Βασικές Ασκήσεις από τα Εργαστήρια της PYTHON (εβδομάδα 6)

► Στο προηγούμενο εργαστήριο είδαμε πως δηλώνουμε και πως καλούμε συναρτήσεις στην Python. Οι συναρτήσεις επιτρέπουν την τυποποίηση λειτουργιών και την επαναχρησιμοποίηση κώδικα. Η συνάρτηση max3, που επιστρέφει την μέγιστη τιμή από τρεις μεταβλητές, μπορεί να υλοποιηθεί με διάφορους τρόπους.

1. ► Με σύνθετα αυτόνομα if:

```
def max3(a,b,c):
        if(a>=b and a>=c):
            return a
        if(b>=a and b>=c):
            return b
        if(c>=a and c>=b):
            return c
```

2. ► Με απλά "εμφωλευμένα" (nested) if:

```
def max3(a,b,c):
    if (a>=b):
        if(a>=c):
            return a
        else:
            return c
    else:
        if(b>=c):
            return b
        else:
            return c
```

3. ► Με ένα if-elif-else:

```
def max3(a,b,c):
      if(a>=b and a>=c):
         return a
      elif (b>=a and b>=c):
         return b
      else:
         return c
```

Εννοείται ότι επειδή στην Python οι μεταβλητές δεν έχουν συγκεκριμένο τύπο, η max3 μπορεί να κληθεί και περνώντας Strings: `print(max3('s','f','g'))`, αλλά και περνώντας ακέραιους: `print(max3(100,30,200))`

4. ► Η παρακάτω εκδοχή του προγράμματος βασίζεται στην χρήση της max2, που είναι υλοποιημένη με ένα απλό if, όπου το else δεν χρειάζεται, επειδή εάν δεν ισχύει η συνθήκη, ο έλεγχος θα περάσει αυτόματα στο δεύτερο return.
Παρατηρείστε ότι εδώ η max3 υλοποιείται με χρήση της max2:

```
def max2(a,b):
    if(a>=b):
        return a
    return b

def max3(a,b,c):
    return max2(a,max2(b,c))

print(max3('s','f','g'))
print(max3('Maria','Nikos','Alex'))
print(max3(100,30,201))
```

► Αυτό το πρόγραμμα υλοποιεί αναζήτηση σε λίστα παραμέτρων ψάχνοντας το στοιχείο με την μέγιστη τιμή.

```
def mymax(lista):
    max=lista[0]
    for i in lista:
        if i>max:
            max=i
    return max

print(mymax([1,2]))
print(mymax(['1','2','3','4']))
print(mymax([0]))
```

► Αυτό το πρόγραμμα βρίσκει το μέγιστο στοιχείο από λίστα χρησιμοποιώντας την max της Python. Έτσι μπορεί να αντικαταστήσει τις παραπάνω εκδοχές των max2 και max3, επειδή η λίστα μπορεί να περιέχει οποιοδήποτε αριθμό στοιχείων, άρα και 2 και 3. Για να μπορέσει όμως να λειτουργήσει σωστά πρέπει να καλέσουμε την max με λίστες από παραμέτρους ομοειδείς, αλλιώς δημιουργείται "TypeError: unorderable types":

```
print(max(100,30,201))
print(max(20,30,1,22,12,93,3,1))
print(max([1])) # προσοχή, max(1) δεν επιτρέπεται!
```

► Το παρακάτω πρόγραμμα είναι μία απλή δική μας υλοποίηση της συνάρτησης len, η οποία μετράει το μέγεθος του string ή της λίστας την οποία περνάμε σαν argument. Η len είναι ήδη υλοποιημένη στην Python και προφανώς δεν χρειάζεται άλλη υλοποίηση, που μάλιστα θα είναι και φτωχότερη.

```
def myLen(lst):
    length=0
    for i in lst:
        length+=1
    return length

print(myLen('a sentence'))
print(myLen(['a','b','c']))
print(myLen([]))
print(myLen(''))
```

► Δημιουργία πρόχειρου ιστογράμματος από τις τιμές πίνακα (Από το βιβλίο Deitel & Deitel), όπως υλοποιείται στη γλώσσα C:

```
#include <stdio.h>
#define SIZE 10

int main(void) {
    int n[SIZE] = {19, 3, 15, 7, 11, 9, 13, 5, 17, 1};
    int i;
    int j;
    printf("%s%13s%17s\n", "Element", "Value", "Histogram");

    for (i = 0; i < SIZE; i++) {
        printf("%7d%13d", i, n[ i ]);
        for (j = 1; j <= n[ i ]; j++) { /* print one bar */
            printf("%c", '*');
        }
        printf("\n");
    }
    return 0;
}
```

Εδώ βλέπουμε ακριβώς το ίδιο πρόγραμμα γραμμένο όμως στη γλώσσα Python. Δημιουργία πρόχειρου ιστογράμματος από τις τιμές λίστας. Σε αντίθεση με την C, εδώ δεν χρειάζεται διπλό for επειδή η γλώσσα επιτρέπει εκφράσεις όπως `'*'*5` π.χ. που θα τύπωνε 5 φορές '*'. Εδώ γίνεται χρήση του `'*'*lista[i]` :

```
def histo(lista):
    print('Element', 'Value', 'Histogram')
    for i in range(0,len(lista)):
        print('%3d%9d  %s'%(i, lista[i], '*'*lista[i]))


n=[19, 3, 15, 7, 11, 9, 13, 5, 17, 1]
histo(n)
```

Και οι δύο υλοποιήσεις μας δίνουν το παρακάτω output:

```
Element Value Histogram
  0       19  ******************
  1        3  ***
  2       15  ***************
  3        7  *******
  4       11  ***********
  5        9  *********
  6       13  *************
  7        5  *****
  8       17  *****************
  9        1  *
```

► Στο προηγούμενο μάθημα είδαμε ότι εάν κάνουμε import μία βιβλιοθήκη, τότε μπορούμε να καλέσουμε τις συναρτήσεις που περιέχει. Το αρχείο **basics.py** περιέχει δύο συναρτήσεις:

```
# basics.py περιέχει 2 συναρτήσεις:

def maxi(x,y):
    if (x < y):
        return y
    return x

def mini(x,y):
    if (x > y):
        return y
    return x
```

Τις συναρτήσεις μπορούμε να τις καλέσουμε από ένα άλλο αρχείο .py έτσι:

```
import basics

print(basics.maxi(3,21))
print(basics.mini("Mitsos","Kitsos"))
```

► Δύο τρόποι για να επεξεργαστούμε μία λίστα[1]:

Τρόπος 1: - Χρησιμοποιώντας **while** και **len**:

```
movies = ["The Holy Grail", "The Life of Brian", "The Meaning of Life"]

count = 0
while count < len(movies):
    print(movies[count])
    count = count+1
```

Τρόπος 2: - Χρησιμοποιώντας **for**:

```
movies = ["The Holy Grail", "The Life of Brian", "The Meaning of Life"]

for each_item in movies:
    print(each_item)
```

► Για να προσθέσουμε πληροφορίες, όπως το έτος δίπλα στον τίτλο της κάθε ταινίας μπορούμε να ξαναδηλώσουμε την λίστα έτσι ώστε να περιέχει σύνθετες πληροφορίες, όπως τίτλο και έτος ταινίας:

```
>>> movies=["The Holy Grail", 1975, "The Life of Brian", 1979, "The Meaning of
Life", 1983]
>>> movies
['The Holy Grail', 1975, 'The Life of Brian', 1979, 'The Meaning of Life', 1983]
>>>
```

Το ίδιο θα μπορούσε να γίνει χρησιμοποιώντας **insert** και **append** για να προσθέσουμε τα έτη στην υπάρχουσα λίστα:

```
>>> movies = ["The Holy Grail", "The Life of Brian", "The Meaning of Life"]
>>> movies.insert(1,1975)
>>> movies.insert(3,1979)
>>> movies.append(1983)
>>> movies
['The Holy Grail', 1975, 'The Life of Brian', 1979, 'The Meaning of Life', 1983]
>>>
```

1 Παράδειγμα από το βιβλίο Head First Python του Paul Barry, O'REILLY - 2011

► Η λίστα movies μπορεί να περιέχει ακόμη πιο σύνθετες πληροφορίες, λίστες από ηθοποιούς:

```
>>> movies = [
           "The Holy Grail", 1975, "Terry Jones & Terry Gilliam", 91,
                 ["Graham Chapman",
                       ["Michael Palin",
                        "John Cleese", "Terry Gilliam",
                        "Eric Idle", "Terry Jones"]]]
>>> print(movies[4][1][3])
Eric Idle
>>> print(movies)
['The Holy Grail', 1975, 'Terry Jones & Terry Gilliam', 91, ['Graham Chapman',
['Michael Palin', 'John Cleese', 'Terry Gilliam', 'Eric Idle', 'Terry Jones']]]
>>> for each_item in movies:
     print(each_item)


The Holy Grail
1975
Terry Jones & Terry Gilliam
91
['Graham Chapman', ['Michael Palin', 'John Cleese', 'Terry Gilliam', 'Eric Idle',
'Terry Jones']]
>>>
```

Το for-loop δουλεύει εντάξει, αλλά επιλέγει μόνο τα περιεχόμενα της εξωτερικής λίστας. Τις εσωτερικές λίστες τις επιλέγει ολόκληρες, σαν στοιχεία της εξωτερικής λίστας.

► Με τον έλεγχο isinstance(each_item, list) για κάθε λίστα τυπώνουμε τα περιεχόμενα της:

```
>>> for each_item in movies:
     if isinstance(each_item, list):
           for nested_item in each_item:
                 print(nested_item)
     else:
           print(each_item)


The Holy Grail
1975
Terry Jones & Terry Gilliam
91
Graham Chapman
['Michael Palin', 'John Cleese', 'Terry Gilliam', 'Eric Idle', 'Terry Jones']
>>>
```

► Για να φτάσουμε στην εσωτερική λίστα πρέπει να γράψουμε τον κώδικά μας με διπλό έλεγχο:

```
>>> for each_item in movies:
      if isinstance(each_item, list):
            for nested_item in each_item:
                  if isinstance(nested_item, list):
                        for deeper_item in nested_item:
                              print(deeper_item)
                  else:
                        print(nested_item)
      else:
            print(each_item)


The Holy Grail
1975
Terry Jones & Terry Gilliam
91
Graham Chapman
Michael Palin
John Cleese
Terry Gilliam
Eric Idle
Terry Jones
>>>
```

► Αν έχουμε λίστες με μεγάλο βάθος η παραπάνω προσέγγιση απαιτεί πολύπλοκο κώδικα. Εδώ βοηθάει η χρήση αναδρομής, όπου μία συνάρτηση καλεί τον εαυτό της:

```
def print_lol(the_list):
    for each_item in the_list:
        if isinstance(each_item, list):
            print_lol(each_item)
        else:
            print(each_item)



movies = [
"The Holy Grail", 1975, "Terry Jones & Terry Gilliam", 91,
["Graham Chapman",
["Michael Palin", "John Cleese", "Terry Gilliam", "Eric Idle", "Terry Jones"]]]

print_lol(movies)
```

► Συνάρτηση Fibonacci:

```
def fib2(n): # return Fibonacci series up to n
      """Return a list containing the Fibonacci series up to n."""
      result = []
      a, b = 0, 1
      while a < n:
            result.append(a)    # see below
            a, b = b, a+b
      return result

f100 = fib2(100)    # call it
print(f100)         # write the result
```

► Αναδρομική συνάρτηση υπολογισμού παραγοντικού:

```
def fact(x):
'''Returns the factorial of its argument, assumed to be a posint'''
      if x == 0:
            return 1
      return x * fact(x - 1)

print
print('N fact(N)')
print("---------")

for n in range(10):
    print(n, fact(n))
```