



FUNCTȚII în PYTHON

- Poziția și ordinea funcțiilor
- Variabile locale și globale

POZIȚIA ȘI ORDINEA SCRIERII FUNCȚIILOR

Ce înseamnă de fapt un limbaj interpretat?

Limbajul de programare conține instrucțiuni care respectă anumite **reguli de sintaxă** și putem programa calculatorul să efectueze o **succesiune de operații** în vederea obținerii unui **rezultat**.

Instrucțiunile se redactează într-un limbaj aproape de cel natural, de cele mai multe ori în limba engleză. Se obține astfel **codul sursă**.

În funcție de limbajul folosit, codul sursă este transformat în cod mașină folosind un **compiler** sau un **interpretor**.

POZIȚIA ȘI ORDINEA SCRIERII FUNCȚIILOR



Compilerul scanează și analizează tot codul sursă, apoi îl transformă pe tot în cod mașină sub forma unui program executabil. Deși este mai rapid, erorile programului sunt afișate la final, depanarea fiind puțin mai dificilă.

Interpreterul transformă în cod mașină linie cu linie codul, nefiind necesar un fișier executabil generat la final. La prima eroare analiza se oprește, fiind astfel mai ușor de depanat programele.

Limbajul Python *este interpretat*, deci atunci când executăm un program, acesta preia secvențial comandă după comandă și *încearcă rularea lor*.

UNDE PUTEM DEFINI O FUNCȚIE?

Nu putem apela o funcție care încă nu a fost definită!

main.py	  	Shell
<pre>1 a='125' 2 tiparire(a) 3 4 def tiparire(x): 5 print(x)</pre>	<pre>Traceback (most recent call last): File "<string>", line 2, in <module> NameError: name 'tiparire' is not defined > </pre>	

Variabila **a** primit valoarea **125**, apoi a fost apelată o funcție inexistentă. Așadar, ***creăm funcția înainte și apoi o putem apela în program!***

ORDINEA DEFINIRII MAI MULTOR FUNCȚII

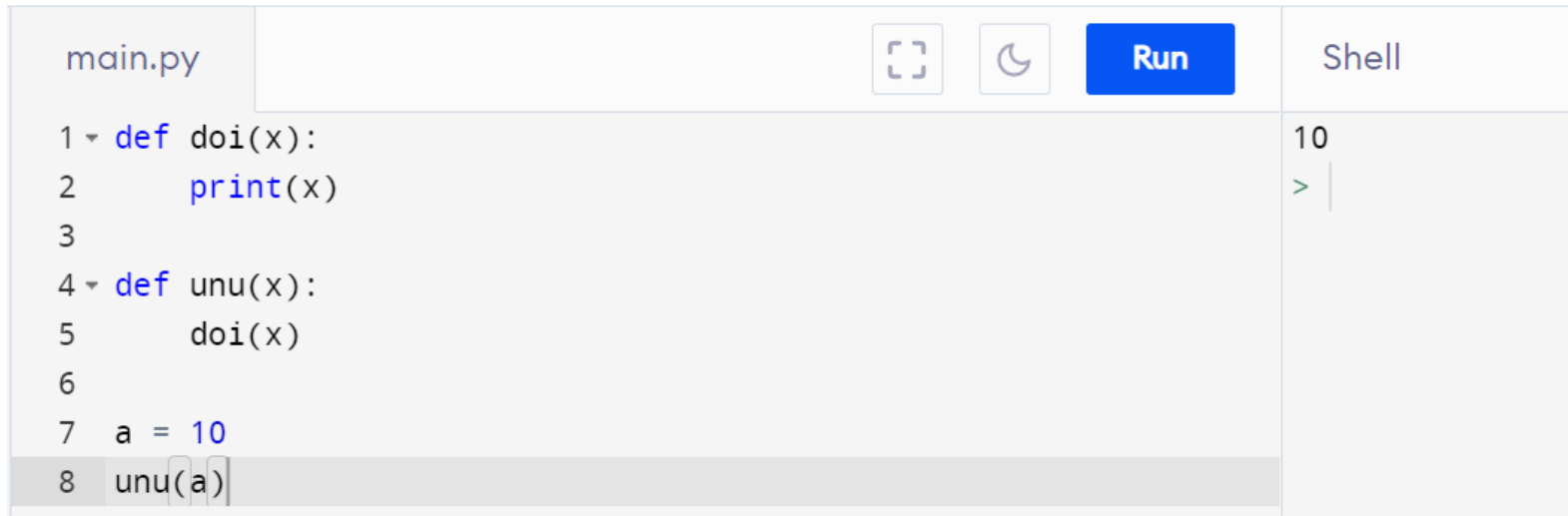
Atunci când programele noastre devin destul de mari, **ordinea definirii subprogramelor este importantă**, deoarece putem din greșeală să le poziționăm astfel:

main.py			Run	Shell
<pre>1 def unu(x): 2 doi(x) 3 4 a = 10 5 unu(a) 6 7 def doi(x): 8 print(x)</pre>				<pre>Traceback (most recent call last): File "<string>", line 5, in <module> File "<string>", line 2, in unu NameError: name 'doi' is not defined > </pre>

A fost reținută definiția funcției **unu**, apoi variabilei **a** i s-a atribuit valoarea **10**. Se apelează funcția **unu**, însă până în acel moment, definiția funcției **doi** este inexistentă, deci rezultă o eroare la interpretare!

Funcțiile nu sunt executate de interpretorul Python, ci doar reținute ca definiții în memorie, deci ordinea acestora nu contează, *atât timp cât sunt definite înaintea apelului!* Încercați să definiți funcția **doi** înaintea funcției **unu** și programul va funcționa corect!

ORDINEA DEFINIRII MAI MULTOR FUNCȚII



The screenshot shows a Python IDE window titled 'main.py'. The code in the editor is as follows:

```
1 def doi(x):  
2     print(x)  
3  
4 def unu(x):  
5     doi(x)  
6  
7 a = 10  
8 unu(a)
```

At the top right of the editor, there are icons for a file explorer, a search icon, and a blue 'Run' button. To the right of the editor is a 'Shell' pane, which currently displays the output '10' and a prompt '> |'.

Și în programare lucrurile sunt foarte fine. Cine spune că Python este ușor, se înșeală amarnic, precum observați. Multe aspecte trebuie luate în considerare, noțiunile teoretice trebuie stăpânite solid și vă vor oferi cu siguranță satisfacții în lumea coding-ului.

VARIABLE LOCALE ȘI GLOBALE

Mai jos am creat o variabilă **x** și i-am atribuit valoarea întreagă **6**. Am definit apoi o funcție, numită **inc**, în speranța că **x** va fi incrementat cu o unitate, însă rezultatul poate fi surprinzător pentru unii dintre voi!

main.py	Run	Shell
<pre>1 x = 6 2 print(x) 3 4 def inc(x): 5 x = x + 1 6 7 inc(x) 8 print(x)</pre>		<pre>6 6 > </pre>

Cu ce am greșit? Am totuși o vagă bănuială... anterior am prezentat faptul că funcțiile au variabile proprii, care se numesc **variabile locale**, *ce nu pot fi accesate din exterior*.

Parametrul, respectiv variabila **x** din corpul funcției, este un obiect complet diferit de celălalt **x** din program după atribuire, deci nu este recunoscut în afara funcției!

Cum putem rezolva totuși problema?

CUVÂNTUL CHEIE "GLOBAL"

Trebuie să anunțăm funcția anterioară despre faptul că variabila noastră **x** este cea din exterior și vom utiliza cuvântul cheie **global**, precum mai jos:

main.py	Run	Shell
<pre>1 x = 6 2 print(x) 3 4 def inc(): 5 global x 6 x = x + 1 7 8 inc() 9 print(x)</pre>		<pre>6 7 ></pre>

Nu a mai fost necesar niciun parametru ori argument la apel în acest caz. Dacă ar fi existat, ar fi apărut o eroare ... **x** nu putea fi și *variabilă locală* și *globală* în același timp în interiorul funcției **inc**:

SyntaxError: name 'x' is local and global on line 4

CUVÂNTUL CHEIE "GLOBAL"

Așadar, o variabilă definită în exteriorul funcției este globală implicit – trebuie doar să specificăm acest fapt în interior.



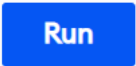
Variabilele declarate în interiorul funcțiilor sunt numite **variabile locale** - mai precis, pot fi declarate în orice bloc (instrucțiune compusă) din cadrul acestora.

Trebuie să folosim cuvântul cheie **global** pentru a scrie și a citi o variabilă din afară în interiorul unei funcții.

Folosirea lui **global** în afara funcțiilor nu are niciun efect.

VARIABLELE "GLOBALE"

EXEMPLU: Vom crea o funcție care adună o valoare trimisă ca argument unei variabile definită în program:

main.py	  	Shell
<pre>1 x = 10 2 print(x) 3 4 def adunare(a): 5 global x 6 x = x + a 7 8 adunare(7) 9 print(x)</pre>		<pre>10 17 > </pre>

Variabila **x** este *globală*, iar variabila **a** este locală (parametrul). Funcția primește ca argument valoarea **7** reținută apoi de **a** și folosită la operația de adunare cu **x**.

VARIABLE NELOCALE

În programele voastre cu siguranță veți folosi funcții care se vor regăsi în interiorul altor funcții – acestea se numesc *imbricate*. Ce ne facem în situația în care vom folosi variabile cu același nume în cadrul lor? Dacă ne dorim însă ca *variabilele să nu fie locale, însă nici globale?*

Să analizăm programul următor:

```
main.py  [ ] [ ] [Run] [Shell]

1 x='Program' #variabila globala
2
3 def extern():
4     x='Local'
5     def intern():
6         nonlocal x
7         x='Non Local'
8         print(x)
9     intern()
10    print(x)
11
12 extern()
13 print(x)
```

Non Local
Non Local
Program
> |

În cadrul funcției **extern()** se află subprogramul **intern()**. Deoarece **x** a fost declarat **nonlocal**, valoarea lui **x** din funcția **extern()** s-a modificat, însă variabila globală din programul principal nu... Deci, dacă modificăm variabila nelocală, are efect și asupra celei locale de la nivelul anterior.

LECȚII PRACTICE

Să se definească funcțiile $\max(a, b)$ și $\min(a, b)$, care returnează respectiv cel mai mare și cel mai mic dintre numerele reale a și b , apoi să se calculeze valoarea expresiei:

a) $S = \max(\min(a_1, a_2), \max(a_3, a_4)) + \min(\max(a_5, a_6), \min(a_7, a_8))$, unde a_1, a_2, \dots, a_8 sînt numere reale date;

b) $T = \min(a_1, a_2) + \min(a_3, a_4) + \dots + \min(a_9, a_{10}) + \max(a_1, a_2) + \max(a_3, a_4) + \dots + \max(a_9, a_{10})$, unde a_1, a_2, \dots, a_{10} sînt numere reale date.

Se dau numerele reale pozitive a, b, c , care sînt lungimile laturilor unui triunghi. Să se calculeze lungimile medianelor triunghiului.

Indicație. Lungimea medianei corespunzătoare laturii de lungimea a se calculează cu ajutorul formulei $m_a = 0,5\sqrt{2b^2 + 2c^2 - a^2}$.