



FUNCTȚII în PYTHON

- Universul modulelor

(Random, Randint, Choice, Seed)

UNIVERSUL MODULELOR

Setul de funcții pe care îl conține implicit limbajul Python nu este foarte mare, însă acesta este extins prin **module** (*colecții de funcții*, numite în alte limbaje, *biblioteci*). Și voi puteți crea module personale salvând funcțiile dezvoltate într-un fișier separat cu extensia **"py"**.

Modulele pe care le vom prezenta fac parte din *biblioteca standard* a limbajului Python, numită în engleză **Python Standard Library**, care conține o multitudine de alte module utile:

<https://docs.python.org/3/library/>

Nu avem cum să le analizăm pe toate aici, ci trebuie doar să știm de existența acestora și în caz de nevoie, să ne documentăm corespunzător, apoi să folosim subrutinele unui anume modul care ne pot fi de folos!

MODULUL RANDOM



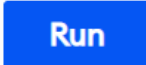
Jocurile pe calculator conțin numere la întâmplare, numite **aleatoare**, astfel încât de fiecare dată pare că totul este altfel, *un nou scenariu*, un nou set de culori, bile ori inamici care apar din poziții diferite.

Un **număr aleator** este în esență unul care nu poate fi prezis de utilizator. De fapt, în spate există **un generator (o funcție)** care ne oferă la fiecare apelare un astfel de număr. Ele sunt *pseudo-aleatoare*, în sensul că sunt create în urma unor *calcule matematice complexe*.

Unul dintre acestea este **random**, creat special pentru a *genera numere aleatoare*.

MODULUL RANDOM- funcția RANDOM

Pentru a putea folosi funcțiile sale, la începutul programului vom scrie "**import random**", care încarcă modulul și ne oferă acces la el:

main.py	  	Shell
<pre>1 import random 2 # pentru intervalul [0.0, 1.0) vom tipări 5 numere aleatoare 3 for x in range(5): 4 print(random.random())</pre>		<pre>0.7311493380789899 0.40921407446843117 0.6322473876964327 0.0640276385963624 0.47839265376490725</pre>



Am importat modulul **random**, apoi am tipărit seturi de câte cinci numere aleatoare folosind instrucțiunea **for** pentru iterații. Pentru a apela o funcție din cadrul unui modul, scriem:

nume_modul.funcția_dorită

Prima oară am folosit chiar funcția **random()**, care generează *un număr real*, de tip **float**, în intervalul **[0.0, 1.0)** – funcția nu acceptă parametri.

MODULUL RANDOM- funcția RANDOM

Bineînțeles, nu ne prea ajută aceste numere... Cum putem însă afișa numere aleatoare întregi între **0** și **100**?

main.py			Run	Shell
1 import random				52
2 # pentru intervalul [0, 100) vom tipări 6 numere aleatoare				79
3 for x in range(6):				88
4 print(round(random.random()*100))				23
				88
				13

Dacă a fost generat **0.59234...** Înmulțindu-l cu **100**, se obține **59.234**, care apoi rotunjit spre un număr întreg devine **59**.

MODULUL RANDOM- funcția RANDOM

Dar dacă este necesar de generat numere în intervalul de la **1** la **100**? În acest caz adunăm **1** la valoarea generată până la **99**, ca în exemplul de mai jos:

main.py	Run	Shell
1 import random		44
2 # pentru intervalul [0, 100) vom tipări 6 numere aleatoare		7
3 for x in range(6):		42
4 print(round(random.random()*99 + 1))		75
		100
		7

MODULUL RANDOM- funcția RANDOM

Cum putem însă afișa numere aleatoare întregi între **0** și **1000**?

main.py					Shell
1	<code>import random</code>				737
2	<code># pentru intervalul [0, 1000) vom tipări 4 numere aleatoare</code>				115
3	<code>for x in range(4):</code>				18
4	<code>print(round(random.random()*1000))</code>				239

Dacă a fost generat **0.59234...** Înmulțindu-l cu **1000**, se obține **592.34**, care apoi rotunjit spre un număr întreg devine **592**.

MODULUL RANDOM- funcția RANDOM

Cum putem genera numere întregi cuprinse într-un anumit interval, de exemplu între **30** și **90**? Putem tipări, precum anterior, numere între **0** și **60**. Doar adunăm **30** pentru fiecare număr în parte apoi:

main.py			Run	Shell
1 <code>import random</code>				47
2 <code># pentru intervalul [30, 90) vom tipări 5 numere aleatoare</code>				36
3 <code>for x in range(5):</code>				44
4 <code>print(round(30 + random.random()*60))</code>				85
				59
				>

MODULUL RANDOM- funcția RANDINT

Modulul **random** conține o serie consistentă și avansată de funcții pentru a genera *numere aleatoare*.

Funcția randint ne scutește de calculele anterioare pentru a obține numere întregi.

Funcția **randint(început,sfârșit)** întoarce un număr aleator întreg cu semn (deci tipul **int**) din intervalul specificat de parametri.

main.py			Run	Shell
<pre>1 import random 2 #generare 5 numere întregi cu semn în intervalul [-100,100] 3 for x in range(5): 4 print(random.randint(-100,100))</pre>				<pre>-13 -42 -60 75 -32 > </pre>


main.py			Run	Shell
<pre>1 import random 2 #numere întregi cu semn 3 for x in range(1): 4 print(random.randint(2,9))</pre>				<pre>6 > </pre>

MODULUL RANDOM- funcția CHOICE

Putem alege unul dintre obiectele reținute de o listă sau un șir de caractere, folosind funcția **choice(colecție_de_date)**. Funcția întoarce la întâmplare un element din cadrul obiectului.

main.py		Shell
<pre>1 import random 2 #alegem aleator 4 elemente dintr-o listă 3 Nume = ['Ion', 'Maria', 'Anatol', 'Mihai', 'Ana', 'Ecaterina'] 4 for x in range(4): 5 print(random.choice(Nume))</pre>		<pre>Ion Ecaterina Maria Ion > </pre>



main.py		Shell
<pre>1 import random 2 #alegem aleator 3 vocale dintr-o listă 3 vocale = ['a', 'e', 'i', 'o', 'u'] 4 for x in range(3): 5 print(random.choice(vocale))</pre>		<pre>i u a > </pre>

main.py		Shell
<pre>1 import random 2 #alegem aleator 1 număr dintr-o listă 3 număr = ['2', '3', '4', '5', '6', '7', '8', '9'] 4 for x in range(1): 5 print(random.choice(număr))</pre>		<pre>6 > </pre>

MODULUL RANDOM- funcția SEED

Funcția **seed(nr_intreg_optional)** este utilă atunci când dorim să **pornim generatorul de la aceeași valoare** – potrivită pentru *testarea programelor* noastre. Dacă folosim același argument de mai multe ori, vom obține de fiecare dată o *serie identică de numere aleatoare*.

Exemplu. Folosim **seed()** pentru a tipări cinci numere aleatoare, iar ca parametru vom specifica, să spunem, **17**:

main.py				Run	Shell
1	import random				9
2	random.seed(17)				7
3	for x in range(5):				5
4	print(random.randint(1,10))				6
					5
					>

MODULUL RANDOM- funcția SEED




Am specificat prin **seed()** de fapt **sămânța întâmplării**, un loc de pornire pentru numerele aleatoare create ulterior. Dacă nu folosim funcția **seed()**, ori o apelăm fără parametrul opțional, valoarea implicită a argumentului este *timpul curent al sistemului*, de fiecare dată vom avea numere diferite și la întâmplare.

De ce este utilă? Să spunem că realizăm un joc și folosim numere aleatoare. Pentru a putea evalua funcționarea programului, modul întâmplător indus, analiza valorilor obținute, avem nevoie ca de fiecare dată, în regim de test, să repornim generatorul, astfel încât la executarea programului, numerele să fie la fel. Bineînțeles că la final, când totul funcționează corect, ștergem apelul funcției **seed()**...

MAI MULTE MODULE

Pentru a include un modul în programul nostru, am văzut că folosim directiva **import**, urmată de numele bibliotecii: **import random**. Putem include mai multe module într-o singură comandă: **import random, math** sau doar o parte dintre subrutinele conținute de un modul: **from random import randint**

Deoarece dorim să fim eficienți – includem doar ce ne este necesar. În acest ultim caz, nu includem tot modulul, ci doar funcțiile specificate, așadar apelul se va face direct, fără a mai fi precedat de numele modulului:

main.py	  	Shell
<pre>1 from random import seed, randint 2 seed(15) 3 for x in range(3): 4 print(randint(1,10))</pre>		<pre>4 1 9 ></pre>

Lista modulelor din biblioteca PYTHON standard

Așadar, accesați linkul de mai jos:

<https://docs.python.org/3/py-modindex.html>

veți găsi lista completă a modulelor din *biblioteca standard* și o scurtă descriere a acestora.

Python » English ▾ 3.10.7 ▾ 3.10.7 Documentation » Python Module Index Quick search Go | modules | index

Python Module Index

[_](#) | [a](#) | [b](#) | [c](#) | [d](#) | [e](#) | [f](#) | [g](#) | [h](#) | [i](#) | [j](#) | [k](#) | [l](#) | [m](#) | [n](#) | [o](#) | [p](#) | [q](#) | [r](#) | [s](#) | [t](#) | [u](#) | [v](#) | [w](#) | [x](#) | [z](#)

[_](#)
[__future__](#)
[__main__](#)
[_thread](#)

Future statement definitions
The environment where top-level code is run. Covers command-line interfaces, import-time behavior, and ``__name__ == '__main__'``.
Low-level threading API.

a
[abc](#)
[aifc](#)
[argparse](#)
[array](#)



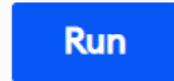
Abstract base classes according to :pep:`3119`.
Deprecated: *Read and write audio files in AIFF or AIFC format.*
Command-line option and argument parsing library.
Space efficient arrays of uniformly typed numeric values.

Rezolvă INDEPENDENT

Pentru problema problema de mai jos (tema pentru acasă de la lecția precedentă) – să se genereze numerele n_1 , n_2 și baza b – în baza exemplului:

Se dau numerele naturale b și n , unde $1 < b < 10$. Să se definească un subprogram care va:

- a) verifica dacă un număr dat este scris corect în sistemul de numerație cu baza b ;
- b) aduna două numere scrise în sistemul de numerație cu baza b ;
- c) scădea două numere scrise în sistemul de numerație cu baza b ;
- d) înmulți două numere scrise în sistemul de numerație cu baza b ;

main.py	  	Shell
<pre>1 import random 2 #numere întregi cu semn 3 for x in range(1): 4 print(random.randint(2,9))</pre>	6 >	

LECȚII PRACTICE *20/21.10.2025

1. Să se afișeze suma valorilor pozitive și suma valorilor negative din n numere generate în intervalul $[-50,50]$. Numărul de elemente n – este introdus de la tastatură.
Exemplu: Date de intrare n=6 numere 6 9 -8 7 -5 -3 Date de ieșire: Spoz= 22 Sneg=-16.
2. Să se simuleze aruncarea unui zar de n ori afișându-se valoarea feței și să se afișeze de câte ori a apărut valoarea 6.
3. Se generează un număr întreg de maxim 9 cifre, să se afișeze numărul de apariții al fiecărei cifre. Exemplu : Date de intrare 364901211 Date de ieșire 0 apare de 1 ori 1 apare de 3 ori 2 apare de 1 ori 3 apare de 1 ori 4 apare de 1 ori 5 apare de 0 ori 6 apare de 1 ori 7 apare de 0 ori 8 apare de 0 ori 9 apare de 1 ori.
4. Se aruncă 2 zaruri până la obținerea unei duble. Să se afișeze suma punctelor.
5. Se generează un tablou A[1..N] de numere întregi, $N \leq 100$. Numărul de elemente n – este introdus de la tastatură. Scrieți un program care calculează suma elementelor mai mici ca elementul maxim.

```
1 import random
2 k=[]
3 for x in range(5):
4     k.extend([random.randint(-100,100)])
5 print(k)
```

```
[-38, 24, 68, 86, 26]
```

```
>
```


TEMĂ PENTRU ACASĂ *03/04.11.2025

1. Să se alcătuiască un joc-test de verificare:
 - a) a adunării a două numere generate de la 0 la 100 ($a+b$)
 - b) a scăderii a două numere generate de la 0 la 100 ($a-b$, $b-a$)
 - c) a înmulțirii a două numere generate de la 0 la 100 ($a*b$)
2. Să se genereze 10 variante „**Superloto 5 din 36**”.
3. Să se genereze un număr natural din 3 cifre, divizibil cu 6 și cu 8.
4. Să se modeleze jocul „**Vreau să fiu milionar**” pentru 5 teme și 10 întrebări pentru fiecare temă.

NOTĂ: Problemele 1-3 (pentru nota 8), Problemele 1-4 (pentru nota 9-10)