Pe calculatoarele personale trebuie să vă instalați o versiune cât mai recentă de **Python 3** (https://www.python.org/downloads/).

Ca IDE recomandăm să vă instalați **PyCharm** (https://www.jetbrains.com/pycharm/download/).

Cuprins

Despre sintaxa Python:

- Blocurile de cod se grupează folosind același nivel de indentare pentru toate instrucțiunile din acel bloc (în loc de folosirea acoladelor {} cum era în C/C++). Pentru indentare se folosesc 4 spații (recomandabil) sau un tab. Trebuie ca în întreg fișierul să se folosească aceeași indentare (fie câte 4 spații, fie câte un tab).
- Nu se folosește ”;” la finalul instrucțiunilor (ca în C/C++), ci doar între instrucțiuni, atunci când se scriu mai multe pe același rând. Dar pentru claritatea codului, nu este recomandat acest lucru, ci fiecare instrucțiune se scrie pe un alt rând.
- Python este case-sensitive (se face diferența dintre literă mică/mare).

Comentarii:

- Pe un singur rând: se folosește ”**#**” (în loc de ”//” ca în C/C++) și tot ce este în dreapta pe acel rând se consideră comentariu.
- Pe mai multe rânduri: se include tot comentariul între câte 3 ghilimele/apostrofuri (**”””**comentariu…**”””** sau **’’’**comentariu… **’’’**) (în loc de: /* comentariu… */ ca în C/C++).
- În PyCharm, pentru a (de)comenta rapid o porțiune de cod, selectați acele rânduri și apăsați tastele ”ctrl + /”.

Citirea datelor de la tastatură:

- Se apelează funcția ”**input**” fără parametru sau cu un parametru de tip *string*, care va fi afișat pe ecran (stringurile se includ între ghilimele/apostrofuri). De la tastatură se va introduce inputul dorit, apoi se apasă Enter.
- Funcția ”input” returnează mereu un *string*, deci rezultatul trebuie apoi transformat în alt tip de date, atunci când este nevoie.

```
x = input()
y = input(”y= ”)          # în C/C++: cout<<”y= ”; cin>>y;
z = input(’z= ’)
nr_intreg = int(input(”numarul intreg este: ”))
nr_real = float(input(”numarul real este: ”))
```

Afișarea datelor pe ecran:

- Se apelează funcția ”**print**”: `print(*objects, sep=' ', end='\n')`
- În loc de ”`*objects`” se pun unul sau mai multe obiecte (cu virgulă între ele) care trebuie afișate.
- Parametrul ”**sep**” (de la ”separator”) este opțional. Dacă se afișează mai multe obiecte cu un singur apel al funcției ”print”, între ele se va afișa implicit câte un spațiu. Dacă doriți altceva, trebuie să-i atribuiți o valoare (neapărat de tip *string*) parametrului ”sep”.
  Exemplu: `sep=", "` (virgulă și spațiu), `sep=";"`, `sep="\n"` (rând nou), `sep=""` (nimic) etc.
- Parametrul ”**end**” este opțional. Implicit, la finalul afișarii se va trece la un rând nou. Dacă doriți ca după afișare să pună altceva în loc de rând nou, trebuie să-i atribuiți o valoare (neapărat de tip *string*) parametrului ”end”.
  Exemplu: `end=", "` (virgulă și spațiu), `end=";"` etc.

```
print(3, 4.5, True, "ana")    # se vor afișa cu spațiu între ele
print("alt print")            # se va afișa pe rând nou
```

sau

```
print(3, 4.5, True, "ana", sep=",", end=";")
     # se vor afișa cu ”,” între ele, iar la final cu ”;”
print("alt print")  # se va afișa pe același rând, după ”;”
```

Variabile:

- Variabilele nu trebuie declarate (nu au tip de date static), ci doar **inițializate** cu o valoare (tipul de date se stabilește automat la atribuire). Atribuirea se face cu instrucțiunea ”**=**”.
- Orice dată (valoare) este un *obiect*. Variabilele sunt **referințe** către obiecte.
- **Numele unei variabile** poate conține litere mari, litere mici, simbolul underscore (“_”) și cifre (dar NU poate să înceapă cu o cifră). ***Exemplu:*** var, Var, VAR, _var, myVar, my_Var, var3.
- Pentru a afla *adresa* din memorie a unei variabile ”var”: `id(var)`
- Pentru a afla *tipul de date* al unei variablile ”var”: `type(var)`

Tipuri de date:

- Tipurile de date sunt **clase**.

1) **Clasa ”NoneType”** conține doar valoarea ”**None**” (se folosește pentru a verifica existența unui obiect).
2) *Tipuri numerice:*

a) **Clasa ”bool”** conține doar valorile ”**True**” și ”**False**” (atenție, se scriu mereu cu prima literă mare).

b) **Clasa ”int”** conține numere întregi de dimensiune oricât de mare (cât permite memoria calculatorului).

c) **Clasa ”float”** conține numere reale (corespunde tipului ”double” din C/C++). Se folosește ”.” între partea întreagă și zecimale (ex: 3.75). Se permite și scrierea: 17e3 (adică $17 * 10^3$).

d) **Clasa ”complex”** conține numere complexe: a+bj, unde a și b sunt numere (întregi sau reale), a reprezintă partea reală, iar b reprezintă partea imaginară. Atenție, nu se pune ”*” între b și j, iar valoarea lui b trebuie obligatoriu specificată, chiar daca este 1. Ex: 4+5j, 3+1j, 2j, complex(4,5).

3) *Tipuri secvențiale:*
   - tupluri (clasa “tuple”), liste (clasa “list”), șiruri de caractere (clasa “str”)
   - bytes / bytesarray
4) *Tipuri mulțime:* set, frozenset
5) *Tipul dicționar*: dict
6) *Tipuri funcționale*

Operatori (numerici, relaționali, logici, pe biți, condițional):

a) operatori numerici:

+, -, *, / (împărțire cu virgulă), // (împărțire ”întreagă”), %, ** (ridicare la putere).

Atenție! În Python nu există operatorii ++ și --. Dar există +=, -=, *= etc.

a//b = cel mai mare întreg care este mai mic sau egal decât a/b

a%b = a – (a//b)*b

a**b = a^b

x = y ** 0.5 # x este radical din y

Obs: Pentru a folosi funcția radical, trebuie la începutul fișierului cu codul (.py) să aveți instrucțiunea

```
import math
```

apoi pentru a folosi functia:

```
x = math.sqrt(y)
```

b) operatori relaționali:

<, <=, >=, > (NU pentru numere complexe)

==, != (compară *valorile*)

is, is not (compară *adresele* din memorie) (x is y) <=> (id(x) == id(y))

in, not in (pentru testarea apartenenței unei valori la o colecție)

- Se permite înlănțuirea operatorilor relaționali: `if a <= y < b:` …
- Pentru compararea numerelor reale: `abs(x-y)<=1e-9` (NU folosiți x==y)

c) operatori logici (booleeni): not, and, or

False <=> 0, 0.0, 0+0j; colecție_vidă: (), [], {}, “”, ‘’ etc

True <=> restul

- Operatorii logici se evaluează prin scurtcircuitare (adică nu se evaluează toate condițiile dacă se știe deja rezultatul: False pentru “and” și True pentru “or”).
- NU folosiți apeluri de funcții în condiții!
- $x\ or\ y = \begin{cases} x, daca\ x\ este\ True \\ y, daca\ x\ este\ False \end{cases}$ 5 or 3.7 == 5
- $x\ and\ y = \begin{cases} y, daca\ x\ este\ True \\ x, daca\ x\ este\ False \end{cases}$ 5 and 0.0 == 0.0

d) operatori pe biți:

~ (negare), & (și), | (sau), ^ (XOR/sau exclusiv)

<<, >> (deplasări/shiftări pe biți către stânga sau dreapta)

~	0	1
	1	0

&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

- Proprietăți ^ (XOR):
a) x ^ x = 0, x este bit
b) x ^ 0 = x, x este bit sau număr

x = (x<<b) <=> x = x * (2**b) b = număr natural

x = (x>>b) <=> x = x // (2**b)

- Operatorii pe biți acționează asupra reprezentării binare interne; se execută rapid, pe procesor.

- Aplicații:

1) Testarea parității unui număr (mai rapid decât cu %)

```
if x & 1 == 1:
    print("Numar impar.")
else:
    print("Numar par.")
```

2) Interschimbare variabile

```
x = x ^ y
y = x ^ y    # = (x^y)^y = x^(y^y) = x^0 = x
x = x ^ y    # = (x^y)^x = y^(x^x) = y^0 = y
```

e) operatorul condițional:

`valoare1 if conditie==True else valoare2`

max = a if a > b else b

Un operator are următoarele proprietăți:

- **aritatea** (numărul de operanzi) 1/2/3 -> operator unar/binar/ternar
- **prioritatea/precedența**
  https://docs.python.org/3/reference/expressions.html#operator-precedence
  Atenție, excepție: `2**-1 = 0.5` (nu necesită paranteze: 2**(-1))
  `x == (not y)` (obligatoriu cu paranteze, altfel dă eroare de sintaxă)
- **asociativitatea** -> majoritatea au de la stânga la dreapta
  Atenție, operatorul ** are asociativitate de la dreapta la stânga.
  2**3**2 = 2**(3**2) = 2**9 = 512 (NU (2**3)**2=8**2=64)

Instrucțiuni (if, for, while, …):

1) instrucțiunea de atribuire:

```
x = 100
x = y = 100
a, b, c = 1, 2, 3
```
=> interschimbare: `x, y = y, x`

2) instrucțiunea de decizie: cuvinte cheie "**if**", "**elif**", "**else**".

- Atentie la indentări și la ":" de la finalul acelor instrucțiuni
- Nu este nevoie să puneți condițiile între paranteze.
- Pot fi 0 sau oricât de multe ramuri cu "elif", iar ramura cu "else" poate lipsi.
- În Python NU există instrucțiunea "switch/case".

```
if conditie_1:
    bloc_instructiuni_1
elif conditie_2:
    bloc_instructiuni_2
elif conditie_3:
    bloc_instructiuni_3
else:
    bloc_instructiuni_4
```

3) instrucțiunea repetitivă cu test inițial: cuvinte cheie "**while**".

- Atenție la indentare și la ":" de la finalul instrucțiunii.
- În Python NU există instrucțiunea "do… while".

```
while conditie:
    bloc_instructiuni
```

4) instrucțiunea repetitivă cu număr fix de iterații/pași: cuvinte cheie ”**for**”, “**in**”.

- Atenție la indentare și la ”:” de la finalul instrucțiunii.

```
for variabila in colectie_iterabila:
    bloc_instructiuni
```

a) parcurgere caractere dintr-un string

```
s = "test"
for x in s:
    print(x)
```

b) parcurgere elemente dintr-o listă

```
L = [1,2,3]
for x in L:
    print(x)
```

c) parcurgere interval de numere întregi cu funcția “**range**”

```
range(b) <=> range(0,b)     => 0, 1, ..., b-1
range(a,b) <=> range(a,b,1) => a, a+1, a+2, ..., b-1
                                sau nimic daca a>=b
range(a,b,pas)  => a, a+pas, a+2*pas, ..., (a+k*pas)!=b
    cu a<b si pas>0, sau a>b si pas<0, sau nimic altfel
```

```
for i in range(10, 20, 2):        #10, 12, 14, 16, 18
  print(i)
for j in range(5, -5, -1):        #5, 4, 3, 2, 1, 0, -1, -2, -3, -4
  print(j)
```

5) instrucțiunea ”continue” aflată în cadrul unui bloc “for” sau “while”:

- Se ignoră instrucțiunile de după și se trece la următoarea iterație din for/while.

```
for x in range(n+1):
    if x%2 == 0:
        continue # se ignora instructiunile de dupa ”continue”,
                 # se trece la urmatoarea iteratie din for
    print(x) # se vor afisa doar nr impare: 1,3,5, ... <=n
```

6) instrucțiunea “break” aflată în cadrul unui bloc “for” sau “while”:

- Se iese din ciclul for/while cel mai apropiat.

```
for x in range(5):
    for y in range(10):
        if y > x:
            print("\nx =", x, "\n")
            break # se iese din for-ul cu y
        print("y="+str(y), end = " ")
```

7) instrucțiunea “pass”:

- Instrucțiunea vidă, nu face nimic.

```
if conditie_1:
    if conditie_2:
        bloc_instructiuni_1
    else:
        pass
else:
    bloc_instructiuni_2
```

8) clauza ”else” după un ciclu “for” sau ”while”:

- Se execută dacă ciclul a fost parcurs integral, fără să fie intrerupt de instrucțiunea “break”.

```
a, b = 20, 50
for x in range(a, b+1):
    d = 2
    while d <= x//2:
        if x % d == 0:
            break # intrerupe while-ul
        d = d + 1
    else: # daca x nu a avut niciun divizor d
        print(f"Cel mai mic numar prim intre {a} si {b} este: {x}.")
        break # intrerupe for-ul
else:
    print(f"Nu exista niciun numar prim intre {a} si {b}.")
```