

Elemente de bază ale limbajului Python



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
$x >= y$	x mai mare sau egal y
$x <= y$	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*)

```
x = 1000
```

```
y = 999
```

```
y = y+1
```

```
print(x == y)
```

```
print(x is y)
```

```
x = 1
```

```
y = 0
```

```
y = y+1
```

```
print(x == y)
```

```
print(x is y)
```

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

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

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

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 scurcircuitare

```
x = True
```

```
print( x or y) #?
```

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

Operatori

- Operatori logici

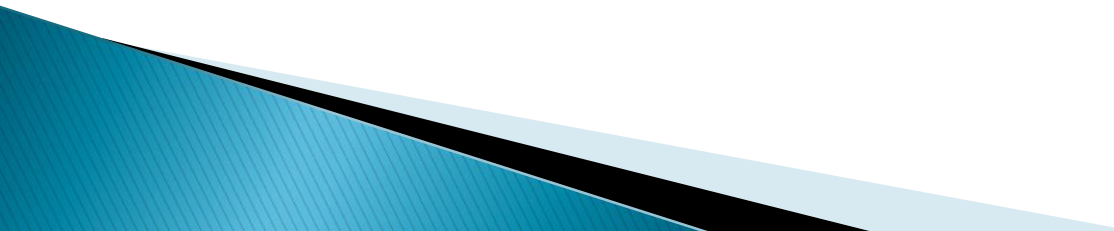
`not, and, or`

– se evaluează prin scurcircuitare

```
x = True
```

```
print( x or y) #True, 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 => ??

"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

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

```
#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 2 operatori biți: Interschimbare

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

```
x = x ^ y
```

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

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

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



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
```

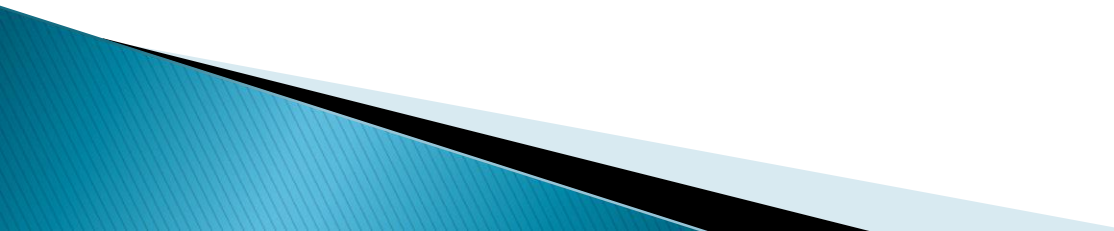
```
print(bin(x))
```

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

Instrucțiuni

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1 # atribuire inlantuita (multipla)
```

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

```
x, y = 1, 2 #atribuire simultana - de tupluri  
#tuple assignment
```

Instrucțiuni

- Instrucțiunea de atribuire

`x = 1`

`x = y = 1`

`x, y = 1, 2`

`x, y = y, x #????`

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

```
x, y = 1, 2
```

```
x, y = y, x ### Interschimbare
```

Instrucțiuni

- Instrucțiunea de atribuire

```
x, y = x if y > x else y, y if y > x else x  
# x, y = min(x,y), max(x,y)  
print("intervalul [" + str(x) + ", " + str(y) + "])"
```

Instrucțiuni

- Instrucțiunea de atribuire

Tema

```
v = [11, 12, 13, 14]
```

```
i = 2
```

```
i, v[i] = v[i], i
```

```
#v[i], i = i, v[i]
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

Varianta 1:

```
a = int(input("introduceti un numar nenegativ"))  
if a<0:  
    print("numar negativ, il consideram in modul")  
    a = -a
```


Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

Varianta 2:

```
a = int(input("a = "))
b = int(input("b = "))
if a > b:
    minim = a
else:
    minim = b
print("minimul dintre cele doua numere este ",minim)
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

Varianta 3:

```
k = int(input())
print('ultima cifra a lui 3**',k, 'este',end=" ")
r = k % 4
if r == 0:
    print(1)
elif r == 1:
    print(3)
elif r == 2:
    print(9)
else:
    print(7)
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

Varianta 3:

```
k = int(input())
print('ultima cifra a lui 3**',k, 'este',end=" ")
r = k % 4
if r == 0:
    print(1)
elif r == 1:
    print(3)
elif r == 2:
    print(9)
else:
    print(7)
```

- `else` poate lipsi
- Nu există `switch`

Instrucțiuni

- Instrucțiunea repetitiva cu test inițial while

```
#suma cifrelor unui numar
```

```
m = n = int(input())
```

```
s = 0
```

```
while n>0:
```

```
    s += n%10
```

```
    n //= 10 #!!nu /
```

```
print("suma cifrelor lui", m, "este",s)
```

Instrucțiuni

- **Instrucțiunea repetitiva cu test inițial while**
 - while poate avea else
 - Nu există instrucțiune repetitivă cu test final (do... while)

Instrucțiuni

- Instrucțiunea repetitiva cu număr fix de iterații (for)

Doar “for each”, de forma

for *variabila* in *colectie_iterabila*

de exemplu:

```
for litera in sir:
```

```
for elem in lista:
```

Instrucțiuni

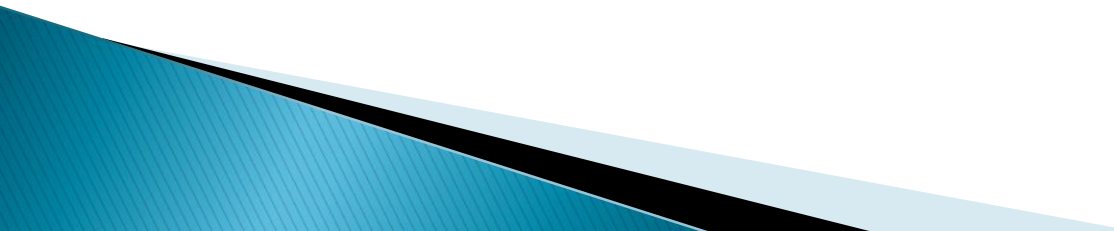
```
s = [3, 1, 8, 12, 5]
```

```
suma = 0
```

```
for nr in s:
```

```
    suma += nr
```

```
print(suma)
```



Instrucțiuni

```
s = [3, 1, 8, 12, 5]  
suma = 0
```

```
for nr in s:  
    suma += nr  
print(suma)
```

```
#parcugere cu indici
```

```
suma = 0
```

```
for i in [0,1,2,3,4]:  
    suma += s[i]  
print(suma)
```


Instrucțiuni

- Instrucțiunea repetitiva cu număr fix de iterații (for)

for i in [0,1,2,3,4]



for i in [0,..., n] **????!**

Instrucțiuni

- Instrucțiunea repetitiva cu număr fix de iterații (for)

```
for i in [0,1,2,3,4]
```

```
for i in [0,..., n] ????
```



Funcția `range()`

Instrucțiuni

- Instrucțiunea repetitiva cu număr fix de iterații (for)

```
for i in [0,1,2,3,4]
```

```
for i in [0,..., n] ????
```



Funcția `range()`

```
for i in range(0,n+1):
```

Instrucțiuni

- Funcția `range()` – clasa `range`, o secvență (iterabilă)

`range([min], max, [pas])` !exclusiv max

`range(b)` => de la 0 la `b-1`

`range(a,b)` => de la a la `b-1`

`range(a,b,pas)` => `a`, `a+p`, `a+2p`...

`pas` poate fi negativ

Instrucțiuni

```
s = "abcde"  
  
for i in range(len(s)):  
    print(s[i])
```

Instrucțiuni

`range(10)` =>

`range(1, 10)` =>

`range(10, 1)` =>

`range(10, 10)` =>

`range(1, 9, 2)` =>

`range(10, 1, -2)` =>

`range(1, 10, -2)` =>

Instrucțiuni

`range(10)` => **0 1 2 3 4 5 6 7 8 9**

`range(1,10)` => **1 2 3 4 5 6 7 8 9**

`range(10, 1)` =>

`range(10, 10)` =>

`range(1,9,2)` =>

`range(10,1,-2)` =>

`range(1,10,-2)` =>

Instrucțiuni

`range(10)` => **0 1 2 3 4 5 6 7 8 9**

`range(1,10)` => **1 2 3 4 5 6 7 8 9**

`range(10, 1)` => **vid**

`range(10, 10)` => **vid**

`range(1,9,2)` =>

`range(10,1,-2)` =>

`range(1,10,-2)` =>

Instrucțiuni

`range(10)` \Rightarrow **0 1 2 3 4 5 6 7 8 9**

`range(1,10)` \Rightarrow **1 2 3 4 5 6 7 8 9**

`range(10, 1)` \Rightarrow **vid**

`range(10, 10)` \Rightarrow **vid**

`range(1,9,2)` \Rightarrow **1 3 5 7**

`range(10,1,-2)` \Rightarrow **10 8 6 4 2**

`range(1,10,-2)` \Rightarrow **vid**

Instrucțiuni

```
print(range(1,10,2))
```

```
print(*range(1,10,2))
```

Instrucțiuni

- Funcția `range()` – clasa `range`, o secvență (iterabilă)
 - memorie puțină, **un element este generat doar cand este nevoie de el**, nu se memorează toate de la început (secvența este generată **element cu element**)

Instrucțiuni

- **break, continue + clauza else pentru instrucțiuni repetitive**
 - **break, continue** – aceeași semnificație ca în C

Instrucțiuni

- **break, continue + clauza else pentru instrucțiuni repetitive**

Exemplul 1 – Citirea de la tastatură de comenzi și afișarea lor până la introducerea comenzii exit

Instrucțiuni

- **break, continue + clauza else pentru instrucțiuni repetitive**

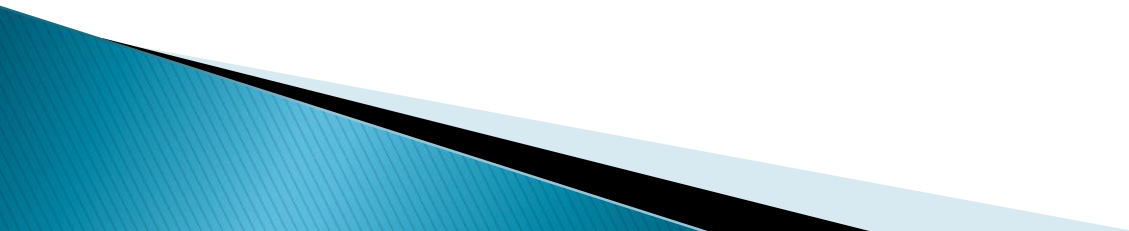
Exemplul 1 – Citirea de la tastatură de comenzi și afișarea lor până la introducerea comenzii exit

```
while True:
    comanda = input('>> ')
    if comanda == 'exit':
        break
    print(comanda)
```

Instrucțiuni

- **break, continue + clauza else pentru instrucțiuni repetitive**

Exemplul 2 – Numărul de divizori proprii ai lui n



Instrucțiuni

- **break, continue + clauza else pentru instrucțiuni repetitive**

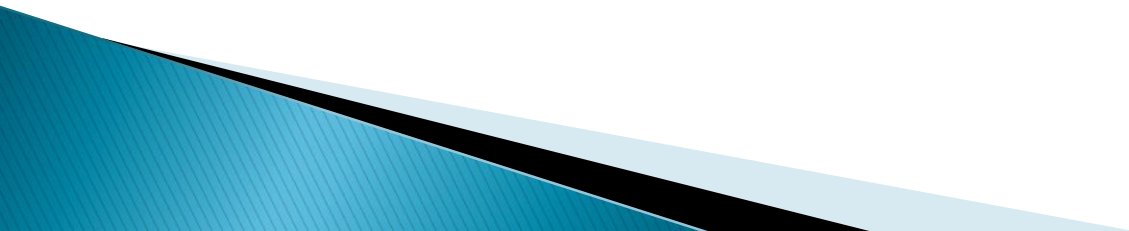
Exemplul 2 – Numărul de divizori proprii ai lui n

```
x = int(input())
k = 0
for d in range(2,x//2+1): # int(x**0.5)+1
    if x%d != 0:
        continue
    k+=1
print("numarul de divizori proprii:",k)
```


Instrucțiuni

- **break, continue + clauza else pentru instrucțiuni repetitive**

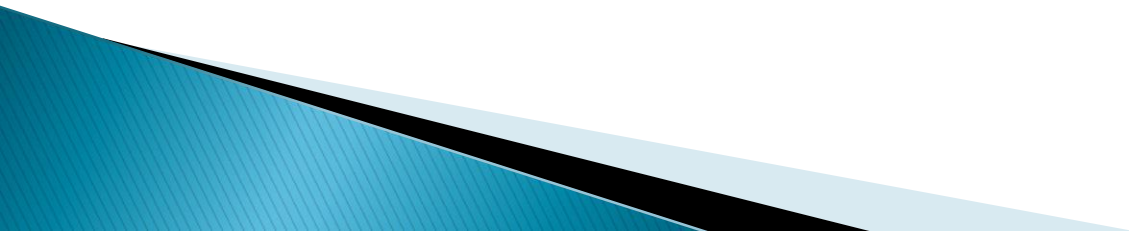
Exemplul 3 – Primul divizor propriu al lui n



Instrucțiuni

Clauza `else` a unei structure repetitive: nu se executa daca s-a iesit din ciclu cu `break`

Exemplul 1 – Determinarea primului divizor propriu al unui număr folosind clauza `else`.



Instrucțiuni

```
x = int(input())  
for d in range(2,x//2+1):  
    if x%d == 0:  
        print("primul divizor propriu:",d)  
        break  
else: #al for-ului, nu al if-ului  
    print("numar prim")
```

Instrucțiuni

Clauza `else` a unei structuri repetitive: nu se executa daca s-a iesit din ciclu cu `break`

Exemplul 2 – Determinarea primului număr prim din intervalul $[a,b]$, cu a și b citite de la tastatură.

Instrucțiuni

#cel mai mic număr prim din intervalul [a,b]

```
a = int(input("a="))
```

```
b = int(input("b="))
```

```
for x in range(a,b+1):
```

```
    for d in range(2, int(x**0.5)+1):#???sqrt?
```

```
        if x%d == 0:
```

```
            break
```

```
    else:
```

```
        print(x)
```

```
        break
```

```
else:
```

```
    print("Nu exista numar prim in interval")
```

Instrucțiuni

- **pass**

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

```
if x<0:
```

```
    pass #urmeaza sa fie implementat
```

Instrucțiuni

- **pass**

```
varsta = int(input())
```

```
if varsta <= 18:  
    print("Junior")
```

```
elif varsta < 65:  
    """nu prelucrăm informații despre persoane cu  
    vârstă cuprinsă între 19 și 64 de ani """
```

```
    pass
```

```
else:  
    print("Senior")
```

Funcții predefinite

- **Modulul builtins**

<https://docs.python.org/3/library/functions.html#built-in-funcs>

Funcții predefinite

- **Conversie**

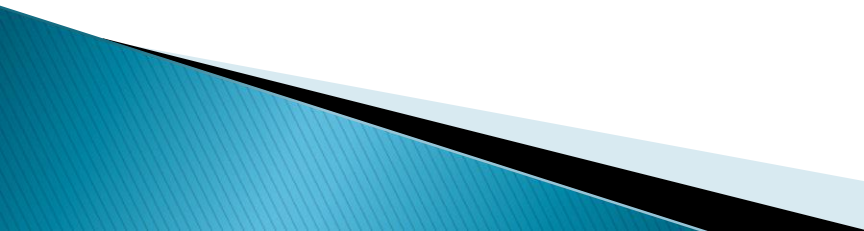
– constructori `int()`, `float()`, `str()`

```
print(bin(23), hex(23)) #str
```

- **Matematică**

```
print(abs(-5))
```

```
print(min(5, 2, 12))
```



Funcții predefinite

- Funcții din alte module

```
import math
```

```
print(math.sqrt(4))
```

```
print(math.factorial(5))
```

sau



Funcții predefinite

- Funcții din alte module

```
import math
```

```
print(math.sqrt(4))
```

```
print(math.factorial(5))
```

sau

```
from math import sqrt
```

```
print(sqrt(5))
```

```
print(factorial(5)) #eroare
```

```
# se putea: from math import sqrt, factorial
```

