

## Laboratorul 3

### Formatori:

Tutor: [Militaru Mihai-Adrian](#) 

Tutor: [Dragomir Titian-Cornel](#) 

+8

### Data de începere a cursului:

 25.09.2023

 [Utilizatori înscrși](#)

 [Calendar](#)

 [Note](#)

 [Cursurile mele](#) [S1-L-AC-CTIRO1-LSD](#) [Săptămâna 3: Recursivitate](#) [Laboratorul 3](#)

## Laboratorul 3

### 1. Ce este recursivitatea?

**Recursivitatea** este un concept fundamental în matematică și în programare ce presupune ca în definirea unei noțiuni să se facă referire la ea însăși.

**FOARTE IMPORTANT!** În cadrul oricărei definiții recursive trebuie să existe măcar un caz de bază (elementar), la care să se ajungă după parcurgerea unui anumit număr de pași.



### 2. Funcții recursive

Știm deja din laboratoarele anterioare că putem defini funcții care în interiorul lor pot apela alte funcții. Putem chiar să definim funcții care să se apeleze pe ele însăși. O astfel de funcție care apare în propria sa definiție poartă numele de funcție recursivă.

#### Exemplu: Factorial

Ca și exemplu vom încerca să calculăm factorialul recursiv pentru numărul  $n$ . Pornim de la relația recursivă a acestuia:

$$factorial(n) = n! = \begin{cases} n \times (n-1)! & \text{pentru } n \neq 1 \text{ sau } n \neq 0 \\ 1 & \text{pentru } n = 1 \text{ sau } n = 0 \end{cases}$$

Observăm că dacă vrem să calculăm termenul  $n!$ , trebuie să aflăm întâi termenul  $(n-1)!$

$$(n-1)! = (n-1) \times (n-2)!$$

Pentru a afla termenul  $(n-1)!$  trebuie să determinăm întâi termenul  $(n-2)!$

$$(n-2)! = (n-2) \times (n-3)!$$

Remarcăm că acest proces se repetă, atfel că pentru a calcula valoarea termenului curent mereu vom avea nevoie de termenul anterior, iar când încercăm să îl calculăm pe acesta avem nevoie de termenul care este înaintea lui.





Repetând acest proces vom ajunge la un moment dat să calculăm valorile:

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

$$2! = 2 \times 1!$$

Este cunoscut faptul că  $1! = 1$  și  $0! = 1$ . Prin urmare acum știm tot ce ne trebuie. Deoarece am ajuns la un caz pe care îl cunoaștem (un caz de bază) și anume  $1!$ , putem să ne întoarcem și să îi calculăm pe  $2!$ ,  $3!$ , ...,  $(n-1)!$  și în cele din urmă pe  $n!$ .

Vom încerca acum să definim o funcție care să facă tot acest calcul al factorialului. Așa cum am stabilit, dorim să calculăm factorialul pentru un număr  $n$ , prin urmare acesta va fi parametrul funcției noastre.

```
def factorial(n):
    # VOM COMPLETA CU COD

rezultat=factorial(6)
```

Acum că ne-am definit antetul funcției, trebuie să stabilim ce instrucțiuni vom scrie. Dacă ne amintim ce am făcut la început, pentru a-l calcula pe  $n!$  a fost nevoie să facem  $n \times (n-1)!$ . Cine este  $(n-1)!$ ? Îl putem scrie ca rezultatul apelului funcției noastre care primește drept argument  $(n-1)!$ .

```
def factorial(n):
    return n*factorial(n-1)    # AM SCRIS RELAȚIA RECURENTĂ

rezultat=factorial(6)
```

Practic, în momentul de față, în funcția noastră am reușit să implementăm acel șir de calcul al factorialului pe care l-am parcurs anterior, deoarece funcția noastră se va autoapela pentru a calcula termenul anterior. Ce mai lipsește? Condiția de oprire (cazul de bază). Noi am știut că trebuie să ne oprim atunci când am ajuns să calculăm  $1! = 1$  și  $0! = 1$ . Prin urmare ar trebui să specificăm în cadrul funcției că dacă parametrul  $n$  este 1 sau 0, rezultatul factorialului este 1.

```
def factorial(n):
    if n==0 or n==1:          # AM DEFINIT CAZUL DE BAZĂ
        return 1
    return n*factorial(n-1)    # AM SCRIS RELAȚIA RECURENTĂ

rezultat=factorial(6)
```

#### Exemplu: Suma cifrelor unui număr

În cadrul problemelor recursive în care se cere să se prelucereze cifrele unui număr, putem considera că numărul este format fie dintr-o singură cifră (cazul de bază), fie dintr-un șir de cifre ( $n//10$ ) ce precede ultima cifră ( $n\%10$ ). Bazându-ne pe această definiție recursivă a unui număr în baza 10 putem foarte ușor să ne gândim la cum am scrie o funcție recursivă ce calculează suma cifrelor unui număr:

$$suma(n) = \begin{cases} n\%10 + suma(n//10) & \text{pentru } n > 9 \\ n & \text{pentru } n \leq 9 \end{cases}$$

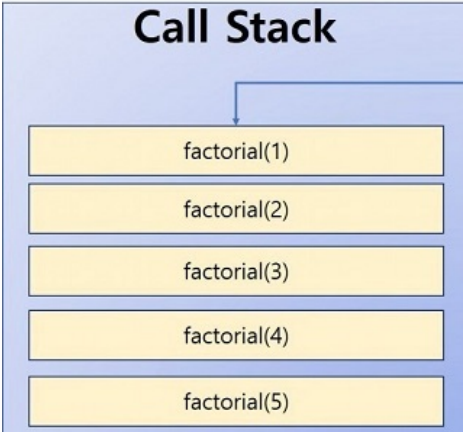
Pentru cazul de bază (când numărul este format dintr-o singură cifră) evident că această sumă ar fi chiar unica cifră a numărului.

```
def suma_recursiva(n):
    if n <= 9:                # CAZUL DE BAZĂ
        return n
    else:
        return n % 10 + suma_recursiva(n // 10)    # // ÎMPARTE ȘI ROTUNJEȘTE ÎN JOS NUMĂRUL OBTINUT

print(suma_recursiva(123456))
```

### 3. Cum funcționează recursivitatea?

Atunci când facem un apel de funcție în Python, valorile parametrilor și al variabilelor locale, precum și locația de unde este efectuat apelul de funcție, sunt puse într-o stivă de apeluri (call stack). Stiva este o structură de date de tipul LIFO (Last In First Out) ceea ce înseamnă că vom adăuga un nou element doar în vârful stivei și vom lua un element doar din vârful stivei.



Spre exemplu, apelul funcției factorial: deoarece factorial(1) este ultimul apel adăugat la stiva, acesta este scos primul din stiva și procesat. Apoi, celelalte apeluri sunt scoase din stivă și procesate: factorial(2), factorial(3), factorial(4), factorial(5).

[◀ Curs 3 - Recursivitate, Multimi inductive, Pattern Matching, Tail Recursion](#)


Sari la...

[Exerciții - Săptămâna 3 ▶](#)

[✉ Contactați serviciul de asistență](#)

Sunteți conectat în calitate de Ciobanu Daria-Andreea (Delogare)  
S1-L-AC-CTIRO1-LSD

Meniul meu

- [Profil](#)
- [Preferinte](#)
- [Calendar](#)
-  [ZOOM](#)
- [Română \(ro\)](#)
- [English \(en\)](#)
- [Română \(ro\)](#)

[Rezumatul păstrării datelor](#)  
[Politici utilizare site](#)