )
[[100, 2, 3], [100, 2, 3], [100, 2, 3] ]
>>> M[0][0] = 28; print( M )
[[28, 2, 3], [28, 2, 3], [28, 2, 3] ]
```

Δομές Δεδομένων : Στοίβα




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

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

def  isEmpty(stack) :
    return len(stack) == 0

def  createStack( ) :
    return [ ]
```

Εφαρμογή: Αντιστροφή σειράς λέξεων

```
stack = createStack()     # Δημιουργία της στοίβας
word = input()           # Εισάγεται η πρώτη λέξη
while word != "" :       # μέχρι να δοθεί η κενή (enter)
    push(stack, word)     # ώθηση στη στοίβα
    word = input()        # εισαγωγή της επόμενης λέξης

while not isEmpty(stack):    # όσο έχει δεδομένα η στοίβα
    word = pop(stack)          # εξαγωγή της επόμενης λέξης
    print( word, end = "   " )  # εκτύπωση στην ίδια γραμμή
```

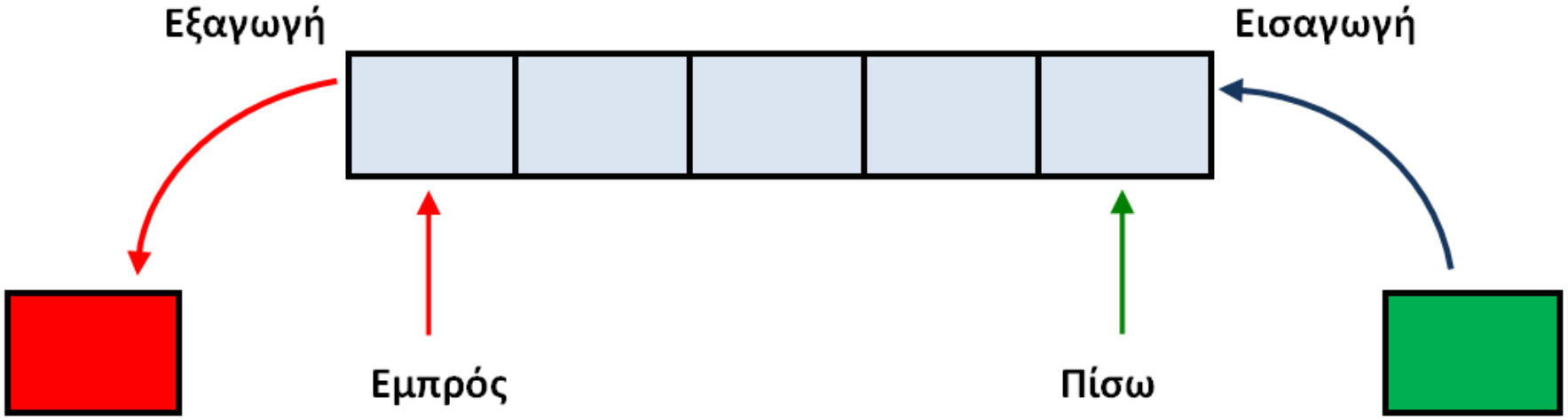
Σκεφτείτε με ποιον τρόπο υλοποιείται :

- Η λειτουργία **undo** σε όλες τις εφαρμογές
- Το **ιστορικό** του web browser

Στοίβα: Αντικειμενοστρεφής Υλοποίηση

```
class  Stack :
    def  __init__(self) :
        self.items = [ ]
    def  isEmpty(self) :
        return self.items == [ ]
    def  push(self, item) :
        self.items.append( item )
    def  pop(self) :
        return self.items.pop( )
```

Δομές Δεδομένων : Ουρά




```
def  enqueue(queue, item) :
    queue.append( item )

def dequeue(stack) :
    return queue.pop(0 )
```

```
def  isEmpty(queue) :
    return len(queue) == 0

def  createQueue( ) :
    return [ ]
```

Η δομή δεδομένων Πλειάδα *(Tuple)*

- Μια πλειάδα είναι μια ακολουθία αντικειμένων χωρισμένων με κόμματα, τα οποία **περικλείονται σε παρενθέσεις** οι οποίες όμως δεν είναι υποχρεωτικές.
- Οι πλειάδες είναι ακολουθίες όπως οι λίστες όμως **δεν μπορούν να τροποποιηθούν**.

```
>>> player = (“Magic”, “Johnson”, 32, 12, 10)
>>> player[3]*4
48
>>> player[2] = 100
TypeError: 'tuple' object does not support item assignment
```

Εφαρμογή: Πολλαπλές εκχωρήσεις στην ίδια γραμμή

day = 7
month = “May”
year = 2019

day, month, year = 7, “May”, 2019

Ουσιαστικά πρόκειται για την απόδοση τιμής σε μια πλειάδα

Εφαρμογή: Αμοιβαία αλλαγή των τιμών δυο μεταβλητών

```
temp = source
source = target
target = temp
```

```
source, target  =  target, source
```

Η έκφραση target, source είναι μια πλειάδα (tuple)

Εφαρμογή: Επιστροφή πολλαπλών τιμών από συνάρτηση

- Πολλές φορές θέλουμε μια συνάρτηση να επιστρέφει περισσότερες από μια τιμές. Έτσι επιστρέφει μια πλειάδα με τις τιμές αυτές.

```
def  powers( x ):
    return  x**2,  x**3
>>> square2, cube2  = powers( 2 )
```

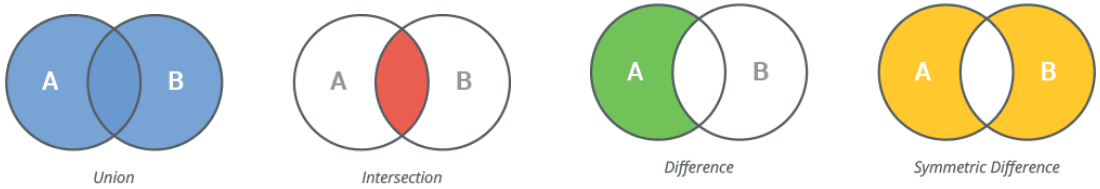
Σύνολα (Set)

- Δημιουργία συνόλου με τη χρήση αγκίστρων {} και διαχωρισμός στοιχείων/τιμών με κόμμα (,). Αρχικοποίηση κενού συνόλου με x = set()
- Διακριτά στοιχεία: μία τιμή εμφανίζεται μία μόνο φορά στο σύνολο.
- Μη διατεταγμένη δομή: Δεν υπάρχει διάταξη/σειρά μεταξύ των στοιχείων του συνόλου. (Μπορεί να να βάλουμε με μία σειρά και να εκτυπωθούν με άλλη).
- Δυναμική δομή: μπορούν να προστεθούν/αφαιρεθούν στοιχεία
- Τα στοιχεία ενός συνόλου μπορεί να είναι διαφορετικών τύπων δεδομένων (int, float, str)

Σύνολα (Set): Μέθοδοι προσθήκης και αφαίρεσης στοιχείων

- Προσθήκη
  - Μέθοδος **.add()** προσθέτει στο σύνολο ένα μεμονωμένο στοιχείο που δίνεται ως όρισμα
  - Μέθοδος **.update()** δέχεται πολλαπλά ορίσματα τύπου λίστας, πλειάδας ή σύνολο και προσθέτει στο σύνολο κάθε στοιχείο που εμφανίζεται στα δεδομένα αυτά.
- Αφαίρεση
  - Μέθοδος **.discard()** αφαιρεί από το σύνολο την τιμή που δίνεται ως όρισμα. Αν η τιμή δεν υπάρχει στο σύνολο, δεν προκύπτει σφάλμα
  - Μέθοδος **.remove()** αφαιρεί από το σύνολο την τιμή που δίνεται ως όρισμα. Αν η τιμή δεν υπάρχει στο σύνολο, προκαλείται σφάλμα

Πράξεις μεταξύ συνόλων



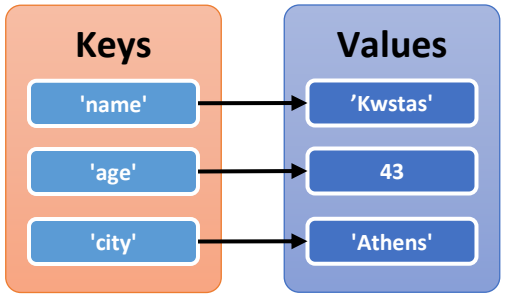
- Τελεστής **in**: εξετάζει αν μία τιμή είναι στοιχείο ενός συνόλου. Επιστρέφει true/false.
- s1**.union(s2)**: επιστρέφει νέο σύνολο που είναι η ένωση δύο συνόλων s1, s2.
- s1**.intersection(s2)**: επιστρέφει νέο σύνολο που είναι η τομή δυο συνόλων s1, s2.
- s1**.difference(s2)***: επιστρέφει νέο σύνολο που είναι η διαφορά των δύο συνόλων, δηλαδή τα στοιχεία του πρώτου συνόλου που δεν ανήκουν στο δεύτερο σύνολο s1 – s2.
- s1**.symmetric_difference(s2)** επιστρέφει νέο σύνολο με την συμμετρική διαφορά των δύο συνόλων.

*** Το *s2.difference(s1)* είναι διαφορετικό από το *s1.difference(s2)*, ενώ για τις άλλες πράξεις το αντίστοιχο είναι ισοδύναμο**

Λεξικά (Dictionaries)

- Το λεξικό (Dictionary) είναι μια δομή δεδομένων στην οποία αποθηκεύουμε ζεύγη δεδομένων με το σχήμα {κλειδί: τιμή}
- Τα κλειδιά είναι μοναδικά εντός του ίδιου λεξικού.
- Οι τιμές μπορεί να είναι απλά δεδομένα ή δομές δεδομένων (περιλαμβανομένων και άλλων λεξικών)
- Μορφή αποθήκευσης δεδομένων σε λεξικά:

```
di = {'name’:’Kwstas', 'age':43, 'city':'Athens'}
```



Μέθοδοι λεξικών

(help(dict) για περισσότερες μεθόδους)

Μέθοδος	Σημασία
.keys()	Επιστρέφει ακολουθίες με τα ονόματα κλειδιών που υπάρχουν στο λεξικό
.values()	Επιστρέφει ακολουθίες μόνο με τις τιμές που υπάρχουν στο λεξικό
.items()	Επιστρέφει ακολουθίες με πλειάδες από ζευγάρια *κλειδί,τιμή*
.clear()	Αφαιρεί όλα τα ζεύγη κλειδιού-τιμής που υπάρχουν στο λεξικό
.get(κ)	Ανακτά την τιμή του κλειδιού k

```
>>> d1 = {'name' : 'Κώστας', 'age' : 43}
>>> d1.keys()
dict_keys(['name', 'age'])

>>> d1.values()
dict_values(['Κώστας', 43])

>>> d1.items()
dict_items([('name', 'Κώστας'), ('age', 43)])
```

Λεξικά (Dictionaries)

- Συγχώνευση λεξικών

```
>>> first_dict = {'name' : 'Sotos', 'age' : 43}
>>> second_dict = {'job' : 'Dev', 'city' : 'Athens'}

>>> first_dict.update(second_dict)
>>> print(first_dict)
{'name': 'Sotos', 'age': 43, 'job': 'Dev', 'city': 'Athens'}
```

Στην περίπτωση που υπάρχουν όμοια κλειδιά, στα δυο λεξικά, τότε εκχωρείτε η τιμή του δεύτερου λεξικού.

- **Προσπέλαση λεξικού**

```
>>> for x in first_dict:
        print(x)

name
age
Job
city
```

```
>>> for x in first_dict:
        print(x, " : ", first_dict[x])

name : Sotos
age : 43
job : Dev
city : Athens
```

```
>>> for x,y in first_dict.items():
        print(x, " : " ,y)

name : Sotos
age : 43
job : Dev
city : Athens
```

Ανάκτηση τιμής

Υπάρχουν δύο τρόποι να ανακτήσουμε την τιμή αν γνωρίζουμε το κλειδί k

```
>>> first_dict = {'name’ : 'Κώστας', 'age' : 43}

>>> first_dict['name']
Κώστας

>>> first_dict.get('name')
Κώστας
```

Η μέθοδος *get(k)* έχει το πλεονέκτημα ότι αν το *k* δεν είναι κλειδί του λεξικού επιστρέφει *None* ή την τιμή που ορίζουμε με το δεύτερο προαιρετικό όρισμα της *get()* ενώ η ανάκτηση τιμής μέσω δείκτη που δεν υπάρχει προκαλεί σφάλμα *KeyError*

Lists, Tuples, Dicts, Sets

Σύγκριση δομών δεδομένων της Python

	Διάταξη /σειρά	Τροποποίηση	Τεμαχισμός	Διπλότυπα	Δημιουργία	Παράδειγμα
Λίστα	✓	✓	✓	✓	[ ] ή list()	[1, 2.3, 5, 'yes', 5]
Πλειάδα	✓	✗	✓	✓	( ) ή tuple()	(1, 2.3, 5, 'yes', 5)
Σύνολο	✗	✓	✗	✗	{ } ή set()	{1, 2.3, 5, 'yes'}
Λεξικό	✗	✓ *	✗	✓	{ } ή dict()	{'year': 2021, 'name': 'Κώστας'}

*Προσοχή τα κλειδιά δεν πρέπει να είναι τροποποιήσιμες δομές δεδομένων (λίστες, λεξικά, σύνολα, κλπ.)

Συναρτήσεις

- Οι συναρτήσεις είναι επαναχρησιμοποιήσιμα τμήματα κώδικα που φέρουν όνομα.
- Εκτελούμε μια συνάρτηση χρησιμοποιώντας το όνομά της οπουδήποτε στο πρόγραμμά μας και για όσες φορές επιθυμούμε.
- Μια συνάρτηση μπορεί να δεχθεί είσοδο και να παραγάγει έξοδο (επιστρεφόμενη τιμή) με χρήση της εντολής **return**.

```
def όνομα_συνάρτησης(παράμετροι): #ορισμός
        συνάρτησης
    εντολές #εκτέλεση εντολών
    return x #επιστροφή αποτελέσματος
```

Συναρτήσεις

Χωρίς επιστρεφόμενη τιμή

```
def hello(): #ορισμός
                συνάρτησης
    print('Hello World!')

hello() #κλήση συνάρτησης
hello() #κλήση συνάρτησης

Hello World!
Hello World!
```

Με επιστρεφόμενη τιμή (***return)***

```
def add(x, y):
    return x + y #

r = add(5, 10) #κλήση συνάρτησης
print(r)
15

r2 = add(20, 30) # 2η κλήση
print(r2)
50
```

Συναρτήσεις: Προαιρετικά ορίσματα

- Τα ορίσματα διακρίνονται σε υποχρεωτικά και προαιρετικά.
- Στα προαιρετικά ορίσματα δίνεται μια προκαθορισμένη τιμή (default value) κατά τον ορισμό της συνάρτησης.

```
def compare(x, y=10): # το x είναι υποχρεωτικό
                      και το y προαιρετικό
    if x > y:
        res = 'x is bigger than y'
    elif x < y:
        res = 'x is smaller than y'
    else:
        res = 'x is equal with y'
    return res

print(compare(x=5))
print(compare(x=5, y=2)

x is smaller than y
x is bigger than y
```

```
print(compare(y=2)

Traceback (most recent call last):
  File » ”/my_dir/my_code.py", line x, in
                        <module>
    print(compare(y=2))
TypeError: compare() missing 1 required
                        positional argument: 'x'
```

Συναρτήσεις: Άγνωστο πλήθος ορισμάτων

- Είναι δυνατόν να χρησιμοποιήσουμε απροσδιόριστο πλήθος ορισμάτων με χρήση του τελεστή *.

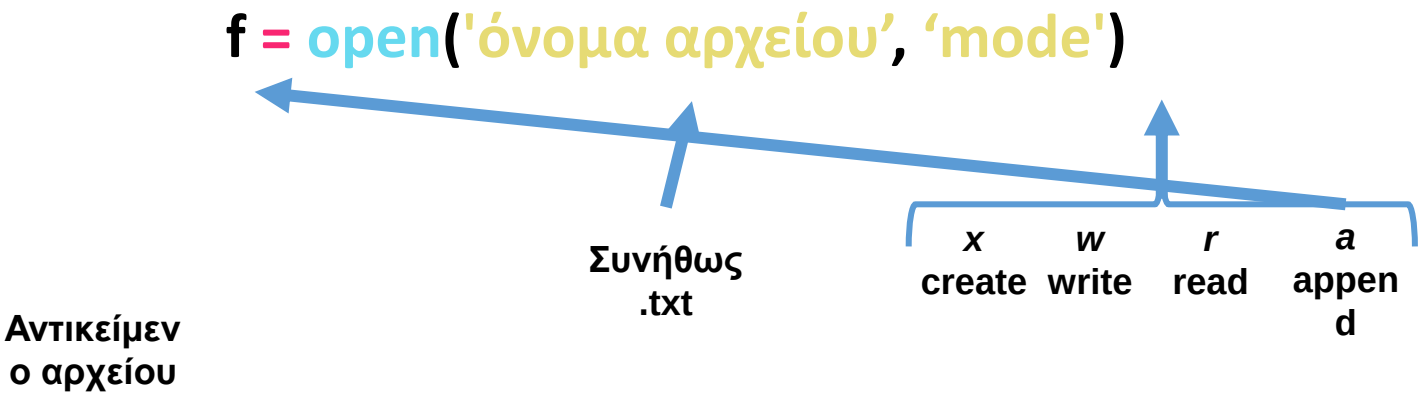
```
def add_arbitrary(*args): # δεν γνωρίζουμε το πλήθος των ορισμάτων
    s = 0
    for number in args:
        s += number
    return s

x = 3
y = 4
print(add_arbitrary(x, y)) #κλήση με δύο ορίσματα

z = 5
w = 6
print(add_arbitrary(x, y, z, w)) #κλήση με 4 ορίσματα

Αποτέλεσμα:
7
18
```

Αρχεία



Αρχεία κειμένου:

```
txt = f.read()  #Επιστρέφεται όλο το περιεχόμενο στο string txt
 txt = f.readlines()     #Επιστρέφεται το περιεχόμενο κατά γραμμή
 for line in txt:
      …....

 f.write("new text")  #Εγγραφή δεδομένων στο αρχείο f, προσοχή στο mode (w, a)
```

Αρχεία

Δυαδικά Αρχεία:

```
Έστω MyDict ένα λεξικό και MyList μία λίστα


import pickle
f = open("myfile","wb")
pickle.dump(MyDict, f)
pickle.dump(MyList, f)
f.close()
```

```
f = open("myfile","rb")
X = pickle.load(f) #Στο χ θα "φορτωθεί" το λεξικό
Y = pickle.load(f)  #Στο Υ θα "φορτωθεί" η λίστα
```

Παιχνίδι #1

Κατασκευάστε ένα **παιχνίδι** στο οποίο ο παίκτης θα προσπαθεί **να μαντέψει έναν ακέραιο αριθμό (1-100)** που γνωρίζει ο υπολογιστής, σύμφωνα με το ακόλουθο παράδειγμα εκτέλεσης:

```
ΠΑΙΧΝΙΔΙ: ΜΑΝΤΕΨΕ ΤΟΝ ΑΡΙΘΜΟ!
Σκέφτηκα έναν ακέραιο από 1 ως 100. Βρες ποιος είναι.
Μάντεψε: 50
Όχι, ο αριθμός μου είναι μικρότερος από 50
Μάντεψε: 30
Όχι, ο αριθμός μου είναι μεγαλύτερος από 30
Μάντεψε: 42
Μπράβο! Τον βρήκες με 3 προσπάθειες.
Τέλος παιχνιδιού.
```

Άσκηση #2 (λύση)

```
print("ΠΑΙΧΝΙΔΙ: ΜΑΝΤΕΨΕ ΤΟΝ ΑΡΙΘΜΟ!")

secret = 42 # Ο ένας παίκτης αλλάζει τον μυστικό αριθμό,
            # ο άλλος προσπαθεί να τον μαντέψει.

print("Σκέφτηκα έναν ακέραιο από 1 ως 100. Βρες ποιος είναι.")

count = 0
while True:
    guess = int(input("Μάντεψε: "))
    count = count+1

    if guess == secret:
        print("Μπράβο! Τον βρήκες με", count, "προσπάθειες.")
        break
    elif guess < secret:
        print("Όχι, ο αριθμός μου είναι μεγαλύτερος από", guess)
    else:   # guess > secret
        print("Όχι, ο αριθμός μου είναι μικρότερος από", guess)

print("Τέλος παιχνιδιού.")
```

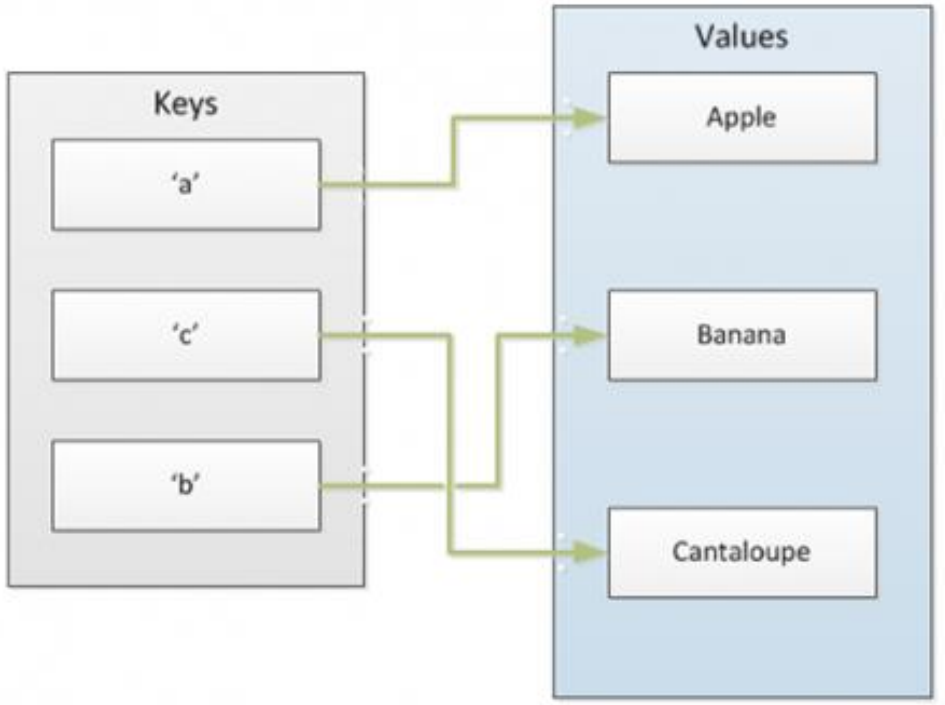
Πλειάδες: Η χρήσιμη συνάρτηση zip

- Κατασκευάζει μια λίστα από πλειάδες κάθε μια από τις οποίες περιέχει τα αντίστοιχα στοιχεία δυο ακολουθιών (λίστες, πλειάδες, αλφαριθμητικά)

```
>>> letters = “ABCDEF”
>>> index = range( 6 )
>>> zip( letters, index )
[  (‘A’ , 0) , (‘B’ , 1) , (‘C’ , 2) , (‘D’ , 3) , (‘E’ , 4) , (‘F’ , 5)  ]
```

Λεξικά (Dictionaries)

```
>>> value = dict( )    # value = { }
>>> value[“a”] = “Apple”
>>> value[“c”] = “Cantaloupe”
>>> value[“b”] = “Banana”
```



Λεξικά (Dictionaries): Το κίνητρο

Λεξικό

Name	Grade
Αστυάναξ	17
Κασσάνδρα	19
Ηλέκτρα	20
Οδυσσέας	20

```
Grade  = {"Αστυάναξ":17, "Κασσάνδρα":19,
          "Ηλέκτρα":20, "Οδυσσέας":20}

name = input("name = ")

if name in Grade :
    print(Grade[name])
else:
    print("Ο ", name, " δεν υπάρχει")
```

Παράλληλες Λίστες

index	Name
0	Αστυάναξ
1	Κασσάνδρα
2	Ηλέκτρα
3	Οδυσσέας

index	Grade
0	17
1	19
2	20
3	20

```
Name  = ["Αστυάναξ", "Κασσάνδρα",
         "Ηλέκτρα", "Οδυσσέας"]
Grade = [17, 19, 20, 20]

name = input("name = ")
found = False
for i in range(len(Name)):
    if Name[i] == name:
        pos = i
        found = True
        break;
if found :
    print(Grade[pos])
else:
    print("Ο ", name, " δεν υπάρχει")
```

Λεξικά (Dictionaries)

- **Λεξικό**: Ενσωματωμένος τύπος mutable δεδομένων της Python που αναπαριστά απεικονίσεις μεταξύ στοιχείων.
- Μια ακολουθία seq απεικονίζει έναν φυσικό αριθμό **i** στην τιμή **seq[i]**
- Ένα λεξικό dict απεικονίζει οποιαδήποτε τιμή **x** (κλειδί) στην τιμή **d[x]**
- HashTables (C++), Associative Arrays (PHP), HashMaps (Java)

```
>>> d = { "A" : 1, "B" : 2, "C" : 3, "D" : 4}
>>> d["B"] = 100
>>> print( d )
 { "A" : 1, "B" : 100, "C" : 3, "D" : 4}
>>> d["Python"] = 3
>>> print( d )
{ "A" : 1, "Python" : 3, "B" : 100, "C" : 3, "D" : 4}
```

Λεξικά (Dictionaries)

```
>>> value = dict()    # δημιουργία κενού λεξικού
>>>
>>> value["A"]=1      # προσθήκη ζεύγους
>>> value["B"]=2
>>>
>>> print( value["A"] + value["B"] )  # προσπέλαση
3
>>> "A" in value      # έλεγχος ύπαρξης
True
>>> "C" in value
False

>>> value["A"]=2019   # ενημέρωση τιμής
>>> print(value)
{'B': 2, 'A': 2019}
>>>
>>> del value["A"]    # διαγραφή ζεύγους κλειδιού-τιμής
>>> print(value)
{'B': 2}
```

Λεξικά (Dictionaries)

Δημιουργία λεξικού

```
>>> value = { 'A' : 1,  'B' : 2}
```

1ος τρόπος

```
>>> value = dict( A=1, B=2 )
```

2ος τρόπος

```
>>> value = dict( )    # value = { }
>>> value["A"] = 1
>>> value["B"] = 2
```

3ος τρόπος

Λεξικά (Dictionaries)

- Κάθε κλειδί αντιστοιχεί σε μια μόνο τιμή, δηλαδή είναι μοναδικό, αλλά οι τιμές όχι
- Δεν υπάρχει διάταξη
- Τα κλειδιά πρέπει να είναι immutable δεδομένα, π.χ. tuples, strings
- Υπάρχουν διάφοροι τρόποι δημιουργίας ενός λεξικού.

```
>>> tupList = [ (“A”, 1), (“B”, 2), (“C”, 3), (“D”, 4) ]
>>> dictionary = dict(  tupList )
>>> print(dictionary)
 { “A” : 1, “B” : 2, “C” : 3, “D” : 4}
>>> d = dict (  name = “Turing”, age = 30 )
>>> print( d )
{ “name” : “Turing”, “age” : 30 }
>>> ‘name’ in d      # Ελέγχει αν υπάρχει ‘name’ στο λεξικό
True
```

Διάσχιση των στοιχείων ενός Λεξικού

items( ) : Επιστρέφει όλα τα ζευγάρια του λεξικού σε μια λίστα.

keys( ) : Επιστρέφει όλα τα κλειδιά του λεξικού σε μια λίστα.

values( ) : Επιστρέφει όλες τις τιμές του λεξικού σε μια λίστα.

```
>>> letters = {"one":3, "two":3, "three":5, "four":4}
>>> letters
{'one': 3, 'two': 3, 'three': 5, 'four': 4}
>>>
>>> letters.keys()
dict_keys(['one', 'two', 'three', 'four'])
>>>
>>> letters.values()
dict_values([3, 3, 5, 4])
>>>
>>> for value in letters.values():
        print(value, end = "  ")


3  3  5  4
```

Διάσχιση των στοιχείων ενός Λεξικού

```
>>> for item in letters.items():
        print(item, end = "  ")


('one', 3)  ('two', 3)  ('three', 5)  ('four', 4)
>>>
>>> for key in letters.keys():
        print(key, end = "  ")


one  two  three  four
>>>
>>> for item in letters.items():
        print(item[0], end = "  ")


one  two  three  four
>>>
>>> for item in letters.items():
        print(item[0][0], end = "")


ottf
```

Εφαρμογή: Πίνακας Συχνοτήτων

Θα αναπτύξουμε μια συνάρτηση αντίστοιχη με το ιστόγραμμα συχνοτήτων της προηγούμενης διάλεξης με τη διαφορά ότι τώρα θα χρησιμοποιήσουμε ένα λεξικό αντί για έναν πίνακα μετρητών. Επίσης τα δεδομένα εισόδου δεν θα είναι αριθμοί αλλά οι χαρακτήρες ενός κειμένου που θα δίνεται ως είσοδος και η συνάρτηση θα επιστρέφει ένα λεξικό με τις συχνότητες που εμφανίζονται τα γράμματα στο κείμενο.

```
>>>  frequency( “ABRACADABRA” )
{ “A” : 5, “R” : 2, “B” : 2, “C” : 1, “C” : 1}
```

Εφαρμογή: Πίνακας Συχνοτήτων

Δημιουργία κενού Λεξικού

```
>>> alphabet = dict( )
```

Ο αλγόριθμος
για κάθε γράμμα του κειμένου :
Αν το έχουμε **βρει ξανά**
  αυξάνουμε τον μετρητή
Αλλιώς
  αρχικοποιούμε : μετρητής = 1

```
for letter in text :
    if letter in alphabet :
        alphabet[ letter ] += 1
    else:
        alphabet[ letter ] = 1
```

Αν δεν χρησιμοποιούσαμε λεξικό θα έπρεπε να υλοποιήσουμε μια αναζήτηση αντί του τελεστή in και μια για να βρούμε τον κατάλληλο μετρητή

Εφαρμογή: Πίνακας Συχνοτήτων

```
def  frequency( text ) :
    alphabet = dict( )
    for letter in text:
        if letter in alphabet :
            alphabet[ letter ] += 1
        else:
            alphabet[ letter ] = 1
    return alphabet
```

Επιστρέφει το λεξικό alphabet που περιέχει την αντιστοίχηση κάθε γράμματος με την συχνότητα εμφάνισής του

Μα είναι ... απλή η μετάβαση

Python 2

```
>>> print "CIE 2020"
CIE 2020
>>> x = input("x = ")
x = 100
>>> x+16
116
>>> 3.0/2
1.5
>>> 3/2
1
>>> print(type(10))
<type 'int'>
```

Python 3

```
>>> print("CIE 2020")
CIE 2020
>>> x = int(input("x = "))
x = 100
>>> x+16
116
>>> 3/2
1.5
>>> 3//2
1
>>> print(type(10))
<class 'int'>
```

Χμμ... Τελικά ... δεν είναι και τόσο απλή

```
>>> from platform import python_version
>>> "python " + python_version()
'python 2.7.13'
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> i=2020
>>> [i*i for i in [11, 111, 1111, 11111]]
[121, 12321, 1234321, 123454321]
>>> print(i)
11111
```

```
>>> from platform import python_version
>>> "python " + python_version()
'python 3.9.0'
>>> range(10)
range(0, 10)
>>> i=2020
>>> [i*i for i in [11, 111, 1111, 11111 ]]
[121, 12321, 1234321, 123454321]
>>> print(i)
2020
```