Τμήμα Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Μάθημα: **Τεχνικές Προγραμματισμού Υπολογιστών. (Εργαστηριακό μάθημα)**
Καθηγητής : Πεφάνης Ευάγγελος

2) Εργαστηριακές σημειώσεις στην γλώσσα προγραμματισμού Python.

Ανακύκλωση (Βρόγχοι)

Η εντολή while

Η εντολή αναλύκλωσης `while` περιγράφει μία επαναληπτική διαδικασία και το συντακτικό της είναι:

```
while expression:
    statements
```

Οι εντολές *statements* εκτελούνται εφόσον η έκφραση *expression* είναι αληθής. Ο τρόπος ομαδοποίησης των εντολών ανακύκλωσης *statements* είναι πάλι η εσοχή (tab).

Ένα πρώτο **παράδειγμα**:

```
max = int(input("Give a positive integer: "))
i = 0
while i < max:
    i = i + 1
    print(i)
print(i)
```

Εύρεση ρίζας θετικού αριθμού με τον αλγόριθμο του Ήρωνα.

```
x = input("Give a number")
x = float(x)
g = 1.0                # first approximation
while abs(g*g - x) > 1.0e-5:
    g = (g+x/g)/2.0
print("The square root of",x,"is approximately",g)
```

Συμβολοσειρές. Μπορούμε να εξαγάγουμε έναν-έναν τους χαρακτήρες μιας συμβολοσειράς.

```
s = input("Give a string: ")

i = 0
```

```
while i < len(s):
    print(i,"character is",s[i])
    i = i + 1
```

Παρατήρηση: η εντολή `iter += 1` είναι ισοδύναμη με την `iter = iter + 1`. Γενικότερα, η `x += y` είναι ισοδύναμη με την `x = x + y`. Επίσης υπάρχουν και οι τελεστές -=, *=, /=, //=. **= με ανάλογες σημασίες.

Εντολή `break`

Μπορούμε να τερματίσουμε μία ανακύκλωση πρόωρα με την εντολή `break`.

```
s = input("Give a string: ")

i = 0
while i < len(s):
    if s[i] == 'o':
        print("The character o is in position",i)
    i = i + 1
if i == len(s):
    print("There is no o in the word", s)
```

Η εντολή `break` είναι απαραίτητη στις ατέρμονες ανακυκλώσεις.

```
while True:
    x = int(input("Input a number between 1 and 100: "))
    if  x >= 1 and x <= 100:
        break
print("Well done!")
```

Εντολή `continue`

Οι εντολές της ανακύκλωσης δεν εκτελούνται και η ανακύκλωση επαναλαμβάνεται όταν υπάρχει η εντολή `continue`.

```
iter = 0
summ = 0
while iter < 10:
    x = int(input('Enter a positive integer: '))
    if x <= 0:
        continue
    summ = summ + x
    iter = iter + 1
print('The sum is', sum)
```

Ανακύκλωση με την εντολή for

Η εντολή ανακύκλωσης `while`, ορισμένες φορές, εκτελεί κάποιες εντολές όταν ένας μετρητής, η λεγόμενη μεταβλητή της ανακύλωσης, παίρνει τιμές από μια ακολουθία ακεραίων. Για παράδειγμα, μπορούμε να τυπώσουμε τα τετράγωνα των ακεραίων από το 1 έως το 10.

```
i = 1
while i < 10:
    print('The square of', i, 'is', i**2)
    i += 1
```

Για να σαρώσουμε επαναληπτικά μία ακολουθία ακεραίων υπάρχει ένας δεύτερος τρόπος στην python με τη χρήση της εντολής `for`.

```
for i in range(0,10):
    print('The square of', i, 'is', i**2)
```

range. Η συνάρτηση `range(start,end)` παράγει μία ακολουθία ακεραίων από το `start` έως το `end-1`.

Η γενική μορφή της συνάρτησης είναι `range(start,end,step)` και κατασκευάζει την αριθμητική πρόοδο `start, start+step, start+2*step,...,start+n*step`.

- Αν `step>0` τότε στην παραπάνω ακολουθία ο `start+n*step` είναι ο μεγαλύτερος ακέραιος μικρότερος από τον `end`.
- Αν `step<0` τότε ο `start+n*step` είναι ο μικρότερος ακέραιος μεγαλύτερος από τον `end`.
- Αν το `start` παραληφθεί τότε `start=0`. Αν το `step` παραληφθεί τότε `step=1`.

Η μεταβλητή `i` που εμφανίζεται μετά τη λέξη-κλειδί `for` λαμβάνει διαδοχικά τις τιμές της ακολουθίας `range(0,max)` και η εντολή `print(i)` εκτελείται. Η διαδικασία επαναλαμβάνεται μέχρι να εξαντληθούν οι τιμές της ακολουθίας.

in. Δείτε τον τελεστή `in` με τον οποίο ελέγχουμε κατά πόσο η μεταβλητή `i` είναι μέλος της ακολουθίας `range(0,max)`.

Το συντακτικό της εντολής `for` είναι

```
for variable in sequence:
    statements
```

Ο τελεστής `in` ελέγχει κατά πόσο η μεταβλητή *variable* είναι μέλος της ακολουθίας *sequence*. Οι εντολές *statements* εκτελούνται και η διαδικασία επαναλαμβάνεται μέχρι να εξαντληθούν οι τιμές της ακολουθίας, ή μια εντολή `break` τερματίσει πρόωρα την ανακύκλωση.

Παράδειγμα. Για την εύρεση κυβικής ρίζας (τέλειου κύβου, έστω n) με εξαντλητική απαρίθμηση σαρώσαμε μία σειρά ακεραίων από το 1 έως το πολύ n.

```
pc = int(input("Give an integer: "))
for x in range(0,pc):
    if x**3 != abs(pc):
        print("The cubic root of",pc,"is",x)
        break
```

Μία βελτίωση του παραπάνω προγράμματος είναι η ακόλουθη (σκεφτείτε γιατί αυτό αποτελεί βελτίωση).

```
pc = int(input("Give an integer: "))
for x in range(0,abs(pc)+1):
    if x**3 >= abs(pc):
       break
if x**3 != abs(pc):
   print(pc,"is not a perfect cube")
else:
   if pc < 0:
       x = -x
   print("The cubic root of",pc,"is",x)
```

strings ως ακολουθίες χαρακτήρων

Οι μεταβλητές τύπου string είναι ακολουθίες χαρακτήρων και μπορούμε να γράψουμε εντολές ανακύκλωσης που τυπώνουν έναν-έναν τους χαρακτήρες μιας ακολουθίας χαρακτήρων:

```
s = 'Nick Papadakis'
for c in s:
    print(c)
```

Συναρτήσεις

Εντολή `def`

Εισαγωγή. Πολλά προγράμματα τα οποία φτιάξαμε μέχρι τώρα θα μπορούσαμε να τα δούμε και ως ολοκληρωμένες μεθόδους οι οποίες παίρνουν δεδομένα και πετυχαίνουν ένα αποτέλεσμα. Για παράδειγμα, μπορούμε να βρούμε τον μεγαλύτερο μεταξύ δύο αριθμών ως εξής:

```
x = 5.0
y = 6.0

if x >= y:
   maxx = x
else
   maxx = y

print(maxx)
```

Το τμήμα του παραπάνω προγράμματος το οποίο κάνει τον υπολογισμό του μεγίστου μπορούμε να το διαχωρίσουμε ως εξής:

```
def maximum(x,y):
   if x >= y:
      maxx = x
   else
      maxx = y
   return maxx
```

Η λέξη `def` είναι δεσμευμένη, είναι μία εντολή της python και ορίζει μία συνάρτηση.

- Η πρώτη γραμμή περιέχει το όνομα της συνάρτησης (η παραπάνω λέγεται `maximum`), επίσης τα ορίσματα σε παρένθεση και η γραμμή τελειώνει με την άνω-κάτω τελεία. Το όνομα μιας συνάρτησης ακολουθεί τους συνηθισμένους κανόνες της Python για τα ονόματα μεταβλητών.
- Όλες τις εντολές που περιέχονται στη συνάρτηση βρίσκονται σε εσοχή (tab).
- Η συνάρτηση τελειώνει με την εντολή `return` η οποία καθορίζει ποιά είναι η τιμή της συνάρτησης.

Το αρχικό πρόγραμμα μπορεί να εκτελεστεί τώρα με τις εντολές.

```
result = maximum(5.0,6.0)
print(result)
```

Δείτε ότι το πρόγραμμά μας αποτελείται τώρα από δύο τμήματα: τη συνάρτηση και το κύριο πρόγραμμα (το οποίο ακολουθεί τη συνάρτηση και περιέχει μία γραμμή για τη χρήση της συνάρτησης.)

Ας δούμε τι ακριβώς συμβαίνει όταν στο πρόγραμμά μας περιέχεται μία συνάρτηση (π.χ. `maximum`) και γράφουμε `maximum(a,b)`.

- Λέμε ότι η συνάρτηση `maximum` *καλείται* με τις **πραγματικές παραμέτρους** (ή **ορίσματα**) `a,b`. Αυτές μπορεί να είναι αριθμοί (π.χ., `maxx=maximum(5.0,6.0)`) αλλά μπορεί να είναι και μεταβλητές οι οποίες έχουν τιμές (π.χ., `a=3.0; b=4.0; maxx=maximum(a,b)`).
- Με την κλήση της συνάρτησης συμβαίνουν δύο πράγματα: Οι **τυπικές παράμετροι** της συνάρτησης παίρνουν τιμές (`x=5.0; y=6.0` είτε `x=a; y=b`) και το πρόγραμμα συνεχίζει με τις εντολές εντός της συνάρτησης.
- Όταν βρεθεί εντολή `return` ο έλεγχος επιστρέφει στο κύριο πρόγραμμα.
- Η τιμή της μεταβλητής που βρίσκεται δίπλα στην `return` (δηλαδή, `maxx`) είναι η *τιμή της συνάρτησης*. Αυτή δίδεται στο κύριο πρόγραμμα (στη μεταβλητή `result`) αριστερά της ισότητας στην κλήση της συνάρτησης.
- Ο έλεγχος έχει τώρα επιστρέψει στο κύριο πρόγραμμα και εκτελούνται οι εντολές μετά της κλήση της συνάρτησης.

Παράδειγμα. Γράψτε μία συνάρτηση η οποία να υπολογίζει την τιμή της $f(x)=x2−2x+1$f(x)=x2−2x+1. Καλέστε την από το κύριο πρόγραμμα για μία τιμή του xx και τυπώστε τα $x,f(x)$x,f(x).

```
def f(x):
    fvalue = x**2 - 2*x + 1
    return fvalue

x = 3.0
result = f(x)
```

```
print(x,result)
```

Παράδειγμα. Γράψτε μία συνάρτηση η οποία να ελέγχει αν το όνομά μας τελειώνει σε "ακης" (akis). Σε αυτή την περίπτωση η τιμή της θα είναι `True`, αλλιώς η τιμή της συνάρτησης θα είναι `False`.

```
def name_end(s):
    if s[-4:] == 'akis':
        return True
    else:
        return False

name = input("Give a name: ")
print(name_end(name))
```

Εντολή `return`. Σημειώστε ότι η εμφάνιση της εντολής return σε οποιοδήποτε σημείο στο σώμα μιας συνάρτησης τερματίζει την εκτέλεση των εντολών της συνάρτησης και επιστρέφει τη ροή του προγράμματος στο σημείο αμέσως μετά την κλήση της. Η χρήση της εντολής return είναι προαιρετική. Αν αυτή δεν εμφανίζεται ή η εμφάνισή της δεν ακολουθείται από κάποια έκφραση, τότε η συνάρτηση επιστρέφει την τιμή None.

Παράδειγμα. Ας ορίσουμε μία συνάρτηση η οποία τυπώνει το μήνυμα `Hello!` όποτε κληθεί:

```
def sayHello():
    print('Hello!')
```

Ας την καλέσουμε:

```
>>> sayHello()
Hello!
```

Αν όμως είχαμε γράψει `print(sayHello(), 'Maria')` θα βλέπαμε το μήνυμα `None Maria` γιατί `None` είναι η τιμή η οποία επιστρέφει η συνάρτηση `sayHello`. Δείτε ακόμα ότι η συνάρτηση αυτή δεν έχει τυπικά ορίσματα και γι' αυτό η λίστα των τυπικών ορισμάτων της είναι κενή, αλλά η κλήση της συνάρτηση διατηρεί την κενή λίστα των ορισμάτων.

Παρατήρηση. Το κεντρικό πλεονέκτημα των συναρτήσεων είναι ότι μπορούμε να τις καλούμε επανειλημμένα από το κύριο πρόγραμμα. Ο κώδικας που περιέχεται στη συνάρτηση εκτελείται σε κάθε κλήση της από το κύριο πρόγραμμα.

Ενσωματωμένες συναρτήσεις (built-in functions)

Πρέπει να παρατηρήσουμε ότι έχουμε ήδη δει συναρτήσεις σε προηγούμενα μαθήματα, όπως αυτές τις οποίες παρέχει το πακέτο `math`.

- `math.sin(x)`, `math.cos(x)`, `math.abs(x)`, etc.

Παρατηρήστε ότι όλες οι παραπάνω παίρνουν ένα όρισμα και επιστρέφουν μία τιμή, όπως ακριβώς οι συναρτήσεις που ορίζονται με την εντολή `def`.

Επίσης, έχουμε δει ενσωματωμένες συναρτήσεις της python.

- `type(a), print(), len(s)` etc.

Παρατήρηση. Η χρήση συναρτήσεων είναι ιδαίτερα διαδεδομένη σε μία γλώσσα προγραμματισμού (όπως η python). Αυτό ισχύει και για τις ενσωματωμένες συναρτήσεις και για αυτές τις οποίες γράφει ο προγραμματιστής. Και τα δύο είδη συναρτήσεων χρησιμοποιούνται με τον ίδιο τρόπο, όπως είδαμε.

Εμβέλεια μεταβλητών

Παράδειγμα. Ας γράψουμε μια συνάρτηση η οποία, δεδομένου του φυσικού αριθμού n, υπολογίζει το άθροισμα $1^2+2^2+\cdots+n^2$.

```
def sumsq(n):
    if n <= 0:
        return 0
    s = 0
    for i in range(1, n+1):
        s += i*i
    return s

print('Sum of the squares of the first four integers =', sumsq(4))
```

Το σώμα μιας συνάρτησης μπορεί να περιέχει οποιαδήποτε εντολή της Python, συμπεριλαμβανομένης, φυσικά, και της εντολής ανάθεσης. Το σώμα της παραπάνω συνάρτησης `sumsq` κάνει χρήση δύο μεταβλητών, των *i* και *s*. Η πρώτη είναι η μεταβλητή της ανακύκλωσης `for` και η δεύτερη περιέχει το τρέχων άθροισμα. Είναι σημαντικό να καταλάβουμε ότι αυτές οι μεταβλητές είναι "τοπικές" με την έννοια ότι είναι άγνωστες στο υπόλοιπο μέρος του προγράμματός μας. Αν, για παράδειγμα, προσπαθούσαμε να τυπώσουμε και την τιμή του μετρητή της ανακύκλωσης, εκτός του σώματος της συνάρτησης, θα παίρναμε το μήνυμα σφάλματος `NameError: name 'i' is not defined`. Το ίδιο ακριβώς συμβαίνει και με τα τυπικά ορίσματα μιας συνάρτησης: συμπεριφέρονται ως τοπικές μεταβλητές, είναι άγνωστες δηλαδή εκτός του σώματος της συνάρτησης. Το γεγονός αυτό μας επιτρέπει να γράψουμε τον κώδικα για τη συνάρτηση `sumsq` ως εξής:

```
def sumsq(n):
    if n <= 0: return 0
    s = 0
    while n > 0:
        s += n*n
        n -= 1
    return s

print('Sum of the squares of the first four integers =', sumsq(4))
```

Επιπλέον, εκτός του σώματος της συνάρτησης `sumsq` μπορούμε να χρησιμοποιήσουμε το όνομα *n* ως όνομα μεταβλητής και η οποία δεν έχει καμμία απολύτως σχέση με το τυπικό όρισμα της `sumsq` με το ίδιο όνομα. Ίσα-ίσα, αυτό τονίζει τη σχέση με το τοπικό όρισμα της `sumsq`.

```
def sumsq(n):
    if n <= 0: return 0
    s = 0
    while n > 0:
        s += n*n
        n -= 1
    return s

n = 7
print('Sum of the squares of the first', n, 'integers =', sumsq(n))
```

Παρατήρηση. Η Python, κατά τον ορισμό μιας συνάρτησης δημιουργεί τον λεγόμενο *χώρο ονομάτων* (namespace) ο οποίος περιέχει τόσο τα τυπικά ορίσματα της συνάρτησης αλλά και μεταβλητές τις οποίες χρησιμοποιεί, για τις ανάγκες της η συνάρτηση. Τα ονόματα αυτών των μεταβλητών είναι άγνωστα έξω από το χώρο ονομάτων. Αναφερόμαστε στις μεταβλητές που εμφανίζονται στο χώρο ονομάτων μιας συνάρτησης ως *τοπικές μεταβλητές*. Τα τυπικά ορίσματα συμπεριφέρονται κι αυτά ως τοπικές μεταβλητές μέσα στο σώμα της συνάρτησης.

Παράδειγμα. Ας δούμε τη συνάρτηση:

```
def f(x):
    y = 1
    x = x + y
    print('x =', x, 'y =', y)
    return x

x = 3
y = 2
z = f(x)
print('x =', x)
print('y =', y)
print('z =', z)
```

Παρατηρούμε ότι το όνομα *x* εμφανίζεται τόσο ως τυπικό όρισμα της συνάρτησης *f* αλλά και στον κώδικα ο οποίος περιέχει την κλήση της συνάρτησης (κύριο πρόγραμμα). Επίσης, το σώμα της συνάρτησης περιέχει την μεταβλητή *y* η οποία εμφανίζεται επίσης και εκτός του σώματος της συνάρτησης.

Αν εκτελέσουμε το πρόγραμμα θα δούμε τα παρακάτω αποτελέσματα

```
x = 4 y = 1
x = 3
y = 2
z = 4
```

τα οποία δείχνουν ότι οι μεταβλητές `x, y` εντός του σώματος της συνάρτησης είναι διαφορετικές από τις μεταβλητές `x, y` στο κύριο πρόγραμμα (βρίσκονται σε διαφορετικούς χώρους ονομάτων (namespace).):

Καθολικές μεταβλητές

Σε πολλές περιπτώσεις το προβλημά μας περιέχει παραμέτρους (ή μεταβλητές) οι οποίες θα θέλαμε να χρησιμοποιηθούν τόσο στο κύριο πρόγραμμα όσο και σε συναρτήσεις. Μπορούμε να καθορίσουμε ότι

αναφερόμαστε σε μία συγκεκριμένη μεταβλητή σε διαφορετικά τμήματα του προγράμματος (π.χ., στο κύριο πρόγραμμα και σε μία συνάρτηση) δηλώνοντάς την με την εντολή `global`.

Παράδειγμα. (Φυσικές σταθερές) Ας υπολογίσουμε, με τη βοήθεια συνάρτησης, τη θέση σώματος σε ελεύθερη πτώση.

```
# function
def height(t):
    global g
    y = 0.5*g*t**2
    return y

# Main program
global g
g = 10.0                      # give value to parameter
time = float(input("Give time: ")
print(time,height(time))
```

Ας υπολογίσουμε επίσης τη θέση σώματος σε ελεύθερη πτώση με αρχική ταχύτητα v0v0. Θα οργανώσουμε το πρόγραμμα ώστε όλες οι φυσικές σταθερές να παίρνουν τιμές μέσω μίας νέας συνάρτησης.

```
# Functions
def physicalParams():
    global g,v0
    g = 10.0
    v0 = 10.0

def height(t):
    global g,v0
    y = v0*t-0.5*g*t**2
    return y

# Main program
global g,v0
physicalParams()
time = float(input("Give time: "))
print(time,height(time))
```

Ερώτηση. Θα μπορούσαμε να παραλείψουμε κάποια από τις εντολές global στο παραπάνω πρόγραμμα; (ποιά;)

Προδιαγραφές

Εισαγωγή. Για κάθε συνάρτηση μπορούμε να πάρουμε ένα κείμενο το οποίο αναφέρει τις *προδιαγραφές* της, δηλαδή, βασικές πληροφορίες γι' αυτήν: τον τύπο και τη σημασία των τυπικών ορισμάτων καθώς και την πληροφορία που επιστρέφει η συνάρτηση. Αυτά μπορούμε να τα ανακαλέσουμε με την εντολή `help`.

Παράδειγμα. Για τις ενσωματωμένες συναρτήσεις έχουμε:

```
>>> help(len)
Help on built-in function len in module builtins:

>>> help('math.exp')
```

```
Help on built-in function exp in math:

math.exp = exp(...)
    exp(x)

    Return e raised to the power of x.
```

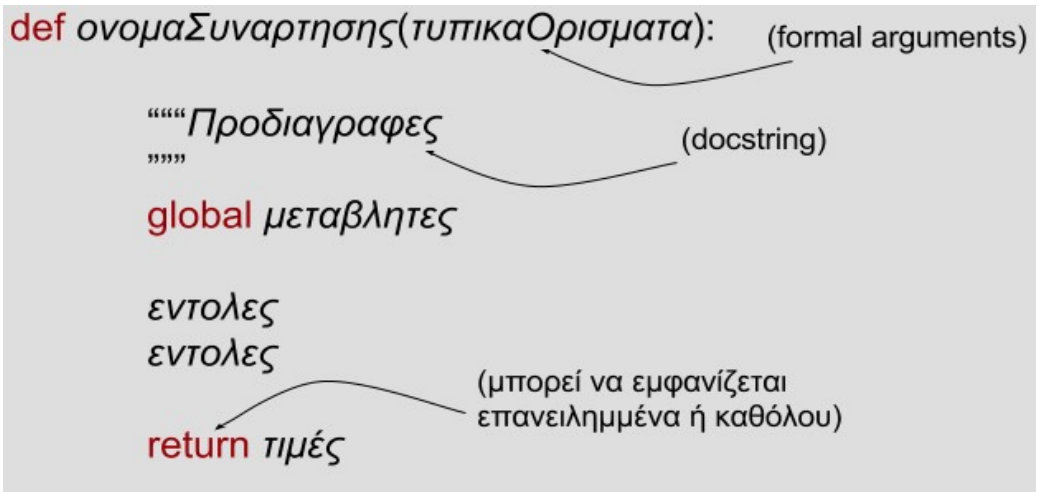
Όταν γράφουμε μία συνάρτηση μπορούμε να καθορίσουμε τις προδιαγραφές της σε κείμενο το οποίο γράφουμε στις επόμενες γραμμές μετά την εντολή def και το οποίο περιέχεται ανάμεσα σε σύμβολα """. (Το κείμενο αυτό λέγεται docstring.)

Παράδειγμα. Ας συμπληρώσουμε τη συνάρτηση την οποία δώσαμε νωρίτερα, με τις προδιαγραφές της.

```
def organizeName(firstName, lastName, reverse = False):
    """firstName, lastName are strings, reverse is a boolean
    if reverse==False returns firstName lastName
    if reverse==True returns lastName, firstName"""

    if reverse == True:
        s = lastName + ', ' + firstName
    else:
        s = firstName + ' ' + lastName

    return s
```

Μπορούμε τώρα να ανακαλέσουμε τις προδιαγραφές της συνάρτησης:

```
>>> help(organizeName)

organizeName(firstName, lastName, reverse=False)
    firstName, lastName are strings, reverse is a boolean
    if reverse==False returns firstName lastName
    if reverse==True returns lastName, firstName
```



Σχήμα. Η γενική μορφή μιας συνάρτησης.

Συναρτήσεις - ειδικότερα θέματα

Ορίσματα με προεπιλεγμένες τιμές (default parameter values)

Παράδειγμα. Ας γράψουμε τώρα μια συνάρτηση η οποία επιστρέφει σε μια ακολουθία χαρακτήρων την ημερομηνία στη μορφή `day-month-year` ή, εναλλακτικά, στη μορφή `month-day-year`, όπως συνηθίζεται σε κάποιες χώρες του κόσμου. Είναι λογικό να θεωρήσουμε ότι η λίστα των τυπικών ορισμάτων θα περιλαμβάνει τα `day`, `month`, `year`, καθώς και ένα τέταρτο όρισμα, ας το πούμε `america` το οποίο ορίζει τον τρόπο σχηματισμού της ακολουθίας χαρακτήρων. Αν η συνάρτηση κληθεί με τιμή του ορίσματος `america` ίσο με `True` τότε θα σχηματίσει την ακολουθία χαρακτήρων `month-day-year`, διαφορετικά την `day-month-year`. (Στην Αμερική χρησιμοποιείται ο τρόπος γραφής ημερομηνίας `month-day-year`.)

```
def dateString(day, month, year, america):
    if america:
        return str(month) + '-' + str(day) + '-' + str(year)
    else:
        return str(day) + '-' + str(month) + '-' + str(year)
```

Η συνηθισμένη μορφή για την ημερομηνία (για εμάς) είναι η `day-month-year`. Η python μας δίνει τη δυνατότητα να απλοποιήσουμε την κλήση της συνάρτησης για αυτή την περίπτωση χρησιμοποιώντας προεπιλεγμένες τιμές για ένα ή περισσότερα ορίσματα (default parameter values).

```
def dateString(day, month, year, america=False):
    if america:
        return str(month) + '-' + str(day) + '-' + str(year)
    else:
        return str(day) + '-' + str(month) + '-' + str(year)
```

Η απουσία του ορίσματος `america` είναι ισοδύναμη με την κλήση της συνάρτησης με το όρισμα `america=False`. Έτσι, όλες οι παρακάτω κλήσεις της `dateString` είναι επιτρεπτές:

```
dateString(12, 10, 2016)
dateString(12, 10, 2016, True)
dateString(12, 10, 2016, america=True)
```

Η πρώτη θα επιστρέψει την ακολουθία χαρακτήρων `'12-10-2016'` ενώ οι επόμενες δύο την ακολουθία χαρακτήρων `'10-12-2016'`.

Παρατήρηση....

Keyword arguments (ορίσματα-δεσμευμένες λέξεις)

Εισαγωγή. Σε όλα τα παραδείγματα συναρτήσεων τα οποία έχουμε δει, η αντιστοίχιση των τυπικών ορισμάτων κατά την κλήση της συνάρτησης γίνεται σύμφωνα με τη θέση τους (η μέθοδος αυτή αντιστοίχισης λέγεται *θεσιακή*). Η Python επιτρέπει και έναν δεύτερο τρόπο αντιστοίχισης τυπικών ορισμάτων και ορισμάτων χρησιμοποιώντας το όνομα του τυπικού ορίσματος. Θα μπορούσαμε να

γράψουμε `dateString(day=12, month=10, year=2016, america=True)`. Σε αυτή την περίπτωση μπορούμε να δώσουμε τιμές στα τυπικά ορίσματα με οποιαδήποτε σειρά θέλουμε

```
dateString(year=2016, month=10, day=12, america=True)
dateString(day=12, year=2016, month=10, america=True)
dateString(america=True, day=12, year=2016, month=10)
```

Δείτε όμως ότι η κλήση `dateString(12, month=10, 2016, False)` παράγει ένα μήνυμα σφάλματος:

```
>>> dateString(12, month=10, 2016, False)
SyntaxError: positional argument follows keyword argument
```

ενώ η κλήση `dateString(12, 10, 2016, america=False)` επιστρέφει την ημερομηνία χωρίς μήνυμα σφάλματος:

```
>>> dateString(12, 10, 2016, america=False)
'12-10-2016'
```

Παρατήρηση. Εφόσον δώσουμε τιμή σε όρισμα συνάρτησης χρησιμοποιώντας το όνομά του (keyword argument) δεν μπορούμε να δώσουμε τιμή σε επόμενο όρισμα με τη μέθοδο αντιστοίχισης κατά τη θέση του ορίσματος.

Παράδειγμα. Γράψτε μία συνάρτηση στην οποία θα δίνεται όνομα και επίθετο και θα μπορεί να το τυπώνει στη μορφή *Όνομα Επίθετο* είτε *Επίθετο, Όνομα*.

```
def organizeName(firstName, lastName, reverse = False):

    if reverse == True:
        s = lastName + ', ' + firstName
    else:
        s = firstName + ' ' + lastName

    return s

# - - - Main program - - -

fName = input("Give your first name: ")
lName = input("Give your last name: ")

# call function with default argument
s = organizeName(fName,lName)
print(s)

# call using keyword arguments
s = organizeName(reverse=True, firstName='Nick', lastName='Papadakis')
print(s)
```

Χρησιμοποιήσατε ένα προεπιλεγμένο όρισμα (`reverse`). Καλέσαμε τη συνάρτηση αφήνοντας την προεπιλεγμένη τιμή στο προεπιλεγμένο όρισμα. Επίσης, καλέσαμε τη συνάρτηση χρησιμοποιώντας keyword arguments με τη σειρά προτίμησής μας (και όχι με τη σειρά που εμφανίζονται τα ορίσματα στον ορισμό της συνάρτησης).

Ερώτηση. Είναι αποδεκτή η κλήση της παραπάνω συνάρτησης ως

```
s = organizeName(reverse=True,'Nick','Papadakis');
```

Αναδρομή

Παραγοντικό, Το παραγοντικό ακεραίου nn ορίζεται ως $n!=n\cdot(n-1)\cdot\ldots\cdot1$n!=n⋅(n−1)⋅…⋅1.

```
def factorialIter(n):
    """Assumes n is int > 0
    returns n!"""
    nfact = 1
    while n > 1:
        nfact = n*nfact
        n -= 1
    return nfact
```

Είναι όμως συχνά πιο απλό να σκεφτούμε ότι το παραγοντικό $(n+1)!$(n+1)! μπορεί να υπολογιστεί άμεσα αν είναι γνωστό το $n!$n!, αφού $(n+1)!=(n+1)\cdot n!$(n+1)!=(n+1)⋅n!. Αυτό ορίζει μία αναδρομική περίπτωση (το $n!$n! υπολογίζεται αν γνωρίζουμε το $(n-1)!$(n−1)!, κλπ) και μας επιτρέπει να αναπτύξουμε μία επαγωγική μέθοδο. Η διαδικασία τερματίζεται αν γνωρίζουμε ότι $1!=1$1!=1 (βασική περίπτωση).
Παρατήρηση. Για να γράψουμε έναν κώδικα ο οποίος θα εφαρμόζει την αναδρομική μέθοδο μπορούμε να γράψουμε μία συνάρτηση η οποία θα εφαρμόζει την αναδρομική σχέση για κάθε ακέραιο nn. Αυτή η συνάρτηση μπορεί να χρησιμοποιεί την ίδια τη συνάρτηση (δηλαδή, αντίγραφο του εαυτού της) για να μπορέσει να εφαρμόσει την αναδρομική σχέση.

```
def factorialRecur(n):
    if n == 1:
        return n
    else:
        return = n*factorialRecur(n-1)
```

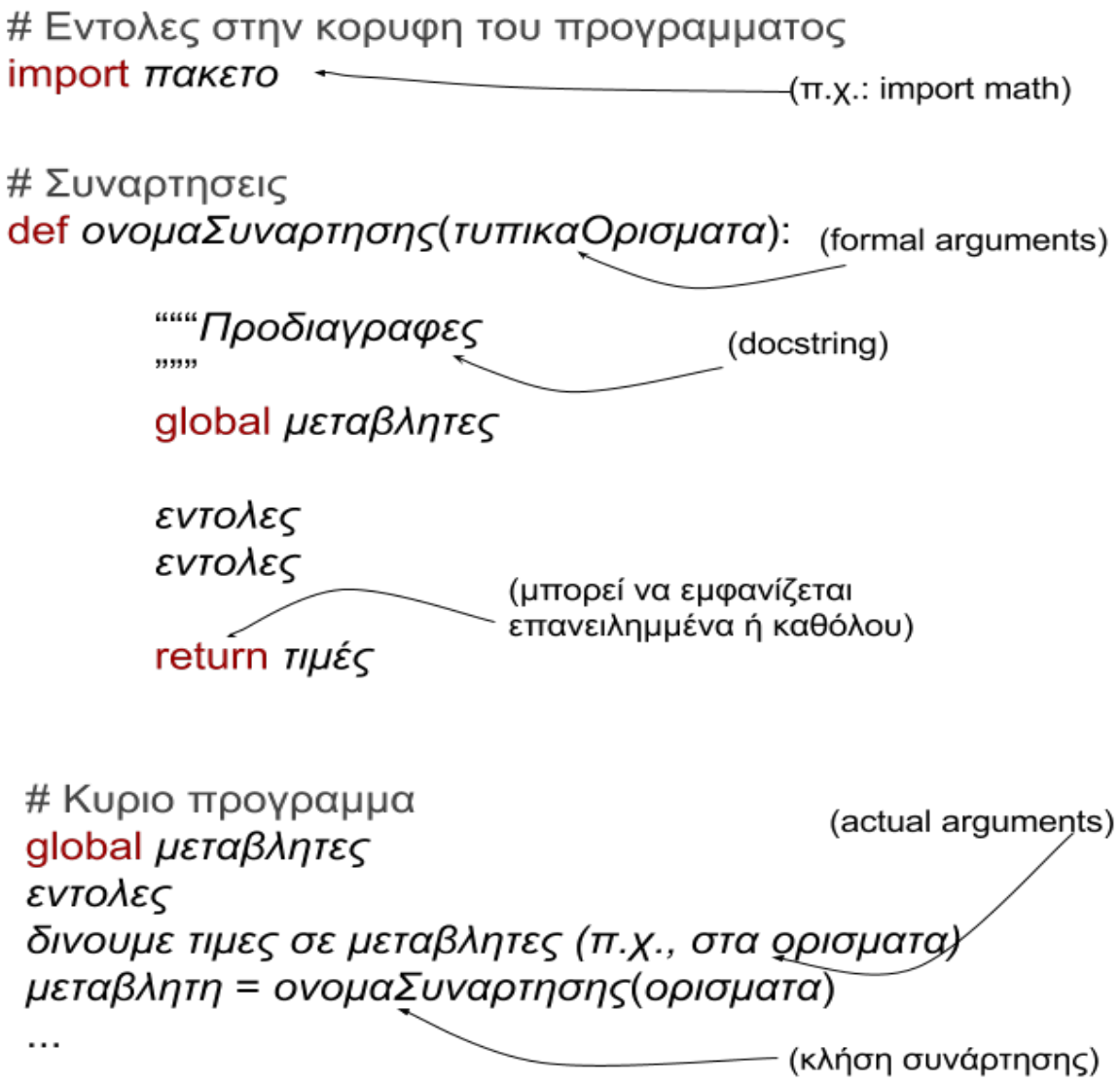
Ακολουθία Fibonacci. Αν ο πληθυσμός (κάποιου βιολογικού είδους) παρασταθεί με FiFi σε κάθε χρονιά ii, τότε έχει υποστηριχθεί ότι η αύξηση του πληθυσμού κάποιων ζώων μπορεί να παρασταθεί από την ακολουθία $Fi=Fi-1+Fi-2$Fi=Fi−1+Fi−2, δηλαδή εξαρτάται από τον πληθυσμό τα δύο προηγούμενα χρόνια. Αυτή είναι μία αναδρομική σχέση. Για την εφαρμογή ενός υπολογισμού χρειαζόμαστε μία βασική περίπτωση: θα υποθέσουμε ότι αρχικά είχαμε μόνο ένα υποκείμενο του βιολογικού είδους, δηλαδή $F0=1,F1=1$F0=1,F1=1.

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return = fib(n-1) + fib(n-2)
```

Παράδειγμα. Θα θέλαμε να μετρήσουμε πόσες κλήσεις γίνονται στη συνάρτηση `fib` μέσω της αναδρομικής κλήσης της.

```
def fib(n):
    global numFibCalls
    numFibCalls += 1
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def testFib(n):
    global numFibCalls
    numFibCalls = 0
    print("Fibonacci number for",n,' = ',fib(n))
    print("Function fib was called",numFibCalls,"times")
```

Γραφική επανάληψη



Σχήμα. Η γενική μορφή προγράμματος με συνάρτηση και κλήση της από το κύριο πρόγραμμα.