



SUBPROGRAME FUNCTII în PYTHON

SUBPROGRAME

Am folosit frecvent **funcțiile** până acum, iar ele sunt de fapt niște *subrutine*, adică *subprograme* care ne ajută la prelucrarea informației.

De exemplu, funcția **len** primește ca parametru un obiect și întoarce lungimea acestuia, fie că e șir de caractere, listă, tuplu, dicționar sau set!

Definiție: Prin **subprogram** vom înțelege *un ansamblu* alcătuit din declarații și instrucțiuni scrise în vederea unei anumite prelucrări, ansamblu implementat separat și identificat printr-un nume.

AVANTAJELE utilizării subprogramelor

Putem enumera unele dintre avantajele utilizării subprogramelor:

- **reutilizarea codului** - odată scris, un subprogram poate fi utilizat de către mai multe programe;
- **elaborarea algoritmilor prin descompunerea problemei** în altele mai simple - în acest fel, rezolvăm cu mult mai ușor problema;
- **reducerea numărului de erori** care pot apărea la scrierea programelor;
- **depistarea cu ușurință a erorilor** - verificăm la început subprogramele, apoi modul în care le-am asamblat (le-am apelat din cadrul programului);
- **realizarea unor programe ușor de urmărit** (lizibile).

Creearea unei FUNCȚII

Funcțiile pot fi grupate în **module**, adică în niște fișiere separate cu extensia ***.py**, pe care le putem include în program folosind directiva **import** (veți vedea mai târziu cum).

Pentru început, vom crea o funcție în cadrul unui program. Să pornim așadar de la un caz simplu.

Exemplu: Se citește **n**, un număr natural. Să se scrie programul care tipărește valoarea calculată a expresiei:

$$E = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

Creearea unei FUNCȚII

EXEMPLUL 1: În programul următor, vom utiliza o funcție numită **subp** (de la subprogram), creată de noi:

```
n = int(input("n="))
def subp(x):
    s = 0
    for i in range(x):
        s += 1/(i+1)
    return s
print( subp(n) )
```

```
n = int(input("n="))
def subp(x):
    s = 0
    for i in range(x):
        s += 1/(i+1)
    return s
print( subp(n) )
```

$$E = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

FUNCȚIA `subp(x)` – Explicație

Antetul funcției este "**def subp(x):**", iar corpul ei, instrucțiunea compusă subordonată (setul de instrucțiuni indentate). Funcția (subprogramul) se numește "**subp**".

Funcția are un parametru numit **x**. Rolul său este important deoarece precizează pentru ce valoare trebuie calculată expresia. Așa cum vom vedea, există posibilitatea să avem mai mulți parametri.

Funcția are variabile proprii - adică definite/declerate în interiorul ei. În exemplu, **s** și **i**. Aceste variabile se numesc **variabile locale**.

Am văzut că funcția întoarce un anumit rezultat! Observați mecanismul prin care am obținut acest lucru. Calculăm expresia în mod obișnuit. Rezultatul este reținut de variabila locală **s**. Prin instrucțiunea "**return s**", funcția a primit ca valoare de retur conținutul variabilei **s**.

Parametri formali / Parametri efectivi

În terminologia utilizată în teoria subprogramelor - în particular, în cazul funcțiilor - se utilizează termenii **parametri formali** și **parametri efectivi**.

Definiție: Parametrii care se găsesc în antetul funcției se numesc **parametri formali**.

Atunci când scriem o funcție nu cunoaștem valoarea propriu-zisă a parametrilor. Funcția trebuie să întoarcă rezultatul corect, oricare ar fi valoarea lor. Din acest punct de vedere ei se numesc *formali*.

Definiție: Parametrii care se utilizează la apel se numesc parametri **efectivi** (*argumente*).

La apel, lucrurile stau altfel: valorile acestora sunt cunoscute. Prin urmare, aceștia se numesc *parametri efectivi*. De exemplu, pentru apelul "**rez = subp(n)**", parametrul efectiv este **n**.

Citirea elementelor unei liste

EXEMPLUL 2: În exemplul de mai jos am definit o funcție care citește de la tastatură numărul de componente ale unei liste, apoi elementele sale:

```
def creare_lista():  
    n = int(input('Nr. de elemente = ?'))  
    lista_locala = []  
    for i in range(n):  
        elem = input('Elementul '+str(i)+' este:')  
        lista_locala.append(elem)  
    return lista_locala  
lista1 = creare_lista()  
print(lista1)
```

```
def creare_lista():  
    n = int(input('Nr. de elemente = ?'))  
    lista_locala = []  
    for i in range(n):  
        elem = input('Elementul '+str(i)+' este:')  
        lista_locala.append(elem)  
    return lista_locala  
lista1 = creare_lista()  
print(lista1)
```


FUNCȚIA `create_lista()` – Explicație

Funcția nu are un parametru formal, însă întoarce prin **return** o listă nouă care poate fi reținută de o variabilă din program (în acest caz, **lista1**).

Nu am specificat tipul de date reținut de elementele listei, așadar a fost **str**, cel implicit pentru funcția **input** – puteți folosi conversia explicită la citire pentru a reține un alt tip de date!

EXERSEAZĂ!!!

1. Creați o funcție care citește o listă cu elemente de tip **int**.
2. Creați o funcție care citește o listă cu elemente de tip **float**.

Citirea elementelor unei liste

EXEMPLUL 3: În următorul exemplul am definit o funcție care citește de la tastatură numărul de componente ale unei liste, apoi elementele sale, element cu element:

```
def creare_lista():
    n = int(input('Nr. de elemente = ?'))
    lista_locala = []
    for i in range(n):
        elem = input('Elementul '+str(i)+' este:')
        lista_locala.append(elem)
    return lista_locala

def afisare_lista(x):
    for i in x:
        print(i)

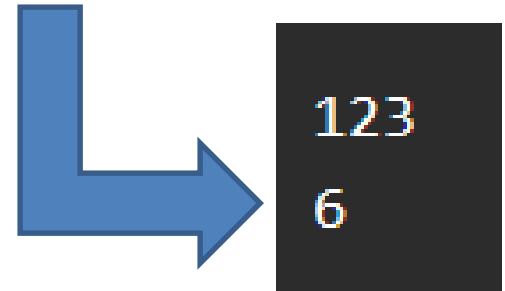
lista1 = creare_lista()
afisare_lista(lista1)
```

MAI MULȚI PARAMETRI FORMALI

EXEMPLUL 4: Până acum am folosit doar unul ori ... niciunul. Să spunem că ne dorim să realizăm o funcție care adună valorile reținute de trei variabile. Operatorul de adunare știm că poate fi folosit pentru clasele **int** / **float** și **str** (concatenează două șiruri de caractere), deci o putem generaliza:

```
def ad3(x,y,z):  
    return x+y+z  
#concatenare  
a,b,c = input(),input(),input()  
print(ad3(a,b,c))  
#adunarea numerelor intregi  
a,b,c = int(input()),int(input()),int(input())  
print(ad3(a,b,c))
```

```
def ad3(x,y,z):  
    return x+y+z  
#concatenare  
a,b,c = input(),input(),input()  
print(ad3(a,b,c))  
#adunarea numerelor intregi  
a,b,c = int(input()),int(input()),int(input())  
print(ad3(a,b,c))
```



REZOLVĂ independent/ lecții practice

Definind funcția-putere, să se calculeze valoarea expresiei:

$$S = 1 + 0,5^2 + 0,5^4 + 0,5^6 + 0,5^8.$$



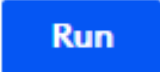
Să se definească o funcție pentru calcularea factorialului și să se calculeze cu ajutorul ei combinații din n elemente luate câte m . Numerele naturale m și n sînt date. Formula de calcul: $C_n^m = \frac{n!}{m!(n-m)!}$.

Să se definească un subprogram care va efectua:

- a) adunarea a două fracții;
- b) înmulțirea a două fracții;

EXPLICAȚII lecții practice

Ridicarea la putere a unui număr:

main.py	  	Shell
<pre>1 a = float(input("Numarul = ")) 2 b = float(input("Puterea = ")) 3 print(a, " la puterea ", b, "= ", a**b)</pre>		<pre>Numarul = 2 Puterea = 6 2.0 la puterea 6.0 = 64.0</pre>

Calcularea factorialului unui număr:

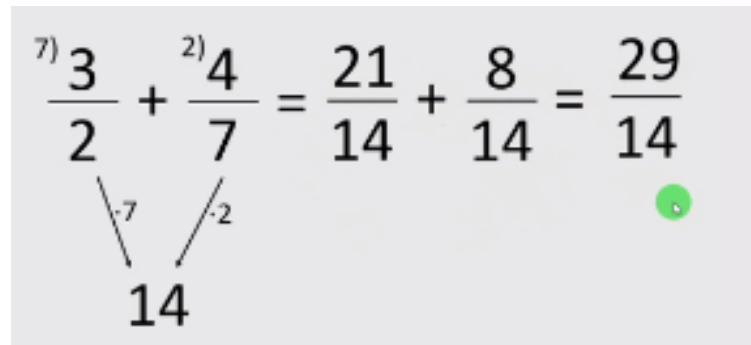
main.py	  	Shell
<pre>1 n =int(input("dati n=")) 2 fact = 1 3 4 for i in range(1,n+1): 5 fact = fact * i 6 print ("Factorialul numarului ",n," este : ",end="") 7 print (fact)</pre>		<pre>dati n=3 Factorialul numarului 3 este : 6 ></pre>

EXPLICAȚII lecții practice

Adunarea a două fracții:

$$\begin{array}{c} \overset{7)}{3} \\ \frac{3}{2} \end{array} + \begin{array}{c} \overset{2)}{4} \\ \frac{4}{7} \end{array} = \frac{21}{14} + \frac{8}{14} = \frac{29}{14}$$

14



Înmulțirea a două fracții:

$$\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$$

$$\frac{2}{7} \cdot \frac{8}{5} = \frac{2 \cdot 8}{7 \cdot 5} = \frac{16}{35}$$