



Școala  
informală  
de IT

# Python Development

Pythonic “Hello, World!”



Școala  
informală  
de IT

# Cuprins

1. Program “Hello, World!”
2. Sintaxă de bază
3. Numere
4. Operatori
5. Variabile
6. Șiruri de caractere



# Program “Hello, World!”

Pythonic “Hello, World!”



# Program “Hello, World!”

- Programul “**Hello, World!**” este cel mai simplu program din lume.
- Este un program minimal al cărui rol este să exemplifice sintaxa și modul de funcționare al unui limbaj.
- Este folosit în procesul de învățare al unui limbaj de programare nou.
- Rezultatul programului este afișarea mesajului **Hello, World!** pe ecranul unui calculator.
- Pentru a scrie acest program, deschideți interpretorul Python și scrieți următoarea instrucțiune:

```
print("Hello, World!")
```

- Rezultatul este uimitor! În terminal apare mesajul **Hello, World!** ca prin magie.
- Din acest moment puteți dezvolta orice program doriți folosind Python. Tot ce trebuie să faceți este să schimbați mesajul pe care să-l afișați.



# Program “Hello, World!”

- Folosind interpretorul Python în mod interactiv, avem dezavantajul de a nu putea refolosi codul.
- Pentru a dezvolta mai ușor programe Python și pentru a putea refolosi codul trebuie să creăm un script.
- Un script Python este un fișier cu extensia **.py** ce poate fi rulat de sine stătător. Acest fișier reprezintă programul nostru.
- Pentru scrierea codului Python poate fi folosit orice editor de texte. Pentru o viață mai ușoară, un mediu de dezvoltare mai rapid și mai performant vom folosi IDE-ul PyCharm.
- După crearea fișierului .py acesta poate fi rulat folosindu-vă de interpretorul Python în Script Mode. Folosiți interpretorul Python urmat de numele scriptului creat.
- Acesta va produce un rezultat similar cu executarea codului în mod interactiv.



# Sintaxă de bază

Pythonic “Hello, World!”



# Sintaxă de bază

- Linia `print("Hello, World!")` reprezintă o instrucțiune Python.
- Cele mai importante reguli de sintaxă în programarea Python sunt:
  - o linie din program poate conține o singură instrucțiune. Instrucțiunile nu se termină cu nici un caracter sau grup de caractere. Pot exista mai mult instrucțiuni pe aceeași linie atâta timp cât nici una din ele nu necesită un bloc de instrucțiuni. În acest caz acestea vor fi separate de caracterul `;`.
  - blocurile de cod sunt obținute prin indentare (nu se folosesc nici un fel de paranteze sau alte caractere).
  - o linie de comentariu începe cu semnul `#`. Un comentariu poate fi adăugat la finalul unei instrucțiuni, pe aceeași linie cu aceasta. Comentariile nu sunt cod recunoscut și rulat de interpretor.



# Numere

Pythonic “Hello, World!”





# Numere

- Numerele reprezintă un **TIP DE DATĂ**.
- În Python există trei tipuri de numere:
  - întregi (**int**) - numere întregi, fără virgulă, negative sau pozitive de lungime nelimitată:
    - 3, 5, 20, -17
  - raționale/zecimale (**float**) - numere cu virgulă, negative sau pozitive care conțin una sau mai multe zecimale (cifre după virgulă). Pot fi folosite valori științifice unde **e** reprezintă puteri ale lui 10.
    - 2.4, 3.0, -7.65, 8.46315, 5e2, -4e3, 1e-1
  - complexe (**complex**) - numere complexe care sunt reprezentate folosind litera **j** pentru definirea părții imaginare:
    - 1j, 3+5j, -8j
- Oricare din tipurile de numere de mai sus poate fi convertit la un alt tip folosind funcțiile **int()**, **float()**, respectiv **complex()**. Numerele complexe nu pot fi convertite în alt tip de numere.
- Tipul unei date/valori poate fi aflat folosind metoda **type()**.



# Operatori

Pythonic “Hello, World!”



# Operatori aritmetici

Operator	Nume	Detalii	Exemplu	Rezultat
+	adunare	Adunarea matematică a două sau mai multe valori.	5 + 2	7
-	scădere	Scăderea matematică a două sau mai multe valori.	5 - 7	-2
*	înmulțire	Înmulțirea matematică a două sau mai multe valori.	3 * 4	12
/	împărțire	Împărțirea matematică a două sau mai multe valori.	5 / 2	2.5
%	modulo	Restul împărțirii unei valori la o altă valoare	7 % 4	3
**	ridicare la putere	Ridicarea unei valori la o anumită putere.	2 ** 4	16
//	împărțire exactă	Câtul împărțirii a două valori.	7 // 4	1



# Operatori de comparare

- Orice comparație făcută va avea ca rezultat o dată de tip **Boolean**:
  - **True** - dacă rezultatul comparației are valoare de adevăr.
  - **False** - dacă rezultatul comparației nu are valoare de adevăr.

Operator	Nume	Descriere	Exemplu
==	egalitate	Este evaluată ca True dacă valoarea din stânga este egală cu valoarea din dreapta, altfel este evaluată ca False.	5 == 5
!=	inegalitate	Este evaluată ca True dacă valoarea din stânga este diferită față de valoarea din dreapta, altfel este evaluată ca False.	7 != 5
>	mai mare	Este evaluată ca True dacă valoarea din stânga este mai mare decât valoarea din dreapta, altfel este evaluată ca False.	5 > 3
<	mai mic	Este evaluată ca True dacă valoarea din stânga este mai mică decât valoarea din dreapta, altfel este evaluată ca False.	6 < 8
>=	mai mare sau egal	Este evaluată ca True dacă valoarea din stânga este mai mare sau egală cu valoarea din dreapta, altfel este evaluată ca False.	7 >= 7
<=	mai mic sau egal	Este evaluată ca True dacă valoarea din stânga este mai mică sau egală decât valoarea din dreapta, altfel este evaluată ca True.	4 <= 6



# Operatori logici

- Operatorii logici sunt folosiți pentru combinarea rezultatelor obținute în urma condițiilor.

Operator	Nume	Descriere	Exemplu
and	ȘI LOGIC	Este evaluată ca True dacă ambele condiții sunt îndeplinite, altfel este evaluată ca False.	$4 < 5$ and $4 > 2$
or	SAU LOGIC	Este evaluată ca True dacă cel puțin o condiție este adevărată, altfel este evaluată ca False.	$4 < 6$ or $8 < 5$
not	NEGARE	Este evaluată ca True dacă valoarea din stânga este mai mare decât valoarea din dreapta, altfel este evaluată ca False.	not( $4 < 5$ and $4 > 2$ )



# Operatori de identitate

- Operatorii de identitate sunt folosiți pentru compararea obiectelor.
- Nu se compară dacă obiectele sunt identice ci dacă sunt aceleași obiecte, adică dacă ocupă aceeași locație de memorie.

Operator	Descriere	Exemplu
is	Este evaluată ca True dacă ambele valori reprezintă aceeași locație de memorie, altfel este evaluată ca False.	7 is 7
is not	Este evaluată ca True dacă cele două valori reprezintă locații de memorie diferite, altfel este evaluată ca False.	7 is not 9



# Operatori de apartenență

- Operatorii de apartenență sunt folosiți pentru verificarea existenței unei secvențe în cadrul unui obiect.

Operator	Descriere	Exemplu
in	Este evaluată ca True dacă secvența evaluată face parte din obiect, altfel este evaluată ca False.	1 in [1, 2, 3]
not in	Este evaluată ca True dacă secvența evaluată nu face parte din obiect, altfel este evaluată ca False.	'a' not in 'python'



# Operatori pe biți

- Informația într-un calculator este reprezentată în baza 2.
- Asta înseamnă ca orice informație este reprezentată printr-o secvență de biți ce pot avea fie valoarea 0, fie valoarea 1.

Operator	Nume	Descriere
&	AND	Setează fiecare bit cu valoarea 1 dacă ambii biți au valoarea 1, altfel setează valoarea 0.
	OR	Setează fiecare bit cu valoarea 1 dacă cel puțin unul din biți este 1, altfel setează valoarea 0.
^	XOR	Setează fiecare bit cu 1 dacă doar unul din biți este 1, altfel setează valoarea 0.
~	NOT	Schimbă valoarea tuturor biților.
<<	left shift	Adaugă biți cu valoarea 0 la finalul secvenței.
>>	right shift	Elimină biții de la finalul secvenței.





# Variable

Pythonic “Hello, World!”



# Variabile

- O variabilă reprezintă o locație în memoria calculatorului ce poate fi identificată prin nume și conține informații (**valoare**).
- În Python nu există nici o instrucțiune pentru declararea unei variabile. O variabilă este creată în momentul în care aceasta este asignată.
- Alocarea și dealocarea memoriei se face de către Garbage Collector-ul Python. Spre deosebire de alte limbaje de programare, developerii Python nu își mai concentrează timpul și resursele pe această sarcină.
- Avantajul folosirii variabilelor îl reprezintă etichetarea unor valori și posibilitatea de refolosire a acestora.
- Numirea variabilelor reprezintă una din cele mai grele sarcini din repertoriul unui developer. Numele unei variabile trebuie să fie sugestiv atât pentru cine scrie codul cât mai ales pentru ceilalți din echipă. Numele unei variabile este dat pentru noi, developerii, nu pentru calculator.



# Variable - Naming convention

- Numele unei variabile trebuie să fie intuitiv.
- În Python se folosește notarea **snake\_case**.
- Numele acestora trebuie să înceapă cu litera mica.
- Numele variabilelor nu poate începe cu o cifră.
- Numele unei variabile poate să conțină orice caracter dintre a-z, A-Z, 0-9 și semnul \_ (underscore).
- Python este un limbaj case-sensitive - există diferențe între variabila **age**, **Age** și **AGE**.

```
#Legal variable names:
```

```
myvar = "John"  
my_var = "John"  
_my_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "John"
```

```
#Illegal variable names:
```

```
2myvar = "John"  
my-var = "John"  
my var = "John"
```



# Variabile - Asignare

- Pentru declararea unei variabile, în Python, este suficientă asignarea acesteia.
- Denumirea unei variabile se face în același moment în care aceasta se asignează.
- Pentru asignarea unei variabile vom folosi operatorul de asignare =

```
a = 3
```

- În exemplul anterior am asignat valoarea 3 unei variabile pe care am denumit-o a.
- În următorul exemplu vom da variabilei un nume mai intuitiv și vom afișa valoarea returnată de funcția `id()`.

```
child_age = 3  
  
print(id(child_age))
```

- În Python, fiecare obiect este identificat printr-un ID unic. Acest ID este asignat obiectului în momentul în care acesta este creat și reprezintă adresa locației de memorie unde acesta este stocat.
- Acest ID este unic pentru fiecare rulare a programului, exceptând cazurile în care obiectele au asignat un ID unic constant, cum este cazul numerelor întregi de la -5 la 256.



# Variabile - Asignare

- Asignarea unei variabile se poate face simplu, folosind operatorul **=**, ca în exemplul anterior sau prin orice combinație între operatorii aritmetici sau pe biți studiați anterior precedați de operatorul **=**.

```
# Asignare simpla: my_var va avea valoarea 5
my_var = 5

# Asignare prin adaugare: my_var va avea valoarea 8 (5 + 3)
my_var += 3 # echivalent cu my_var = my_var + 3

# Asignare prin inmultire: my_var va avea valoarea 16 (8 * 2)
my_var *= 2 # echivalent cu my_var = my_var * 2

# Asignare prin shiftare catre dreapta: my_var va avea valoarea 1 (16 >> 4 = 2)
# 0b10000 [16] >> 4 = 0b1 [1]
my_var >>= 4 # echivalent cu my_var = my_var >> 4

# Asignare prin shiftare catre stanga: my_var va avea valoarea 8 (1 << 3 = 8)
# 0b1 [1] << 3 = 0b1000 [8]
my_var <<= 3 # echivalent cu my_var = my_var << 3
```



# Șiruri de caractere

Pythonic "Hello, World!"



# Șiruri de caractere

- Dacă în capitolele anterioare am discutat despre numere, acum a venit rândul caracterelor.
- Un șir de caractere reprezintă un alt tip de date alcătuit din unul sau mai multe caractere.
- Acest tip de date o să îl întâlniți sub denumirea de **string** (pl. *strings*).
- Un string este definit între ‘ ‘ (ghilimele simple) sau “ ” (ghilimele duble).
- Am folosit un string chiar în programul dezvoltat la începutul cursului:

```
print("Hello, World!")
```

- În instrucțiunea de mai sus am folosit funcția **print()** pentru a afișa mesajul “Hello, World!” (string-ul / șirul de caractere).
- Așa cum am folosit variabile cărora le-am asignat ca valori niște numere, la fel putem asigna și șiruri de caractere:

```
my_message = "Ana are mere."  
print(my_message)
```

- În Python nu există noțiunea de caracter, un caracter poate fi reprezentat ca fiind un șir de caractere format dintr-un singur caracter.

```
my_char = "a"
```

- La fel de ușor poate fi definit și un șir de caractere gol:

```
my_empty_string = ""
```



# Șiruri de caractere - ASCII

- Un șir de caractere poate conține orice caracter din codul ASCII.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32; Space	64	40	100	&#64; @	96	60	140	&#96; `		
1	1	001	SOH	(start of heading)	33	21	041	&#33; !	65	41	101	&#65; A	97	61	141	&#97; a		
2	2	002	STX	(start of text)	34	22	042	&#34; "	66	42	102	&#66; B	98	62	142	&#98; b		
3	3	003	ETX	(end of text)	35	23	043	&#35; #	67	43	103	&#67; C	99	63	143	&#99; c		
4	4	004	EOT	(end of transmission)	36	24	044	&#36; \$	68	44	104	&#68; D	100	64	144	&#100; d		
5	5	005	ENQ	(enquiry)	37	25	045	&#37; %	69	45	105	&#69; E	101	65	145	&#101; e		
6	6	006	ACK	(acknowledge)	38	26	046	&#38; &	70	46	106	&#70; F	102	66	146	&#102; f		
7	7	007	BEL	(bell)	39	27	047	&#39; '	71	47	107	&#71; G	103	67	147	&#103; g		
8	8	010	BS	(backspace)	40	28	050	&#40; (	72	48	110	&#72; H	104	68	150	&#104; h		
9	9	011	TAB	(horizontal tab)	41	29	051	&#41; )	73	49	111	&#73; I	105	69	151	&#105; i		
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42; *	74	4A	112	&#74; J	106	6A	152	&#106; j		
11	B	013	VT	(vertical tab)	43	2B	053	&#43; +	75	4B	113	&#75; K	107	6B	153	&#107; k		
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44; ,	76	4C	114	&#76; L	108	6C	154	&#108; l		
13	D	015	CR	(carriage return)	45	2D	055	&#45; -	77	4D	115	&#77; M	109	6D	155	&#109; m		
14	E	016	SO	(shift out)	46	2E	056	&#46; .	78	4E	116	&#78; N	110	6E	156	&#110; n		
15	F	017	SI	(shift in)	47	2F	057	&#47; /	79	4F	117	&#79; O	111	6F	157	&#111; o		
16	10	020	DLE	(data link escape)	48	30	060	&#48; 0	80	50	120	&#80; P	112	70	160	&#112; p		
17	11	021	DC1	(device control 1)	49	31	061	&#49; 1	81	51	121	&#81; Q	113	71	161	&#113; q		
18	12	022	DC2	(device control 2)	50	32	062	&#50; 2	82	52	122	&#82; R	114	72	162	&#114; r		
19	13	023	DC3	(device control 3)	51	33	063	&#51; 3	83	53	123	&#83; S	115	73	163	&#115; s		
20	14	024	DC4	(device control 4)	52	34	064	&#52; 4	84	54	124	&#84; T	116	74	164	&#116; t		
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53; 5	85	55	125	&#85; U	117	75	165	&#117; u		
22	16	026	SYN	(synchronous idle)	54	36	066	&#54; 6	86	56	126	&#86; V	118	76	166	&#118; v		
23	17	027	ETB	(end of trans. block)	55	37	067	&#55; 7	87	57	127	&#87; W	119	77	167	&#119; w		
24	18	030	CAN	(cancel)	56	38	070	&#56; 8	88	58	130	&#88; X	120	78	170	&#120; x		
25	19	031	EM	(end of medium)	57	39	071	&#57; 9	89	59	131	&#89; Y	121	79	171	&#121; y		
26	1A	032	SUB	(substitute)	58	3A	072	&#58; :	90	5A	132	&#90; Z	122	7A	172	&#122; z		
27	1B	033	ESC	(escape)	59	3B	073	&#59; ;	91	5B	133	&#91; [	123	7B	173	&#123; {		
28	1C	034	FS	(file separator)	60	3C	074	&#60; <	92	5C	134	&#92; \	124	7C	174	&#124;		
29	1D	035	GS	(group separator)	61	3D	075	&#61; =	93	5D	135	&#93; ]	125	7D	175	&#125; }		
30	1E	036	RS	(record separator)	62	3E	076	&#62; >	94	5E	136	&#94; ^	126	7E	176	&#126; ~		
31	1F	037	US	(unit separator)	63	3F	077	&#63; ?	95	5F	137	&#95; _	127	7F	177	&#127; DEL		

- Orice caracter din codul ASCII poate fi reprezentat printr-un număr întreg.
- Pentru a obține codul ASCII al unui caracter putem folosi funcția **ord()**.

```
print(ord('a'))
```

- Pentru a obține caracterul reprezentat de un cod ASCII putem folosi funcția **chr()**. 

```
print(chr(97))
```





# Șiruri de caractere - escaping

- Există anumite caractere și/sau combinații de caractere care au un rol special. Cel mai simplu exemplu sunt ghilimelele - acestea (simple sau duble) sunt folosite pentru definirea unui șir de caractere.
- Pentru folosirea caracterelor speciale cu rol de caractere integrate în string-ul nostru vom folosi caracterul \ (backslash) pentru a face escape.

```
print("Si Dumnezeu a zis: \"Sa se faca lumina!\")
```

- În exemplu anterior am folosit caracterul \ înaintea caracterului “ pentru a îl integra în șirul nostru de caractere, astfel încât mesajul final să fie: ***Si Dumnezeu a zis: “Sa se faca lumina!”***.
- O listă cu caracterele speciale o găsiți în imaginea următoare:

Escape Sequence	Meaning
<code>\newline</code>	Ignored
<code>\\</code>	Backslash (\)
<code>\'</code>	Single quote (')
<code>\"</code>	Double quote (")
<code>\a</code>	ASCII Bell (BEL)
<code>\b</code>	ASCII Backspace (BS)
<code>\f</code>	ASCII Formfeed (FF)
<code>\n</code>	ASCII Linefeed (LF)
<code>\r</code>	ASCII Carriage Return (CR)
<code>\t</code>	ASCII Horizontal Tab (TAB)
<code>\v</code>	ASCII Vertical Tab (VT)
<code>\ooo</code>	ASCII character with octal value <i>ooo</i>
<code>\xhh...</code>	ASCII character with hex value <i>hh...</i>



# Șiruri de caractere - multiline string

- Pentru scrierea pe mai multe linii există două variante:

- folosirea caracterului ***newline***: \n

```
print("Somnoroase pasarele \nPe la cuiburi se aduna,\nSe ascund in ramurile-\nNoapte buna!")
```

- folosirea ghilimelelor triple (simple sau duble):

```
print("""Somnoroase pasarele  
Pe la cuiburi se aduna  
Se ascund in ramurile -  
Noapte buna!""")
```

- Cele două exemple de mai sus sunt echivalente, ambele vor afișa mesajul:

```
Somnoroase pasarele  
Pe la cuiburi se aduna  
Se ascund in ramurile -  
Noapte buna!
```



# Șiruri de caractere - raw string

- Pentru a evita rolul caracterelor speciale putem folosi noțiunea de **raw string**.

```
print('Somnoroase pasarele\nPe la cuiburi se aduna.')
```

- În exemplu anterior mesajul nostru va fi afișat pe două linii datorită caracterului newline \n.
- Folosind raw string putem afișa mesajul așa cum arată, fără a fi nevoiți să facem escape caracterului \n.
- Pentru definirea unui raw string vom prefixa string-ul nostru cu caracterul **r** sau **R**.

```
print(r'Somnoroase pasarele\nPe la cuiburi se aduna.')
```

- În acest exemplu mesajul va fi afișat exact așa cum apare între ghilimele. Această tehnică ne ajută când vrem să afișăm un text așa cum este, fără a îl formata.



# Șiruri de caractere - formatting

- Un șir de caractere poate fi formatat prin mai multe metode. Prin formatarea unui string înțelegem înlocuirea unor placeholderuri din cadrul string-ului cu date:
  - metoda **format()** - definirea placeholderului se face între acolade { }

```
print("For only {price:.2f} dollars! {cheers_msg}".format(price=49, cheers_msg="Have a great day!"))
```

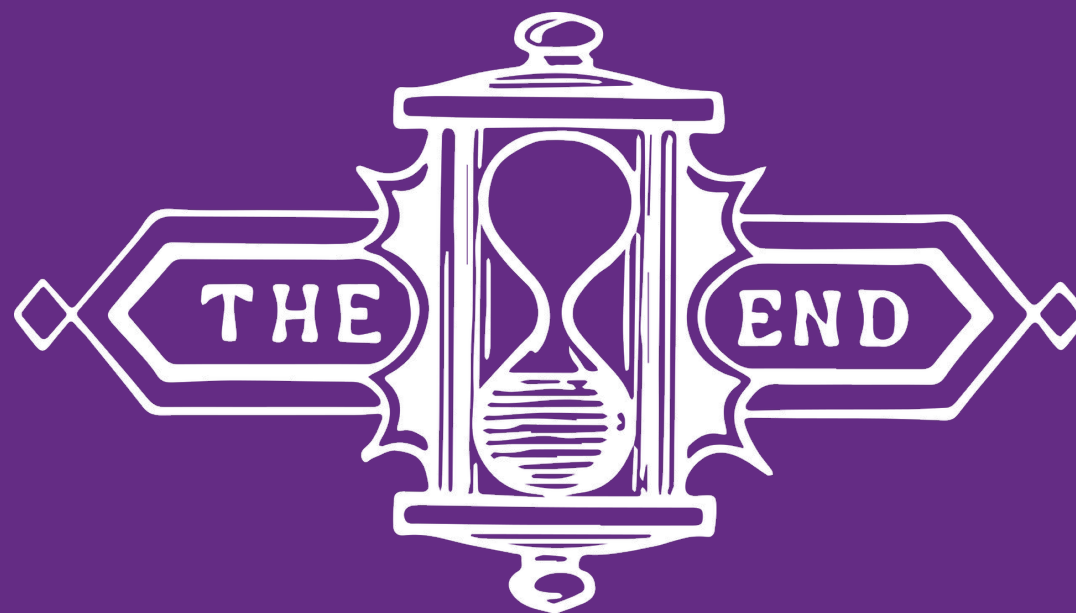
- în exemplul anterior mesajul afișat va fi: ***For only 49 dollars! Have a great day!***
- placeholderurile pot fi identificate folosit nume (cum este și cazul de mai sus {cheers\_msg}), placeholderuri numerotate sau chiar placeholderuri goale:

```
print("For only {0:.2f} dollars! {1}".format(49, "Have a great day!"))  
print("For only {:.2f} dollars! {}".format(49, "Have a great day!"))
```

- toate exemplele de mai sus vor produce același rezultat.
- în cazul ultimelor două cazuri trebuie respectată ordinea parametrilor primiți de funcția **format()**.
- folosind F-strings - acestea au fost introduse pentru a face mai ușoară formatarea string-urilor:
  - nu mai este nevoie de definirea placeholderurilor - formatarea se face pe loc
  - sunt string-uri care încep cu litera f sau F, iar valorile ce se doresc interpolate sunt folosite între acolade { }.

```
print(F'For only {49:.2f} dollars! {"Have a great day!"}')
```





Vă mulțumesc!

