

## TUTORIAT 4 si 5

Linkuri utile:

- <https://www.python.org/>
- <https://docs.python.org/3/>
- <https://www.codecademy.com/learn/learn-python-3>
- <https://www.learnpython.org/>
- <https://stanfordpython.com/#/>

Comentariu multiplu PyCharm: CTRL + /

Ce conține tutoriatul ?

- I. Sortări
- II. Tupluri (**clasa tuple**)
  1. Creare
  2. Accesare elemente și subsecvențe
  3. Metode uzuale
  4. Împachetare și despachetare
- III. Exerciții

### I. Sortări

Diferența între metodele sorted și sort

	SORTED	SORT
APLICABILITATE	comună secvențelor	specifică listelor
SINTAXĂ	<code>sorted(nume_iterabil, key=key, reverse=boolean)</code>	<code>sort(key=key, reverse=boolean)</code>
APEL	<code>sorted(nume_iterabil, key=None, reverse=False)</code>	<code>nume_lista.sort(key=None, reverse=False)</code>
EFFECT	returnează listă nouă cu elementele sortate din <code>nume_iterabil</code> (!NU MODIFICĂ <code>nume_iterabil</code> )	<b>modifică</b> <code>nume_lista</code> (elementele în ordinea sortării)

Valorie parametrilor

- `nume_iterabil` -> necesar (secvența de sortat)
- `key` -> opțional (o funcție, cu un singur argument, care decide ordinea de sortare, valoarea default este None)
- `reverse` -> opțional ( boolean, *False* pentru sortare crescătoare, *True* pentru sortare descrescătoare, valoarea default False)

Exemple:

```
stringuri = [ "g", "b", "h", "a", "z", "m", "r"]  
sorted_stringuri = sorted(stringuri)  
print(sorted_stringuri) => ["a", "b", "g", "h", "m", "r", "z"]
```

```
sir = "caine"  
print(sorted(sir)) => ["a", "c", "e", "i", "n"]
```

```
numere = [1, 6, -5, 4, 8, 15]  
numere.sort()  
print(numere) => [-5, 1, 4, 6, 8, 15]
```

```
my_list = [1, 6, -5, 4, 8, 15]  
sorted_my_list = sorted(my_list, reverse=True)  
print(sorted_my_list) => [15, 8, 6, 4, 1, -5]
```

```
cuvinte = [ "xilofon", "pian", "chitara", "toba", "vioara"]  
cuvinte_sortat = sorted(cuvinte)  
print(cuvinte_sortat) => ["chitara", "pian", "toba", "vioara", "xilofon" ]  
cuvinte_sortat_lungime = sorted(cuvinte, key=len)  
print(cuvinte_sortat_lungime) => ["pian", "toba", "vioara", "xilofon", "chitara"]
```

**!ATENȚIE!** Elementele egale își păstrează ordinea inițială (sortare stabilă)

## Definire funcții

Sintaxă: def nume\_funcție(parametrii):

    #instrucțiuni



Exemplu:

# sortăm cuvintele după penultima literă

def cheie\_sortare (cuvant):

    return cuvant[-2]

cuvinte = [ "xilofon", "pian", "chitara", "toba", "vioara"]

cuvinte\_sortate = sorted(cuvinte, key=cheie\_sortare)

print(cuvinte\_sortate) => ["pian", "toba", "xilofon", "chitara", "vioara"]

## Modulul **FUNCTOOLS**

- ➔ utilizat pentru lucrul cu funcții de nivel înalt (funcție care returnează o funcție sau funcție care are ca argument o altă funcție)
- ➔ conține 11 funcții cu funcționalități diferite în funcție de versiunea python
- ➔ printre aceste funcții se numără: `reduce()`, `total_ordering()`, **`cmp_to_key()`**

### Funcția **cmp\_to\_key**

- ➔ transformă o funcție de comparație de stil vechi ( stil C/C++) într-o funcție cheie (key)

( o funcție de comparație este reprezentată de orice funcție care acceptă două argumente, le compară și returnează un număr negativ pentru < , zero pentru = sau un număr pozitiv pentru > ; o funcție cheie acceptă un argument și returnează o altă valoare pentru a fi folosită drept cheie de sortare)

- ➔ utilizată cu instrumente care acceptă funcții cheie, de exemplu: `min()`, `max()`, `sorted()`
- ➔ acceptă un singur argument (o funcție, în general un apelabil)
- ➔ returnează o cheie specială, folosită pentru a compara elemente
- ➔ pentru a fi folosită trebuie importat modulul `functools` (`import functools`)
- ➔ apelare: `functools.cmp_to_key(numie_funcție_comparare)`
- ➔ Cum funcționează `cmp_to_key` ? [https://www.geeksforgeeks.org/how-does-the-functools-cmp\\_to\\_key-function-works-in-python/](https://www.geeksforgeeks.org/how-does-the-functools-cmp_to_key-function-works-in-python/), <https://newbedev.com/python-how-does-the-functools-cmp-to-key-function-works>

### Funcții **lambda**

- ➔ funcții **anonime** (nu este necesar să aibă nume, **simple** și rapide)
- ➔ acceptă orice număr de argumente
- ➔ au o **singură expresie**
- ➔ este executată expresia și este returnat rezultatul
- ➔ rezultatul returnat este de tip **funcție (class function)**
- ➔ **sintaxa: lambda argumente: expresie**

Exemple: `f = lambda a: a ** 2`

`print(f(4)); print(type(f)) => 16, <class function>`

`functie = lambda a,b: a if a < b else b`

`print(functie(4, 2)) => 2`

## II. Clasa **TUPLE**

- ➔ **IMUTABILE** (nu li se poate modifica valoarea după creare)
- ➔ conțin elemente de tipuri diferite (recomandabil, NU)
- ➔ elementele sunt indexate de la **0**
- ➔ sunt permise valorile duplicate
- ➔ pot fi aplicate metodele comune pentru secvențe (doar cele care nu modifică valoarea)
- ➔ folosite pentru: returnare valori multiple, împachetare/despachetare, chei în dicționare, atribuire multiplă

### II.1. Creare

Metoda de creare	Exemplu
cu ajutorul constructorului <b>tuple()</b>	<code>nume_tuplu = tuple()</code>
enumerarea <b>elementelor între ()</b>	<code>nume_tuplu = (1, 14, -5, 7, 100)</code>
specificarea unui <b>iterabil în constructor</b>	<code>nume_tuplu = tuple("tutoriat")</code> <code>nume_tuplu = tuple(range(10))</code>
enumerare <b>elemente separate prin virgulă</b>	<code>nume_tuplu = 2, 5, 10, 14</code>

**!ATENȚIE!** `my_tuple = ()` -> tuplu vid

`my_tuple = (6)` **NU** este tuplu cu un element, ci este un obiect din clasa `int`

`my_tuple = (6,)` **tuplu cu un element**

Tuplurile pot avea elemente de tip tuplu. Acestea poartă denumirea de **tupluri imbricate**.

**Exemplu:**

`my_tuple = (4, 12, (2, 60), (10, "afara"))`

**!ATENȚIE!** Nu se poate utiliza metoda **comprehensiunii** (obiectele construite astfel sunt de tip generator).

**Exemplu:**

`my_tuple = (i ** 2 for i in range(6))`

`print(type(my_tuple), my_tuple)`



## II.2. Accesare elemente și subsecvențe

Pentru a accesa elementele unui tuplu se utilizează [ ]. **!ATENȚIE!** Indexarea începe de la 0.

### Exemple:

`my_tuple[i]` => elementul de pe poziția i din tuplul `my_tuple`, (începând cu poziția 0)

**!ATENȚIE!** i poate fi și negativ.

Tip accesare	Rezultat
<code>nume_tuplu [i]</code>	elementul de pe poziția i
<code>nume_tuplu[i:j]</code>	elementele de pe pozițiile i, i+1,...,j-1
<code>nume_tuplu[ :j]</code>	elementele de pe pozițiile 0,1,...,j-1
<code>nume_tuplu [i: ]</code>	elementele de pe pozițiile 0,1,...,len(nume_sir)-1
<code>nume_tuplu[:]</code>	toată secvența
<code>nume_tuplu[i:j:k]</code>	elementele de pe pozițiile i,i+k,i+2k,....

### Modificări în tupluri

➔ dacă elementele unui tuplu sunt de tip mutabil, atunci valoarea lor poate fi schimbată

Exemplu:

```
my_tuple = ([2, 5, 8], [10, 5], 4, "afara" )
```

```
my_tuple[0][2] = 7
```

```
my_tuple[1].extend(range(2))
```

```
my_tuple[0].append(12)
```

```
print(my_tuple)
```

```
my_tuple[3] = "ana" => EROARE tuple object does not support item assignment
```

```
my_tuple[2] = 4 => EROARE tuple object does not support item assignment
```

## II.3. Metode uzuale

**Concatenări:** +, \*

Exemplu: `tuplu1 = (1, 5, 9, 13)`

`tuplu2 = (2, 7, 11)`



```
tuplu1 = tuplu1 + tuplu2 => tuplu1 = (1, 5, 9, 13, 2, 7, 11)
```

**!ATENȚIE!** se creează un obiect nou, nu adaugă la tuplul tuplu1

```
tuplu = (0, ); n = 4
```

```
tuplu * n sau n * tuplu => (0, 0, 0, 0)
```

**Funcții comune** tuturor secvențelor: **len**(nume\_tuplu) => returnează numărul de elemente al tuplului, **min**(nume\_tuplu) => returnează valoarea celui mai mic element din tuplu, **max**(nume\_tuplu) => returnează valoarea celui mai mare element din tuplu

**Apartenența: in, not in**

Exemplu: my\_tuple = (1, 5, 9, 13)

```
if 14 not in my_tuple:
```

```
    print("10 nu se gaseste in tuplu")
```

**Metode** ( funcție a unei clasei)

Metoda	Apel	Efect
count	tuplu.count(element)	returnează numărul de apariții a lui element în tuplu
index	tuplu.index(element)	returnează indexul primei apariții a lui element în tuplu
sorted(nume_tuplu, key=None, reverse=False)	sorted(nume_tuplu, key=None, reverse=False)	sortează tuplul în ordine crescătoare( dacă reverse = True, sortează lista în ordine descrescătoare)

## II.4. Împachetare și despachetare

**Exemple:**

```
(a, b) = (10, 20)      a, b = b, a # ATENȚIE SWAP
```

```
print(a, b)            print(a, b)
```

```
my_tuple = (4, 5, 8, 9)
```

```
x, y, z, t = my_tuple
```

```
print(x, y, z, t)
```

```
def functie_multipla():
```

```
    x = 14
```

```
    y = "afara"
```

```
c, d, *e = 4, 10, 25, 45, 8
```

```
print(c, d, e)
```

```
# e este de tip listă
```



return x, y

### **III.Exerciții**

#### **A.Sortări**

1. Se citesc de la tastatură  $n$ , număr natural, și  $n$  elemente, numere naturale. Să se ordoneze crescător după numărul de cifre. Exemplu:  $n = 7$ ,  $lista = [4, 89, 45, 125, 9, 14789, 10] \Rightarrow [4, 9, 89, 45, 10, 125, 14789]$
2. Se citesc de la tastatură  $n$ , număr natural, și  $n$  elemente, numere naturale. Să se ordoneze descrescător după numărul de cifre pare. Exemplu:  $n = 5$ ,  $my\_list = [20, 5, 123, 11, 486] \Rightarrow [486, 20, 123, 5, 11]$
3. Se consideră o listă de numere naturale cu  $n$  elemente. Să se sorteze lista după ultima cifră, folosind o funcție lambda. Exemplu:  $n = 4$ ,  $new\_list = [15, 89, 41, 12] \Rightarrow [41, 12, 15, 89]$
4. Pentru  $n$ , număr natural, discipline școlare se citesc informațiile: cod disciplină (număr natural de cel mult 4 cifre, pot exista discipline cu același cod), nume disciplină (nume format dintr-un singur cuvânt), număr ore pe săptămână (informațiile despre o disciplină se găsesc pe o linie).
  - a) Să se memoreze o listă de tupluri cu informațiile despre cele  $n$  discipline;
  - b) Să se afișeze disciplinele sortate după cod și în caz de egalitate după nume;
  - c) Să se afișeze disciplinele sortate descrescător după numărul de ore și în caz de egalitate după nume.

#### **B.Tupluri**

5. Se dau două tupluri cu 3 elemente. Să se interschimbe valorile celor două tupluri. Exemplu:  $tuplu1 = (10, 20, 30)$   $tuplu2 = (40, 50, 60) \Rightarrow tuplu1 = (40, 50, 60)$ ,  $tuplu2 = (10, 20, 30)$
6. Să se sorteze un tuplu de tupluri dat după prima componentă, apoi după cea de-a doua (utilizând o funcție lambda).

Exemplu:  $tuple1 = ("a", 100), ("z", 14), ("c", 20) \Rightarrow ("a", 100), ("c", 20), ("z", 14)$

7. <https://pynative.com/python-tuple-quiz/>