

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

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)

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

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

-11//3

-4

-11//-3

3



rotunjire **inferioară**

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
$x >= y$	x mai mare sau egal y
$x <= y$	x mai mic sau egal y

Operatori

- Operatori relaționali

- Se pot înlănțui: $5 \leq x < 10$

Operatori

- Operatori relaționali

- Se pot înlănțui: `5 <= 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**

Instrucțiunea de atribuire, de exemplu `x = 2` este o **instrucțiune**, nu o **expresie** care se evaluează la o valoare, ca în C. Astfel:

NU corect `if x=2:`

NU corect `y = (x=2) + 5*x`

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 # y = 5*x + (x:=2)
print(x,y)

if x:=y:
    print(x,y)
```

Operatori

- **Operatori de atribuire**

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

```
while (x:=int(input()))>0:  
    print(x)
```

Operatori

- Operatori logici

`not, and, or`

– se evaluează prin scurtcircuitare

```
x = True  
print( x or y ) #?  
print( x and y ) #?
```

Operatori

- Operatori logici

`not, and, or`

– se evaluează prin scurtcircuitare

```
x = True  
print( x or y) #True, nu da eroare  
print( x and y) #NameError
```


Operatori

- Operatori logici

`not, and, or`

- se evaluează prin scurtcircuitare

- 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 => ??

"a" or True => ??

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

Reprezentarea numerelor întregi în calculator

- ▶ naturale – baza 2 $x = \overline{a_1 \dots a_n}_{(2)} \Rightarrow 00\dots 0a_1\dots a_n$

`repr(11) = 00001011`

Reprezentarea numerelor întregi în calculator

- ▶ naturale – baza 2 $x = \overline{a_1 \dots a_n}_{(2)} \Rightarrow 00\dots 0a_1\dots a_n$
- ▶ negative – complement față de 2, un bit pentru semn

Reprezentarea numerelor întregi în calculator

- ▶ naturale – baza 2 $x = \overline{a_1 \dots a_n}_{(2)} \Rightarrow 00\dots 0a_1\dots a_n$
- ▶ negative – complement față de 2, un bit pentru semn

$$x + (-x) = 0$$

$$\text{repr}(x) + \text{repr}(-x) = 2^n = \underbrace{10 \dots 0}_{n \text{ biți}}_{(2)}$$

$$\text{not}(\text{repr}(x)) = \text{complementul lui } \text{repr}(x)$$

$$\text{repr}(x) + \text{not}(\text{repr}(x)) = 1\dots 1_{(2)} = 2^n - 1$$

$$\begin{aligned} \text{Avem atunci } \text{repr}(-x) &= \text{not}(\text{repr}(x)) + 1 = \\ &= \text{not}(\text{repr}(x-1)) \end{aligned}$$

Reprezentarea numerelor întregi

- ▶ $\text{repr}(-x) = \text{not repr}(x) + 1$
- ▶ **Exemplu:** pentru $n=8$ biți reprezentarea lui -11 se obține astfel:

$$\text{repr}(11) = 00001011$$


$$\text{not}(\text{repr}(11)) = 11110100$$

$$\text{not}(\text{repr}(11)) + 1 = 11110101 = \text{repr}(-11)$$

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 pe biți

$\sim 11 = ?$

$\sim 0 = ?$

11:

$y = \sim 11$:

0	0	0	0	1	0	1	1
1	1	1	1	0	1	0	0

Negativ; aflăm $|y|$



Operatori pe biți

$\sim 11 = ?$

$\sim 0 = ?$

11:

$y = \sim 11$:

Scădem 1

Trecem la complement

0	0	0	0	1	0	1	1
1	1	1	1	0	1	0	0

Negativ; aflăm $|y|$



Operatori pe biți

$\sim 11 = ?$

$\sim 0 = ?$

11:

$y = \sim 11$:

Scădem 1

Trecem la complement

0	0	0	0	1	0	1	1
1	1	1	1	0	1	0	0
1	1	1	1	0	0	1	1

Negativ; aflăm $|y|$



Operatori pe biți

$$\sim 11 = ?$$

$$\sim 0 = ?$$

11:

$y = \sim 11$:

Scădem 1

Trecem la complement

0	0	0	0	1	0	1	1
1	1	1	1	0	1	0	0
1	1	1	1	0	0	1	1
0	0	0	0	1	1	0	0

Negativ; aflăm $|y|$

$|y| = 12$

$$\sim 11 = -12$$

(avem $\text{repr}(-x) = \text{not}(\text{repr}(x-1)) \Rightarrow \text{repr}(-12) = \text{not}(\text{repr}(11))$)

Operatori

Observații:

$x = x \gg k \quad \Leftrightarrow \quad x = \text{parte \u00e2ntreag\u0103 inferioar\u0103 din}$
 $\quad \quad \quad x / 2^k \text{ (} x \text{ div } 2^k \text{)}$
 $\quad \quad \quad = x // (2^{**}k)$

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

Operatori

Observații:

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

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

$x = 11 = 0b00001011$

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

$x = 11 = 0b00001011$

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

Operatori

Observații – Operatorul xor:

- asociativ, comutativ
- $a \wedge 0 = a$ pentru $a \in \{0, 1\}$
- $a \wedge 1 = \sim a$ pentru $a \in \{0, 1\}$
- $a \wedge b = 1 \Leftrightarrow a \neq b$ pentru $a, b \in \{0, 1\}$
- $x \wedge x = 0$

Operatori

Aplicație 1 operatori biți: Interschimbare

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

```
x = x ^ y
```

```
y = x ^ y  #  $y = (x \wedge y) \wedge y = x \wedge (y \wedge y) = x \wedge 0 = x$ 
```

```
x = x ^ y  #  $x = (x \wedge y) \wedge x = x \wedge (y \wedge x) = (x \wedge x) \wedge y = 0 \wedge y = y$ 
```

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

Operatori

Aplicație 2 operatori biți: Test paritate

```
#testam daca ultimul bit din reprezentare este 0 sau 1
x = int(input())
if x&1 == 0:
    print("par")
else:
    print("impar")
```

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

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

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

Operatori

Aplicație 3 operatori biți: Test x este putere a lui 2

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

$$x = a_1 \dots a_k 1 0 0 \dots 0_{(2)}$$

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

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

Operatori

Temă

```
x = 272
```

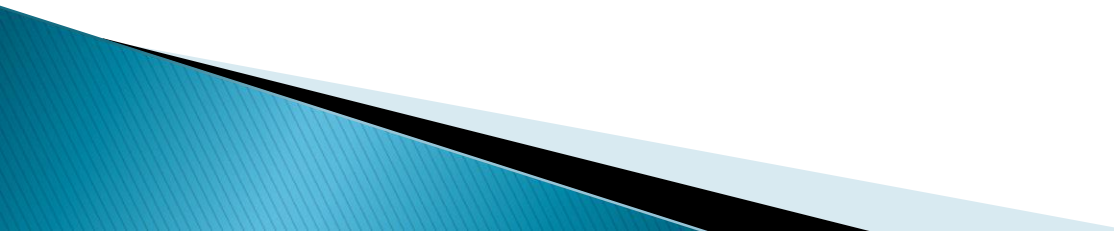
```
print(bin(x))
```

```
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 scurtcircuitare

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	→	(1 + 1) << 2
2 * 3 ** 4		2 * (3 ** 4)

Operatori

- Precedența operatorilor:

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

```
grupa // 10 == 14 or grupa//10 == 10 and media >= 9
```



Este un filtru corect pentru a obține studenții din seriile 14 și 10 care au media cel puțin 9?

