Fundamentele programării

O funcție este un bloc de instrucțiuni care are:

- un nume
- poate avea o listă de parametrii
- poate returna o valoare
- are un corp format din instrucțiuni
- are o documentație care include:
  - o scurtă descriere
  - tipul și descriere parametrilor
  - condiții impuse parametrilor de intrare
  - tipul și descrierea valorii returnate
  - condiții impuse rezultatului
  - excepții ce pot să apară

```
def max(a, b):
    """
    Compute the maximum of 2 numbers
    a, b - numbers
    Return a number - the maximum of two integers.
    Raise TypeError if parameters are not integers.
    """
    if a>b:
        return a
    return b
def isPrime(a):
    """
      Verify if a number is prime
      a an integer value  (a>1)
      return True if the number is prime, False otherwise
    """
```

Funcțiile trebuie să:

- aibă nume sugestive
- aibă specificații
- includă comentarii

O funcție ca în exemplul de mai jos este corectă sintactic dar la laborator/examen nu considerăm:

```
def f(k):
    l = 2
    while l<k and k % l>0:
        l=l+1
    return l>=k
```

Varianta acceptată este:

```
def isPrime(nr):
    """
      Verify if a number is prime
      nr - integer number, nr>1
      return True if nr is prime, False otherwise
    """
    div = 2  #search for divider starting from 2
    while div<nr and nr % div>0:
        div=div+1
    #if the first divider is the number itself than the number is prime
    return div>=nr;
```

Definiția unei funcții în Python:

Folosind instrucțiunea **def** se pot defini funcții în python.

Corpul funcției nu este executat, este doar asociat cu numele funcției.

```
def max(a, b):
    """
    Compute the maximum of 2 numbers
    a, b - numbers
    Return a number - the maximum of two integers.
    Raise TypeError if parameters are not integers.
    """
    if a>b:
        return a
    return b
```

Apel de funcții

Corpul unei funcții este un bloc de instrucțiuni care este executat în momentul în care funcția este apelată. Blocurile sunt delimitate folosind identarea.

```
max(2,5)
```

Spații de nume

- este o mapare între nume și obiecte
- are funcționalitățile similare cu un dicționar

În python sunt mai multe spații de nume:

- Implicit/General, conține denumiri predefinite
- Global, accesibil din întreaga aplicație
- Local, accesibil în interiorul funcțiilor

Transmiterea parametrilor

Parametru formal este un identificator.

Parametru actual este valoarea oferită pentru parametrul formal la apelarea funcției

- Parametrii sunt transmiși prin referință

```
def change_or_not_immutable(a):
    print ('Locals ', locals())
    print ('Before assignment: a = ', a, ' id = ', id(a))
    a = 0
    print ('After assignment: a = ', a, ' id = ', id(a))

g1 = 1      #global immutable int
print ('Globals ', globals())
print ('Before call: g1 = ', g1, ' id = ', id(g1))
change_or_not_immutable(g1)
print ('After call: g1 = ', g1, ' id = ', id(g1))
```

```
def change_or_not_mutable(a):
    print ('Locals ', locals())
    print ('Before assignment: a = ', a, ' id = ', id(a))
    a[1] = 1
    a = [0]
    print ('After assignment: a = ', a, ' id = ', id(a))

g2 = [0, 1] #global mutable list
print ('Globals ', globals())
print ('Before call: g2 = ', g2, ' id = ', id(g2))
change_or_not_mutable(g2)
print ('After call: g2 = ', g2, ' id = ', id(g2))
```

Vizibilitatea variabilelor

- variabilele definite într-o funcție au vizibilitate locală
- variabilele definite într-un modul au vizibilitate globală
- orice nume (variabile, funcții) pot fi folosite doar după ce a fost legat (prima atribuire)

```
global_var = 100

def f():
    local_var = 300
    print local_var
    print global_var
```

Domeniu de vizibilitate

- Putem folosi global pentru a referi o variabilă din spațiu de nume global în cel local

- **nonlocal** este folosit pentru a referi variabile din funcția exterioară (doar dacă avem funcții în funcții)

```
a = 100
def f():
    a = 300
    print (a)

f()
print (a)
```

```
a = 100
def f():
    global a
    a = 300
    print (a)

f()
print (a)
```

global(), **locals()** sunt funcții pentru a inspecta spațiile de nume

```
a = 300
def f():
    a = 500
    print (a)
    print locals()
    print globals()

f()
print (a)
```

Cum scriem funcții - cazuri de testare

Instrucțiunea **assert** permite inserarea de expresii care ar trebui să fie adevărate.

T1 Calculează cel mai mare divizor comun

Cazuri de testare Format tabelar

Input: (params a,b)	Output: gdc(a,b)
2 3	1
2 4	2
6 4	2
0 2	2
2 0	2
24 9	3

Funcție de test

```
def test_gcd():
    assert gcd(2, 3) == 1
    assert gcd(2, 4) == 2
    assert gcd(6, 4) == 2
    assert gcd(0, 2) == 2
    assert gcd(2, 0) == 2
    assert gcd(24, 9) == 3
```

Implementare gdc

```
def gcd(a, b):
    """
     Compute the greatest common divisor of two positive integers
     a, b integers a,b >=0
     Return the greatest common divisor of two positive integers.
    """
    if a == 0:
        return b
    if b == 0:
        return a
    while a != b:
       if a > b:
           a = a - b
       else:
           b = b - a
    return a
```