

Elemente de bază ale limbajului Python



Diferențe față de C/C++

- Variabilele în Python nu au tip de date static (nu se declară tipul lor, o variabilă este “declarantă” **când i se atribuie prima dată o valoare**).
- Tipul unei variabile se poate schimba pe parcursul execuției programului .

```
x = 7 #variabila x este "declarata"  
print(x)  
x = "abc"  
print(x)
```

Diferențe față de C/C++

- ▶ Nu sunt necesari delimitatori de blocuri de tip `{}` sau `begin/end` etc, este obligatorie indentarea blocurilor de cod (și suficientă pentru delimitarea acestora)
- ▶ Nu este nevoie să punem `;` la finalul unei linii (dacă nu mai urmează alte linii de cod pe aceeași linie)

```
i = 1
while i<10:
    print(i)
    i = i + 1    #nu i++
print("gata afisarea")
```

Elemente de bază ale limbajului

1. Afișarea unei variabile + tipul acesteia (al valorii asignate)

Elemente de bază ale limbajului

```
print("mesaj")
x = 1
print("x=", x, type(x), id(x)) #pe acelasi rand cu spatiu, apoi linie noua
print("x=" + str(x))
x = "Sir"
print("x=", x, type(x), id(x)) #nu are acelasi id
y = 2
print(x, end=' ') #pentru a nu trece la linie noua modific parametrul end
print(y)
print(x, y, sep='*')
```

Elemente de bază ale limbajului

2. Citirea de la tastatură + funcții de conversie

▶ funcția `input`

- parametru (opțional) mesajul care se va afișa pe ecran
- returnează **șirul de caractere** introdus până la sfârșitul de linie (de tip `str`, **este necesară conversia** dacă vrem alt tip de date)

Elemente de bază ale limbajului

```
#citire-necesara conversie
```

```
x = input("x=")
```

```
print("x=", x, type(x))
```

```
x = int(input("intreg=")) #ValueError daca introducem gresit
```

```
print("x=", x, type(x))
```

```
x = float(input("real="))
```

```
print("x=", x, type(x))
```

```
x = complex(input("complex="))
```

```
print("x=", x, type(x))
```

Elemente de bază ale limbajului

3. Erori

```
i = 1  #i="ab", i=-1
print(i)
if i>0: #daca i nu este numar?
    print("ok")
else:
    print(i + " este negativ") #daca i nu este sir?
    print(y) #da eroare daca i=1?
```


Variabile

Variabile

- În C/C++ o variabilă se declară și are: tip, adresa, valoare
- În Python **variabilele** sunt referințe spre obiecte (**nume** date obiectelor); orice **valoare** este un **obiect**
- Un obiect **ob** are asociat:
 - un număr de identificare: **id(ob)**
 - un tip de date: **type(ob)**
 - o valoare – poate fi convertită la șir de caractere **str(ob)**

Variabila; obiect; alocare; atribuire

C

Python

`m = 1000`

`m:` 1000

`m` → 1000

Variabila; obiect; alocare; atribuire

C

Python

```
m = 1000
```

m: 1000

m → 1000

```
m = m+1
```

Variabila; obiect; alocare; atribuire

C

```
m = 1000
```

m:

1001

```
m = m+1
```

Python

m

1000

1001



Variabila; obiect; alocare; atribuire

C

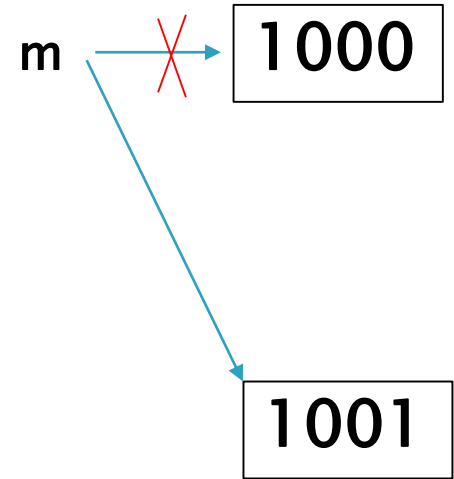
```
m = 1000
```

m: 1001

```
m = m+1
```

```
n = m
```

Python



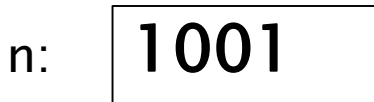
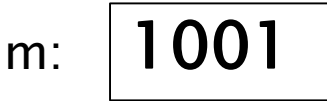
Variabila; obiect; alocare; atribuire

C

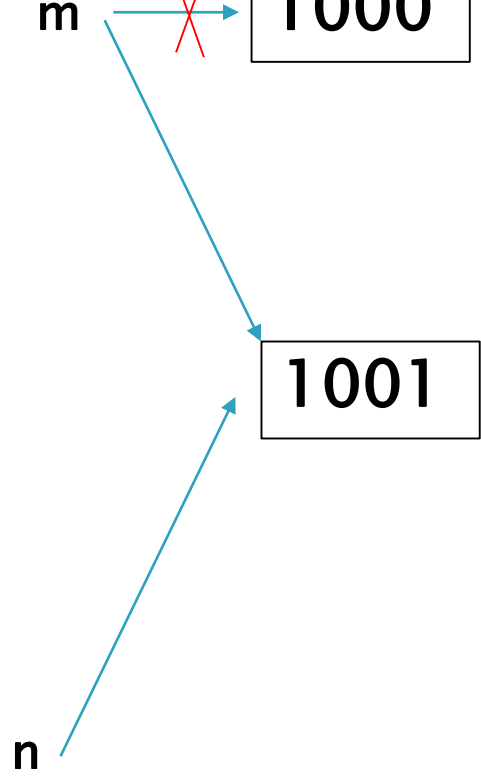
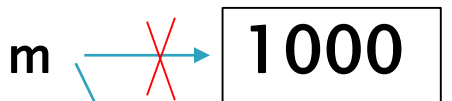
```
m = 1000
```

```
m = m+1
```

```
n = m
```



Python



Variabila; obiect; alocare; atribuire

C

```
m = 1000
```

m: 1001

```
m = m+1
```

```
n = m
```

n: 1001

```
n = n+1 ???
```

Python

m  1000

1001

n 

Variabile

- Tipul unei variabile se stabilește prin inițializare și se poate schimba prin atribuiri de valori de alt tip
- Numele unei variabile – identificatori
- Recomandare nume:

`litere_mici_separate_prin_underscore`



Variabile

- optimizare: numerele întregi din intervalul $[-5, 256]$ sunt prealocate (în cache) – **toate obiectele care au o astfel de valoare sunt identice (au același id)**
- variabile cu aceeași valoare **pot avea** același id (dacă este o valoare prealocată, atunci sigur da)

Variabile

▶ Exemplu

```
x = 1
```

```
y = 0
```

```
y = y + 1
```

```
z = x
```

```
print(x,y,z,x*x)
```

```
print(id(x),id(y),id(z),id(x*x))
```

```
#toate expresiile cu valoare 1 au acelasi id
```

Variabile

▶ Exemple

```
x = 1000
```

```
y = 999
```

```
y = y+1
```

```
z = x
```

```
print(x,y,z,10*x//10)
```

```
print(id(x),id(y),id(z),id(10*x//10))
```

Variabile

- `del x` – șterge o variabilă din memorie

```
m = input()  
del m  
print(m) #eroare
```

- Garbage collector – șterge obiecte către care nu mai sunt referințe

Tipuri de date

Tipuri de date

- **int**
 - numere întregi cu oricât de multe cifre (limita dată doar de performanța sistemului pe care se rulează)

Tipuri de date

- **int**

- numere întregi cu oricât de multe cifre (limita dată doar de performanța sistemului pe care se rulează)
- memorate ca vectori de “cifre” din reprezentarea în baza 2^{30} (cu cifre de la 0 la $2^{30}-1 = 1073741823$)

Exemplu: Reprezentarea pentru 234254646549834273498:

ob_size	3		
ob_digit	462328538	197050268	203

deoarece $234254646549834273498 = 462328538 \times (2^{30})^0 + 197050268 \times (2^{30})^1 + 203 \times (2^{30})^2$

Tipuri de date

- **int**

- constante în baza 10 (implicit), dar și în bazele 2 (prefix 0b,0B), 8 (prefix 0o, 0O), 16 (prefix 0x,0X):

```
print(0b101, 0o10, 0xAB)
```

Tipuri de date

- **int**

- constante în baza 10 (implicit), dar și în bazele 2 (prefix 0b,0B), 8 (prefix 0o, 0O), 16 (prefix 0x,0X):

```
print(0b101, 0o10, 0xAB)
```

- putem folosi `int(sir)` pentru creare/conversie (există și varianta `int(sir, base=baza)`)

```
print(int(9.7) + int("101", base = 2) + int("101",2))
```

Tipuri de date

- **float**

- Constante: 3.5, 1e-2 (notație științifică)
- float([x]):

`float("inf"); float("infinity"); float("nan")`

Tipuri de date

- **float**
 - IEEE-754 double precision
 - operațiile aritmetice cu tipul de date *float* nu au precizie absolută:

NU: $0.1 * 0.1 == 0.01$

DA: $\text{abs}(0.1 * 0.1 - 0.01) < 1e-9$

Tipuri de date

- **complex**
 - de forma $a + bj$ (!!! nu i, merge și J)

Tipuri de date

- `complex`

```
z = complex(-1, 4)

print("Numarul complex:", z)

print("Partea reala:", z.real)

print("Partea imaginara:", z.imag)

print("Conjugatul:", z.conjugate())

print("Modul:", abs(z))
```

Tipuri de date

- **bool**
 - True, False
 - putem folosi `bool()` pentru conversie
 - În context boolean – **conversia oricărei valori la bool**

Context boolean – condiție if, while; operand pentru operatori logici – conversii

Tipuri de date

- **bool**
 - Se consideră **False**:
 - **None, False**
 - **0, 0.0, 0j, Decimal(0), Fraction(0,1)**
 - **Colecții și secvențe vide** (+obiecte în care `__bool__()` returnează False sau `__len__()` returnează 0)

```
print(bool(0), bool(-5))
```

```
print(bool(""), bool(" "))
```

```
print(bool(None), bool([]))
```


Tipuri de date

- NoneType

- **None**

- Nu exista char

`ord("a")`

`chr(97)`

Tipuri de date

Secvențe:

Mutable (le putem modifica) și imutable

- liste `list`: `a = [3, 1, 4, 7]` – mutable
- tupleuri `tuple`: `a = (3, 1, 4, 7)`
- șiruri de caractere `str`: `a = "3147sir"`

Tipuri de date

Mulțimi:

- set: $a = \{1, 4, 5\}$
- frozenset: $fa = \text{frozenset}(a)$ – nu se poate modifica

Dicționare

Operatori

Operatori

- aritate (număr de operanzi)
- prioritate (precedența) $1 + 2 * 3$
- asociativitatea: $x \text{ op } y \text{ op } z$
 - de la stânga la dreapta sau de la dreapta la stânga
 - stabilește ordinea în care se fac operațiile într-o expresie în care un operator se repetă

$$1 + 2 + 3 = (1 + 2) + 3$$

$$10 ** 2 ** 7 = 10 ** (2 ** 7)$$

Operatori

- Operatori aritmetici

+	adunare
-	scădere
*	înmulțire
/	Împărțire exactă, rezultat float (nu ca în C/C++ sau Python 2)
//	împărțire cu rotunjire la cel mai apropiat întreg mai mic sau egal decât rezultatul împărțirii exacte dacă un operator este float rezultatul este de tip float
%	restul împărțirii
**	ridicare la putere

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi **pentru tipurile pentru care au sens** (de exemplu și pentru numere complexe)

$3 + 2.0$

$2j * 3j$

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

$$\begin{array}{ccc} 3 + 2.0 & \longrightarrow & 5.0 \\ 2j * 3j & & (-6+0j) \end{array}$$

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

1/1

1/2

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

1/1	→	1.0
1/2		0.5

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

4//2

5//2.5

2.5//1.5

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

4//2

2

5//2.5



2.0

2.5//1.5

1.0

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

11//3

11//-3

-11//3

-11//-3

Operatori

- Tipul rezultatului – similar C/C++, diferit la / și //
- se pot folosi pentru tipurile pentru care au sens (de exemplu și pentru numere complexe)

11//3

3

11//-3



-4

rotunjire inferioară

-11//3

-4

-11//-3

3

Operatori

- $\mathbf{x \% y = x - ((x // y) * y)}$

`11%3`

`11%-3`

`-11%3`

`-11 %-3`

`10.5%2`

`3%1.5`

Operatori

- $\mathbf{x \% y = x - ((x // y) * y)}$

$$11 \% 3 \quad \longrightarrow \quad 2$$

Operatori

- $\mathbf{x \% y = x - ((x // y) * y)}$

$$11 \% 3 \qquad \qquad \qquad 2$$

$$11 \% -3 \qquad \longrightarrow \qquad -1$$

$$11 \% -3 = 11 - ((-3) * (11 // -3))$$

$$= 11 - (-3) * (-4) = 11 - 12 = -1$$

Operatori

- Operatori relaționali

$x == y$	x este egal cu y
$x != y$	x nu este egal cu y
$x > y$	x mai mare decât y
$x < y$	x mai mic decât y
$x \geq y$	x mai mare sau egal y
$x \leq y$	x mai mic sau egal y

Operatori

- Operatori relaționali

- Se pot înlănțui: $1 < x < 10$
- operatorul `is` – testeaza dacă obiectele sunt identice (au *acelasi id, nu doar aceeași valoare*)

<pre>x = 1000 y = 999 y = y+1 print(x == y) print(x is y)</pre>	<pre>x = 1 y = 0 y = y+1 print(x == y) print(x is y)</pre>
---	--

Operatori

- Operatori de atribuire

=

+=, -=, *=, /=, **=, //=, %=,

&=, |=, ^=, >>=, <<= (v. operatori pe biți)

În Python nu există operatorii ++ și --

Operatori

- Operatori de atribuire

Din versiunea 3.8 a fost introdus și operatorul de atribuire în expresii (operatorul walrus) :=

```
#print(x=1) NU
```

```
print(x := 1)
```

```
y = (x:=2) + 5*x
```

```
print(x,y)
```

```
if x:=y :
```

```
    print(x,y)
```

```
while (x:=int(input()))>0: #trebuie ()
```

```
    print(x)
```

Operatori

- Operatori logici

`not, and, or`

– se evaluează prin scurtcircuitare

```
x = True
```

```
print( x or y ) #nu da eroare
```

```
print( x and y ) #NameError
```

Operatori

- Operatori logici

`not`, `and`, `or`

- se evaluează prin scurcircuitare
 - în **context Boolean** orice valoare se poate evalua ca `True/False`
- => operatorii logici nu se aplica doar pe valori de tip `bool` (ci pentru orice valori), iar rezultatul nu este neapărat de tip `bool` (decât în cazul operatorului `not`)

Operatori

- Operatori logici

$$x \text{ and } y = \begin{cases} y, & \text{dacă } x \text{ se evaluează ca } True \\ x, & \text{altfel} \end{cases}$$

$$x \text{ or } y = \begin{cases} x, & \text{dacă } x \text{ se evaluează ca } True \\ y, & \text{altfel} \end{cases}$$

$$\text{not } x = \begin{cases} False, & \text{dacă } x \text{ se evaluează ca } True \\ True, & \text{altfel} \end{cases}$$

`"a" and True ==> True`

`"a" or True ==> 'a'`

Operatori

```
x = 0
```

```
y = 4
```

```
if x:
```

```
    print(x)
```

```
print(x and y)
```

```
print(x or y)
```

```
print(not x, not y)
```

```
print((x<y) and y)
```

```
print((x<y) or y)
```



Operatori

- **Operatori pe biți**
 - pentru numere întregi
 - rapizi, asupra reprezentării interne

$\sim x$	complement față de 1
$x \& y$	și pe biți
$x y$	sau pe biți
$x \wedge y$	sau exclusiv pe biți
$x \gg k$	deplasare la dreapta cu k biți
$x \ll k$	deplasare la stânga cu k biți

\sim	0	1
	1	0

$\&$	0	1
0	0	0
1	0	1


$ $	0	1
0	0	1
1	1	1

\wedge	0	1
0	0	1
1	1	0

Operatori

$$11 \ \& \ 86 = ?$$


11:	0	0	0	0	1	0	1	1
86:	0	1	0	1	0	1	1	0
<hr/>								
11 & 86:	0	0	0	0	0	0	1	0



Operatori

$$11 \ \& \ 86 = ?$$

11:	0	0	0	0	1	0	1	1
86:	0	1	0	1	0	1	1	0
11 & 86:	0	0	0	0	0	0	1	0



2

Operatori

Observații:

$$\begin{aligned}x = x \gg k &\quad \Leftrightarrow \quad x = \text{parte întregă inferioară din} \\&\quad x / 2^k \quad (x \text{ div } 2^k) \\&\quad = x // (2^{**}k)\end{aligned}$$

$$x = x \ll k \quad \Leftrightarrow \quad x = x * (2^{**}k)$$

$$x = 11 = 0b00001011$$

$$x \ll 3 = 0b01011000 = 88 = 11 * (2^{**}3)$$

$$x = 11 = 0b00001011$$

$$x \gg 3 = 0b00000001 = 1 = 11 // (2^{**}3)$$

Operatori

Aplicație 1 operatori biți: Test paritate

#ultimul bit din reprezentarea binara este 0 sau 1

```
x = int(input())
```

```
if x&1 == 0:
```

```
    print("par")
```

```
else:
```

```
    print("impar")
```

$$x = a_1 \dots a_{n-1} a_n_{(2)}$$

$$1 = 0 \dots 0 1_{(2)}$$

$$x \& 1 = 0 \dots 0 a_n_{(2)}$$

Operatori

Aplicație 2 operatori biți: Interschimbare

```
x = 12; y=14
```

```
x = x ^ y
```

```
y = x ^ y # x ^ y ^ y = x
```

```
x = x ^ y # x ^ y ^ x = y
```

```
print(x,y)
```



Operatori

Aplicație 3 operatori biți: Test putere a lui 2

```
print( x & (x-1) == 0)
```

$$x = a_1 \dots a_{k-1} 1 0 0 \dots 0_{(2)}$$

$$x - 1 = a_1 \dots a_{k-1} 0 1 1 \dots 1_{(2)}$$

$$x \& (x - 1) = a_1 \dots a_{k-1} 0 0 0 \dots 0_{(2)}$$

Operatori

Temă

```
x = 272
```

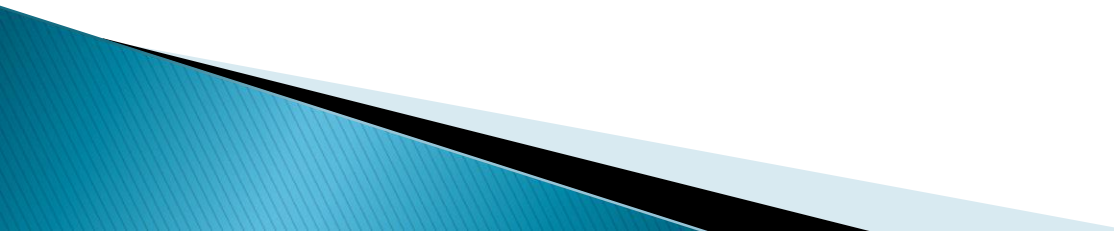
```
print(bin(x))
```

```
print(bin(x & 0b10001), x & 0b10001)
```

```
print(bin(17 | 0b10001), 17 | 0b10001)
```

```
print(bin(~x), ~x)
```

```
print(bin(x>>1), x>>1)
```



Operatori

- Operatorul condițional (ternar)

expresie_1 **if** *conditie* **else** *expresie_2*

Operatori

- Operatorul condițional (ternar)

expresie_1 **if** *conditie* **else** *expresie_2*

x = 5; y = 20

z = x-y if x > y else y-x

Operatori

- Operatorul condițional (ternar)

expresie_1 **if** *conditie* **else** *expresie_2*

```
x = 5; y = 20
```

```
z = x-y if x > y else y-x
```

```
print("Modulul diferentei", z)
```

```
z = x if x > y else ""
```

– evaluat tot prin scurcircuitare

Operatori

- **Operatori de identitate:** `is`, `is not`
- **Operatori de apartenență :** `in`, `not in` (la o colecție)

```
s = "aeiou"  
x = "e"  
print("vocala" if x in s else "consoana")
```

```
ls = [0, 2, 10, 5]  
print(3 not in ls)
```

Operatori

- Precedența operatorilor:

<https://docs.python.org/3/reference/expressions.html#operator-precedence>

1 + 1 << 2

2 * 3 ** 4

Operatori

- Precedența operatorilor:

<https://docs.python.org/3/reference/expressions.html#operator-precedence>

1 + 1 << 2

2 * 3 ** 4



(1 + 1) << 2

2 * (3 ** 4)

Comentarii

- Prefixat de # => comentariu pe o linie
- Pentru mai multe linii – # pe fiecare linie sau delimitatori de șiruri de caractere
 - Încadrat de ' ' ' => pe mai multe linii
 - Încadrat de " " " => docstring – comentariu pe mai multe linii, folosit în mod special pentru documentare

