

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
$x \geq y$	x mai mare sau egal y
$x \leq y$	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:

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

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
```

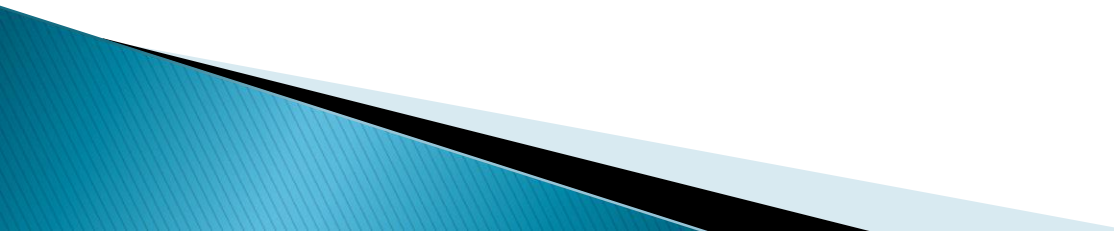
```
print(bin(x))
```

```
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 == 13 and media >= 9
```

Instrucțiuni

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1 #atriburie multipla
```

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

```
x, y = 1, 2 #atriburie compusa/tuple assignment
```


Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

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

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

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

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

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

```
x, y = min(x,y), max(x,y)
```

```
print("intervalul [" + str(x) + ", " + str(y) + "])"
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

Varianta 1. Instrucțiunea de decizie

```
if expresie_logică:  
    instructiuni
```

```
x = int(input())  
if x<0:  
    print('valoare incorecta')
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

Varianta 2. Instrucțiunea alternativă

```
if expresie_logică:  
    instructiuni_1  
else:  
    instructiuni_2
```

```
if x%2 == 0:  
    print('numar par')  
else:  
    print('numar impar')
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

Varianta 3. Instrucțiuni alternative imbricate

```
if expresie_logică:
    instructiuni_1
elif:
    instructiuni_2
elif:
    ...
else:
    instructiuni
```

← poate lipsi

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

```
k = int(input())  
print('ultima cifra a lui 3**',k, 'este',end=" ")
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

```
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

Instrucțiuni

- Instrucțiunea de decizie/potrivire multiplă `match`

`match expresie:`

`case tipar1: instructiune`

`case tipar2: instructiune`

`case _: instructiune` *#poate lipsi, similar default*

Instrucțiuni

- Instrucțiunea de decizie/potrivire multiplă `match`

```
x = int(input())
match x:
    case 1: print("luni")
    case 2: print("marti")
    case 3: print("miercuri")
    case 4|5: print("ultimele doua zile cu ore")
    case _: print("nu avem ore") #poate lipsi
```

Instrucțiuni

- Instrucțiunea de decizie/potrivire multiplă `match`

```
x = int(input())
match x:
    case n if n<10:
        print("o cifra")
    case n if n<100:
        print("doua cifre")
    case _:
        print("multe cifre")
```

Instrucțiuni

- Instrucțiunea repetitiva cu test inițial while

`#suma cifrelor unui numar`

Instrucțiuni

- Instrucțiunea repetitiva cu test inițial while

```
#suma cifrelor unui numar
```

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

```
s = 0
```

```
while n>0:
```

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 do... while

Instrucțiuni

- Instructiunea 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 = "abcde"
```

```
for litera in s:
```

```
    litera="a"
```

```
print(s)
```

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

```
    s[i]='a'
```


Instrucțiuni

```
s = "abcde"
```

```
for litera in s:
```

```
    litera="a"
```

```
print(s)      #nu se modifica, nu da eroare
```

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

```
    s[i]='a'  #eroare
```

```
#TypeError: 'str' object does not support item assignment
```



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(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

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

`range(b)` \Rightarrow de la 0 la $b-1$

`range(a,b)` \Rightarrow de la a la $b-1$

`range(a,b,pas)` $\Rightarrow a, a+p, a+2p...$

↑
`pas` poate fi negativ

- 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

`range(10) =>`

`range(1,10) =>`

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

`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(1,10,2) =>`

`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(1,10,2) => 1 3 5 7 9`

`range(10,1,-2) => 10 8 6 4 2`

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

Instrucțiuni

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

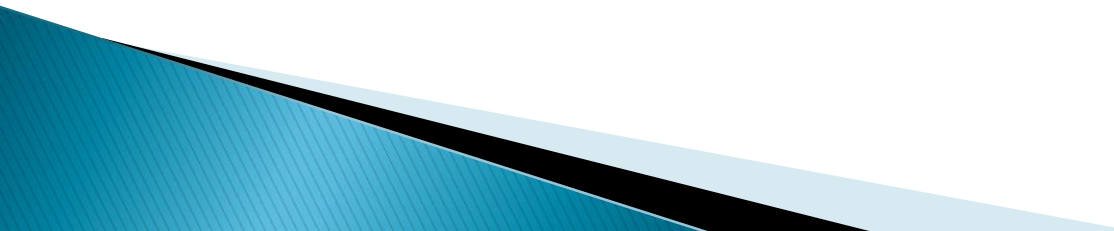
```
print(*range(10,1,-2))
```

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

```
s = "abcde"
```

```
for i in range(len(s)):
```

```
    print(s[i])
```



Instrucțiuni

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

Instrucțiuni

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

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

Instrucțiuni

#primul divizor propriu



Instrucțiuni

```
#primul divizor propriu
```

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

```
dx = None
```

```
for d in
```

Instrucțiuni

```
#primul divizor propriu
```

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

```
dx = None
```

```
for d in range(2,x//2+1):
```

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

```
        dx = d
```

```
        break
```



Instrucțiuni

```
#primul divizor propriu
```

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

```
dx = None
```

```
for d in range(2,x//2+1):
```

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

```
        dx = d
```

```
        break
```

```
if dx: #if dx is not None:
```

```
    print("primul divizor propriu:",dx)
```

```
else:
```

```
    print("numar prim")
```



Instrucțiuni

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

Instrucțiuni

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

```
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

```
#numarul de divizori proprii - cu continue
```

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

```
k = 0
```

```
for d in range(2,x//2+1):
```

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

```
        continue
```

```
    k+=1
```

```
print("numarul de divizori proprii:",k)
```

Instrucțiuni

Exercițiu: Date a și b , să se determine cel mai mic număr prim din intervalul $[a, b]$

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):
```



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,x//2+1):  
        if x%d == 0:  
            break
```

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,x//2+1):
        if x%d == 0:
            break
    else:
        print(x)
        break
```

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,x//2+1):
        if x%d == 0:
            break
    else:
        print(x)
        break
else:
    print("Nu exista numar prim in interval")
```


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):
        if x%d == 0:
            break
    else:
        print(x)
        break
else:
    print("Nu exista numar prim in interval")
```

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
```

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
```

Funcții predefinite

- **Matematică**

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

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

```
x = 3.0
```

```
print(x.is_integer())
```

Funcții predefinite

- **Matematică**

```
import math
```

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

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

Funcții predefinite

- **Matematică**

```
from math import sqrt
```

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

```
print(factorial(5)) #eroare, se putea: from math  
import sqrt, factorial
```

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

```
print(math.factorial(4)) #eroare
```