Introducere în Python

- Python:
  - folosim ‘new line’ pentru a semnala terminarea unei instrucțiuni (nu folosim “;” ca în alte limbaje de programare)
  - se bazează pe indentare folosind spații albe sau tab pentru a defini scopul variabilelor, funcțiilor sau claselor (folosim indentare în loc de “{}”)

- Tipuri de date primitive

  Numbers: int și float se comportă la fel ca în alte limbaje**.**

```
x = 3
print(type(x)) # Afiseaza "<class 'int'>"
print(x)       # Afiseaza "3"
print(x + 1)   # Adunare; Afiseaza "4"
print(x - 1)   # Scadere; Afiseaza "2"
print(x * 2)   # Inmultire; Afiseaza "6"
print(x ** 2)  # Exponential; Afiseaza "9"
x += 1
print(x)  # Afiseaza "4"
x *= 2
print(x)  # Afiseaza "8"
y = 2.5
print(type(y)) # Afiseaza "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Afiseaza "2.5 3.5 5.0 6.25"
```

  **În Python nu există operațiile de incrementare x++ (++x) sau x-- (--x).*

  Bool: Pentru operațiile cu variabile de tip bool nu se folosesc simbolurile && și ||, ci *and* sau *or*.

```
t = True
f = False
print(type(t)) # Afiseaza "<class 'bool'>"
print(t and f) # Logical AND; Afiseaza "False"
print(t or f)  # Logical OR; Afiseaza "True"
print(not t)   # Logical NOT; Afiseaza "False"
print(t != f)  # Logical XOR; Afiseaza "True"
```

String: Python oferă multe funcții pentru șirurile de caractere.

```
hello = 'hello' # Un string se declara intre ghilimele simple
world = "world"        # sau intre ghilimele duble
print(hello)           # Afiseaza "hello"
print(len(hello))      # Lungimea sirului de caracter; Afiseaza 5
hw = hello + ' ' + world # Concatenare
print(hw)                # Afiseaza "hello world"
hw12 = '%s %s %d' % (hello, world, 12) # sprint - formatarea
                                       # sirurilor de caractere
print(hw12)  # Afiseaza "hello world 12"

s = "hello"
print(s.capitalize())       # Afiseaza "Hello"
print(s.upper())            # Afiseaza "HELLO"
print(s.replace('l', '(ell)')) # Inlocuieste toate instantele
                               # primului argument cu al doilea
                              # argument,Afiseaza "he(ell)(ell)o"
print(' hello     world     '.strip() ) # Sterge spatiile albe
                                      # de la inceput si sfarsit;
                                      # Afiseaza "hello      world"
```

- Colecții

În Python există următoarele colecții: listă, dicționar, set și tuplu.

Liste: O lista este echivalentă cu un vector, dar se poate redimensiona și poate conține elemente de tipuri diferite.

```
xs = [3, 1, 2]     # Creaza o lista
```

```
print(xs, xs[2])    # Afiseaza "[3, 1, 2] 2"
print(xs[-1])       # Indicii negativi numara
                    # de la sfarsitul listei; Afiseaza "2"
xs[2] = 'foo'       # Listele pot contine
                    # elemente de tipuri diferite
print(xs)           # Afiseaza "[3, 1, 'foo']"
xs.append('bar')    # Adaugare la sfarsitul listei
xs += ['foobar']    # Adaugare la sfarsitul listei
print(xs)           # Afiseaza "[3, 1, 'foo', 'bar', 'foobar' ]"
x = xs.pop()        # Elimina si returneaza ultimul
                    # element al listei
y = xs.pop(-1)      # Elimina si returneaza elementul de pe pozitia
                    # i din lista
print(x, y, xs)     # Afiseaza "foobar bar [3, 1, 'foo']"
```

Slicing – putem accesa o sublistă a unei liste

```
nums = list(range(5))   # range(n) este o functie care creeaza o
                        # lista de intregi, pornind de la 0 pana
                        # la n-1, iar range(a, b), creaza
                        # o lista cu numere intregi de la [a, b-1]
print(nums)             # Afiseaza "[0, 1, 2, 3, 4]"
print(nums[2:4])        # Acceseaza o sublista de la indicele 2 la
                        # 4 (exclusiv); Afiseaza "[2, 3]"
print(nums[2:])         # Acceseaza o sublista de la indicele 2 la
                        # sfarsitul listei ; Afiseaza "[2,3,4]"
print(nums[:2])         # Acceseaza o sublista de la inceputul
                        # listei pana la indicele 2 (exclusive);
                        # Afiseaza "[0, 1]"
print(nums[:])          # Acceseaza lista; Afiseaza "[0,1,2,3,4]"
print(nums[:-1])        # Acceseaza lista de la primul pana la
                        # penultimul element; Afiseaza "[0,1,2,3]"
nums[2:4] = [8, 9]      #
print(nums)             # Afiseaza "[0, 1, 8, 9, 4]"

numbers = list(range(5, 10)) # Numerele de la 5 la 9 (inclusiv)
nums_reverse = numbers[::-1] # Parcurgem lista de la ultimul
                             # element  pana la primul.
print(nums_reverse)    # Afiseaza "[9, 8, 7, 6, 5]"
```

Bucle: Putem itera prin elementele unei liste în felul următor:

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
      print(animal)
# Afiseaza "cat", "dog", "monkey", fiecare pe cate o linie.
```

Dacă vrem să accesăm indicele fiecărui element în corpul buclei, putem folosi funcția ***enumerate***.

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
      print('#%d: %s' % (idx + 1, animal))
# Afiseaza "#1: cat", "#2: dog", "#3: monkey", fiecare pe cate o
linie.
```

Dacă avem două liste X și Y și trebuie să formăm perechi (x_i, y_i) cu elementele listelor, atunci putem folosi funcția **zip**.

```
X = [1, 2, 3, 4, 5]
Y = [9, 8, 7]
# inmultim listele element cu element (element wise)
xy_dot = [a * b for (a, b) in zip(X, Y)]
# se formeaza perechi cu al i-lea element din prima lista si al i-
lea din a doua lista
print(xy_dot) # Afiseaza [9, 16, 21]
```

List comprehensions: Deseori vrem să transformăm un tip de date într-un altul. Ca exemplu, să considerăm următorul cod care calculează pătratele numerelor:

Commented [1]: limba romana? :-)

Commented [2]: nu am gasit o traducere care sa sune bine, ne puteti ajuta cu o sugestie :) ?

Commented [3]: ma gandesc pana diseara ;)

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
      squares.append(x ** 2)
print(squares)   # Afiseaza [0, 1, 4, 9, 16]
```

Putem scrie codul mult mai simplu folosind *list comprehensions:*

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Afiseaza [0, 1, 4, 9, 16]
```

De asemenea, *list comprehensions* poate conține și condiții:

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Afiseaza "[0, 4, 16]"
```

Putem folosi *list comprehensions* pentru a transforma o listă cu valori bool într-o listă cu valori de 0 și 1.

```
bool_list = [True, False, True, True, False]
num_list = [x * 1 for x in bool_list]
print(num_list) # Afiseaza [1, 0, 1, 1, 0]
```

Dicționare: Un dicționar stochează perechi (cheie, valoare).

```
d = {'cat': 'cute', 'dog': 'furry'}  # Creaza un dictionar
print(d['cat'])       # Acceseaza o valoare
                      # in functie de cheia din
                      # dictionar. Afiseaza "cute"
print('cat' in d)     # Verifica daca in dictionar
                      # exista cheia; Afiseaza "True"
d['fish'] = 'wet'     # Adauga o noua pereche in dictionar
print(d['fish'])      # Afiseaza "wet"
print(d['monkey'])    # KeyError: cheia 'monkey'
                      # nu exista in dictionar
print(d.get('monkey', 'N/A'))  # Acceseaza o valoare
                               # din dictionar pe baza cheii,
                               # iar daca nu exista se returneaza
                               # valoarea default 'N/A';
                               # Afiseaza "N/A"
print(d.get('fish', 'N/A'))  # Afiseaza "wet"
del d['fish']                # Sterge elementul din dictionar
print(d.get('fish', 'N/A')) # "fish" nu mai este cheie;
                           # Afiseaza "N/A"
```

Este ușor să iterăm prin cheile unui dicționar:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
      legs = d[animal]
      print('A %s has %d legs' % (animal, legs))
```

```
# Afiseaza "A person has 2 legs", "A cat has 4 legs", "A spider
has 8 legs"

# Afisam cheile din dictionar
for key in d.keys():
      print(key)
# Afiseaza spider, person, cat cate o cheie pe un rand.

# Afisam valorile din dictionar
for value in d.values():
      print(value)
# Afiseaza 8, 4, 2 cate un numar pe un rand.
```

Dacă dorim să accesăm cheia și valoarea corespunzătoare, putem folosi metoda *items*:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
      print('A %s has %d legs' % (animal, legs))
# Afiseaza "A person has 2 legs", "A cat has 4 legs", "A spider
has 8 legs"
```

Dictionary comprehensions: Similar cu **list comprehensions**, dar ne permite să construim dicționare mai ușor. De exemplu:

```
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)  # Afiseaza "{0: 0, 2: 4, 4: 16}"
```

Set: Un set este o colecție neordonată de elemente distincte.

```
animals = {'cat', 'dog'}
print('cat' in animals)     # Verifica daca un element se afla
                            # in set; Afiseaza "True"
print('fish' in animals)  # Afiseaza "False"
animals.add('fish')       # Adauga un element in set
print('fish' in animals)  # Afiseaza "True"
print(len(animals))    # Numarul elementelor in set; Afiseaza "3"
animals.add('cat')     # Adaugarea unui element care deja
                       # exista in set, nu are niciun efect
print(len(animals))    # Afiseaza "3"
```

```
animals.remove('cat')  # Elimina un element din set
print(len(animals))    # Afiseaza "2"

# operatii cu multimi

A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# reuniunea
print(A | B) # Afiseaza {1, 2, 3, 4, 5, 6, 7, 8}
print(A.union(B))

# intersectia
print(A & B) # Afiseaza {4, 5}
print(A.intersection(B))

# diferenta
print(A - B) # Afiseaza {1, 2, 3}
print(A.difference(B))

# diferenta simetrica - elementele din reuniune care nu sunt in
intersectie
print(A ^ B) # Afiseaza {1, 2, 3, 6, 7, 8}
A.symmetric_difference(B)
```

Set comprehensions: Similar cu **dictionary comprehensions**, ne permite sa construim set-uri mai usor. De exemplu:

```
from math import sqrt
nums = {int(sqrt(x)) for x in range(30)}
print(nums)  # Afiseaza "{0, 1, 2, 3, 4, 5}"
```

Tupluri: Un *tuplu* este o listă (immutable) ordonată de valori. *Tuplul* este foarte similar cu o listă, o diferență fiind că tuplul poate fi folosit ca cheie în dicționar și ca element al unui set, în schimb lista nu poate fi folosită în aceste scopuri.

```
# Creaza un dictionar cu chei de tip tuple
d = {(x, x + 1): x for x in range(10)}
t = (5, 6)       # Creaza un tuple
```

```
print(type(t))   # Afiseaza "<class 'tuple'>"
print(d[t])      # Afiseaza "5"
print(d[(1, 2)]) # Afiseaza "1"
t2 = ('cat', 5, 6)     # Creaza un tuple
print(t2[0])           # Afiseaza "cat"
# Creaza o lista de tuple
list_tuples = [('cat', 5, 6), ('dog', 8, 1)]
print(len(list_tuples))      # Afiseaza "2"
print(list_tuples[1][0])     # Afiseaza "dog"
```

- Funcții

  În Python funcțiile sunt definite cu ajutorul cuvântului cheie ***def***.

```
def sign(x):
      if x > 0:
            return 'positive'
      elif x < 0:
            return 'negative'
      else:
            return 'zero'

for x in [-1, 0, 1]:
      print(sign(x))
# Afiseaza "negative", "zero", "positive"
```

Deseori vom defini funcții care au argumente cu valori implicite, ca în exemplul următor:

```
def hello(name, loud=False):
      if loud:
            print('HELLO, %s!' % name.upper())
      else:
            print('Hello, %s' % name)

hello('Bob') # Afiseaza "Hello, Bob"
hello('Fred', loud=True)  # Afiseaza "HELLO, FRED!"
```

Vom defini o funcție care primește două liste și returnează *True* doar dacă listele au lungimea diferită și elementul minim din prima listă nu se află în a doua listă sau elementul minim din a doua listă se află în prima listă, altfel va returna *False*.

```
def my_function(first_list, second_list):

    if(len(first_list) == 0 or len(second_list) == 0):
        raise ValueError('Lists must not be empty!')

    min_first = min(first_list)
    min_second = min(second_list)
    if (min_first not in second_list or min_second in first_list)
and len(first_list) != len(second_list):
        return True
    else:
        return False

first_list = [1, 2, 3, 4]
second_list = [0, 1, 3]
print(my_function(first_list, second_list)) # Afiseaza False

first_list = [-1, 1, 2, 3, 4]
second_list = [0, 1, 3]
print(my_function(first_list, second_list)) # Afiseaza True

first_list = [1, 2, 3, 4]
second_list = [0, 1, 3, 9]
print(my_function(first_list, second_list)) # Afiseaza False

first_list = [1, 2, 3, 4]
second_list = []
print(my_function(first_list, second_list)) # ValueError: Lists
must not be empty!
```

Vom defini o funcție care va returna două numere naturale diferite, generate aleator în intervalul [0, 5].

```
import random # importul pachetului

def two_rand_nums():
    a = random.randint(0, 5)
    b = random.randint(0, 5)
    while(a == b):
        print('a == b')
        b = random.randint(0, 5)

    return a, b
```

```
print(two_rand_nums())
```

- Clase

  Sintaxa pentru definirea unei clase este următoarea:

```
class Greeter:

    # Constructor
    def __init__(self, name):
        self.name = name  # Crearea unei instante

    # Metoda
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)

g = Greeter('Fred')  # Construieste un obiect de tipul Greeter
g.greet()
# Apeleaza metoda greet(); Afiseaza "Hello, Fred"
g.greet(loud=True)
# Apeleaza metoda greet(); Afiseaza "HELLO, FRED!"
```

În definirea unei clase, fiecare metodă va avea ca prim argument *self*, iar accesarea atributelor și metodelor în interiorul clasei se va face cu self.atribut / self.metoda().

Nu există specificatori de access!

- Exerciții

1. Se dau următoarele etichete prezise de un clasificator binar, y_pred = [1, 1, 1, 0, 1, 0, 1, 1, 0, 0] și etichetele y_true = [1, 0, 1, 0, 1, 0, 1, 0, 1, 0].
   a. Definiți metoda *accuracy_score(y_true, y_pred)* care să calculeze acuratețea clasificatorului binar.
      Obs:
      - Acuratețe = $\frac{\sum_{i=1}^{n} \quad y\_pred_i == y\_true_i}{n}$

b. Definiți metoda *precision_recall_score(y_true, y_pred)* care returnează precizia și recall-ul clasificatorului binar.
Obs:
   - 0 - negativ, 1 - pozitiv
   - Precizie = $\frac{tp}{tp + fp}$
   - Recall = $\frac{tp}{tp + fn}$
   - tp = true positive, numărul etichetelor prezise ca fiind pozitive și care au fost clasificate corect
   - fp = false positive, numărul etichetelor prezise ca fiind pozitive, dar care sunt de fapt negative
   - fn = false negative, numărul etichetelor prezise ca fiind negative, dar care sunt de fapt pozitive

c. Definiți metoda *mse(y_true, y_pred)* (mean square error) care calculează media pătratelor erorilor de clasificare.
Obs:
   - MSE = $\frac{\sum_{i=1}^{n} (y\_pred_i - y\_true_i)^2}{n}$

d. Definiți metoda *mae(y_true, y_pred)* (mean absolute error) care calculează media erorii absolute de clasificare.
Obs:
   - MAE = $\frac{\sum_{i=1}^{n} |y\_pred_i - y\_true_i|}{n}$