

Evaluare – update



Evaluate

- ▶ Test de laborator – prima sâmbătă din ianuarie după vacanță (9 ianuarie 2021)
- ▶ Examen în sesiune

Nota finală = media celor două note

Condiții necesare:

Nota test laborator ≥ 5

Nota examen ≥ 5

Variabile – amintim

Variabile

- În C/C++ o variabilă 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)**

Variable

C

Python

`m = 10`

`m:` 10

`m` → 10

Variable

C

Python

```
m = 10
```

m: 10

m → 10

```
m = m+1
```

Variable

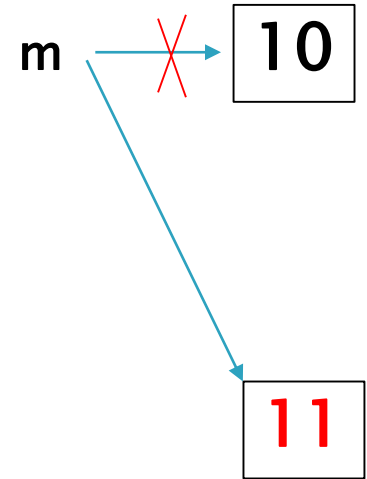
```
m = 10
```

```
m = m+1
```

C

m: 11

Python



Variable

```
m = 10
```

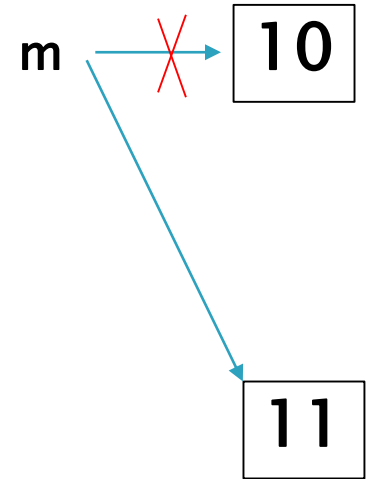
```
m = m+1
```

```
n = m
```

C

m: 11

Python



Variable

C

Python

```
m = 10
```

m: 11

m ~~→~~ 10

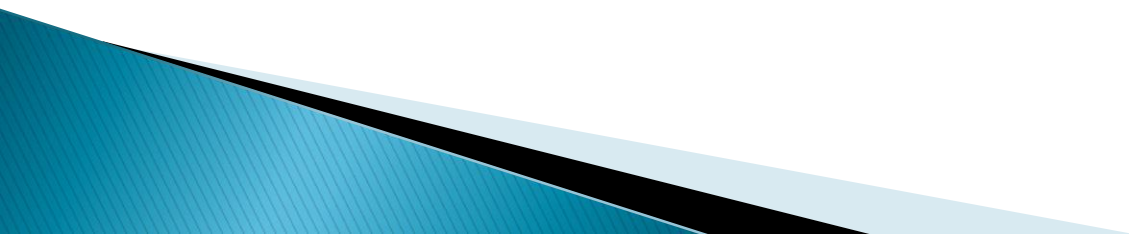
```
m = m+1
```

n: 11

11

```
n = m
```

n



Variable

```
m = 10
```

```
m = m+1
```

```
n = m
```


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


C

m: 11

n: 11

Python

m  10

 11

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

▶ Exemple

```
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))
```

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
- 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)`)

```
int(9.5)    #round(9.5)
```

```
int("101",base=2)
```

```
int("101",2)
```

Tipuri de date

- **float**

- IEEE-754 double precision
- Constante: 3.5, 1e-2 (notație științifică)
- float([x]):

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

Tipuri de date

- `float`
 - operațiile aritmetice cu tipul de date *float* nu au precizie absolută:

NU: `0.1*0.1 == 0.01`

DA: `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
- tupluri `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

- aritate (număr de operanzi)
- prioritate (precedența)
- asociativitatea: $x \text{ op } y \text{ op } z$

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

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

`10**7**2`  `10**(7**2)`

!!! Prioritățile și asociativitatea operatorilor

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 daca obiectele au *acelasi id*

```
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)
```

!! variabile din cache

Operatori

- Operatori de atribuire

=

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

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

Operatori

- Operatori logici

`not, and, or`

- se evaluează prin scurtcircuitare

—

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)

Operatori

- Operatori logici

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

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

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

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

Observație: `not` are prioritate mai mică decât operatorii de alte tipuri:

`not a==b` \Leftrightarrow `not (a == b)`

`a == not b` eroare de sintaxă (trebuie `a == not(b)`)

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

\wedge	0	1
0	0	1
1	1	0

Operatori

Observații:

$x = x \gg k \iff x = x // (2^{}k)$**

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

Operatori

```
x = 272
```

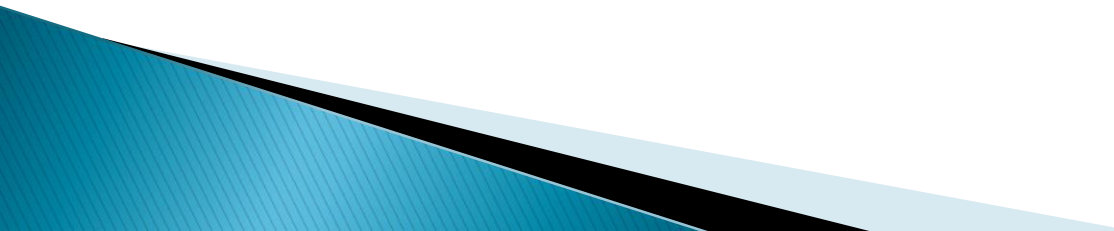
```
print(bin(x))
```

```
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

print(z)



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(z)
```

- evaluat tot prin scurtcircuitare
- orice expresie

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

```
print(x) if x>y else print(y)
```

Operatori

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

Operatori

- **Precedența operatorilor:**

<https://docs.python.org/3/reference/expressions.html>

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

Instrucțiuni

- Instrucțiunea de atribuire

x = 1

x = y = 1

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

```
x, y = 1, 2 #atribuire de tupluri
```

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 atribuire

```
x = 1
```

```
x = y = 1
```

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

```
x, y = y, x #!!! Interschimbare (tupluri)
```

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

```
i = 2
```

```
i, v[i] = v[i], i !!!???????
```

Instrucțiuni

- Instrucțiunea de atribuire

```
x = 1
```

```
x = y = 1
```

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

```
x, y = y, x #!!! Interschimbare (tupluri)
```

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

```
i = 2
```

```
i, v[i] = v[i], i #mai bine v[i], i = i, v[i]?
```

Instrucțiuni

- Instrucțiunea de decizie (condițională) `if`

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

```
if x<0:
```

```
    print('valoare incorecta')
```

```
if x%2==0:
```

```
    print('numar par')
```

```
else:
```

```
    print('numar impar')
```

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)
```

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
- Nu există `switch`

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

- Instrucțiunea repetitiva cu număr fix de iterații (for)

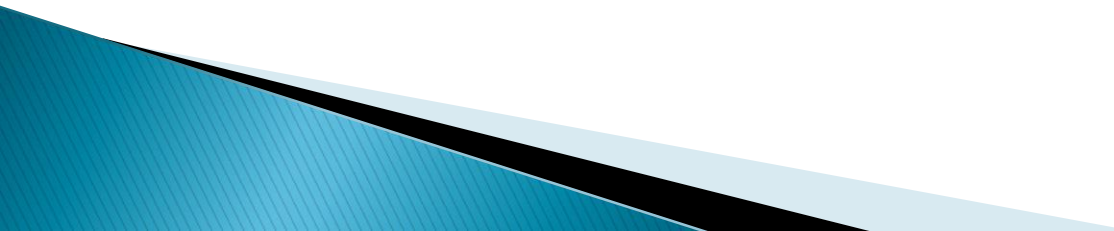
Doar “for each”, de forma

for *variabila* in *colectie_iterabila*

de exemplu:

```
for litera in sir:
```

```
for elem in lista:
```



Instrucțiuni

```
s = "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))
```

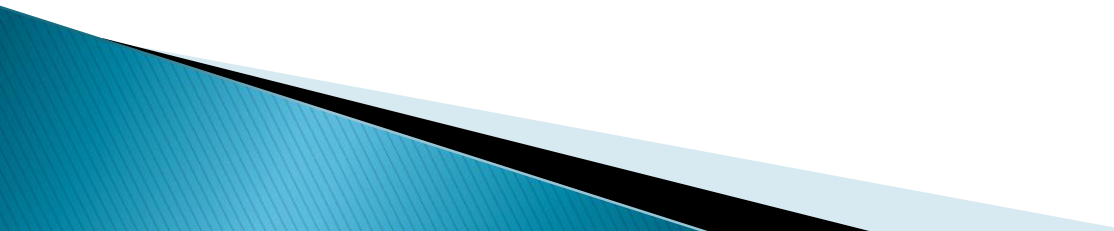
```
print(*range(10,1,-2))
```

```
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

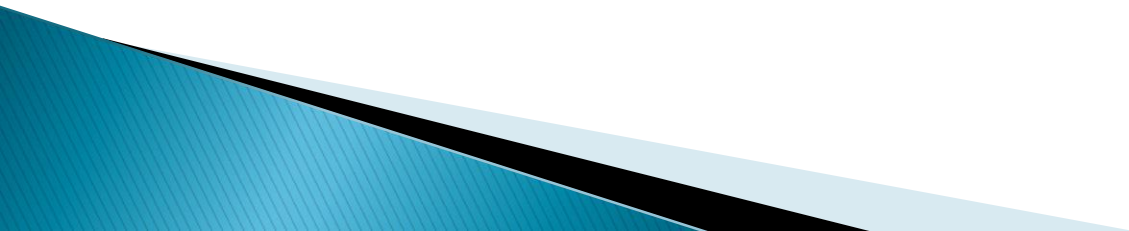
Instrucțiuni

- **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 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

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())
```

```
print(round(x + 0.7))
```

```
import math
```

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



