

Liste

Liste

Alte operații

- inversare:

`ls.reverse()` - **in place (modifică lista)**

```
ls = [3,5,7]
```

```
ls.reverse()
```

```
print(ls)
```



Liste

Sortare

- Metodă comună secvențelor:

```
sorted(iterabil, key=None, reverse=False)
```

returnează o listă nouă obținută prin sortarea elementelor lui `iterabil`, **nu modifică** `iterabil`

- Metodă specifică: – **modifică lista**

```
ls.sort(key=None, reverse=False)
```



Sortarea unei secvențe/liste

- ▶ `sorted(iterable, key=None, reverse=False)`
- ▶ `ls.sort(key=None, reverse=False)`

Parametri cu nume (numiți)

- **key** – funcție (cu un argument) care dă **cheia** de comparare
 - **reverse** – **True/False** – ordonat descrescător/crescător
- **sortare stabilă**

Sortare

```
print(sorted([4,1,7,3,6]))
```

```
s = "Programarea"
```

```
sorted(s)
```

```
print(s) #??
```

Sortare

```
print(sorted([4,1,7,3,6]))
```

```
s = "Programarea"
```

```
sorted(s)
```

```
print(s) #??
```

```
s = sorted(s)
```

```
print(s) #?
```

Sortare

```
ls = ["Vom", "sorta", "aceasta", "lista", "de", "siruri"]
```

```
ls_sorted = sorted(ls)
```

```
print(ls)
```

```
print(ls_sorted)
```

```
ls.sort()
```

```
print(ls)
```



Sortare

```
ls = ["Vom", "sorta", "aceasta", "lista", "de", "siruri"]
```

```
ls_sorted = sorted(ls, key = len, reverse = True)
```

```
#sortare stabila
```

```
print(ls_sorted)
```



Sortare

```
ls = ["Vom", "sorta", "aceasta", "lista", "de", "siruri"]
```

```
ls_sorted = sorted(ls, key = str.lower)
```

```
print(ls_sorted)
```



Sortare

```
ls = ["Vom", "sorta", "aceasta", "lista", "de", "siruri"]
```

```
def cheie(s):  
    return s[-1]
```

```
ls_sorted = sorted(ls, key = cheie) #sortare stabila  
print(ls_sorted)
```

Sortare

`functools.cmp_to_key()` pentru a converti un criteriu de comparare (stil C) în key

Sortare

```
import functools

def comp_perechi(p,q):
    if p[1] < q[1]:
        return -1
    if p[1] == q[1]:
        return p[0]-q[0]
    return 1

ls = [(3,4), (5,3), (3,1), (1,6), (2,5)]
```

Sortare

```
import functools

def comp_perechi(p,q):
    if p[1] < q[1]:
        return -1
    if p[1] == q[1]:
        return p[0]-q[0]
    return 1

ls = [(3,4), (5,3), (3,1), (1,6), (2,5)]
ls_sorted=sorted(ls, key =
                    functools.cmp_to_key(comp_perechi))
print(ls_sorted)
```

Sortare

```
ls = [(3,4), (5,3), (3,1), (1,6), (2,4)]
```

```
ls_sorted = sorted(ls, key=lambda x: x[1])  
print(ls_sorted)
```

```
ls_sorted = sorted(ls, key=lambda x: (x[1],x[0]))  
print(ls_sorted)
```

Tupluri

Tupluri

- Clasa `tuple`
- Imutabil
- pot fi eterogene
- Suporta operațiile comune (!care nu modifică valoarea)

Tupluri

- Creare – ca la liste, dar cu ()

```
t = ()
```

```
t = (1, 2, 3)
```

```
t = 1, 2, 3
```

Tupluri

- Creare – ca la liste, dar cu ()

```
t = (2) #NU
```

```
print(type(t))
```

```
t = (2,) #DA
```

```
print(type(t))
```

Tupluri

- Creare – ca la liste, dar cu ()

`#Tupluri imbricate`

`t = ((1, "A"), (2, "B"), 3)`

Tupluri

- Creare – folosind tuple()

```
t = tuple("abc")
```

```
t = tuple(range(3))
```

Tupluri

- Ce nu se poate modifica la un tuplu?

```
t = (2,3)
```

```
t[0] = 7
```

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

Tupluri

- Ce nu se poate modifica la un tuplu?

```
t=(1,2,[5,6])
```

```
#t[1]=11 #NU
```

```
t[2][0]=15 #DA
```

```
t[2].append(7) #DA
```

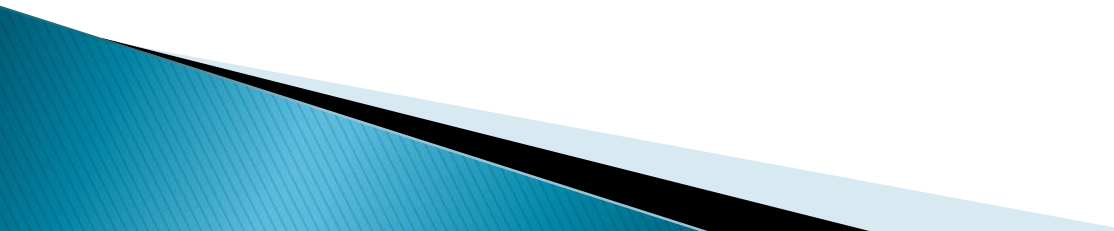
```
print(t)
```



Tupluri

- len, min, max
- count, index
- Operatorii in, not in

< , <=, >, >= , ==, !=

- Accesare, feliere, indecși negativi
 - Concatenare +, *
 - sorted
- 

Tupluri

```
t = (2,4,16,8,10)
```

```
print(t[0],t[-1],t[1:3],t[-2:])
```

```
for x in t:
```

```
    print(x)
```



Tupluri

```
t = (2,4,16,8,10)
```

```
print(t[0],t[-1],t[1:3],t[-2:])
```

```
if 0 not in t:
```

```
    t = t + (0,) #!!!nu +(0)
```

```
    print(t)
```



Tupluri

```
t = (2,4,16,8,10)
```

```
print(t[0],t[-1],t[1:3],t[-2:])
```

```
if 0 not in t:
```

```
    t = t + (0,) #!!!nu +(0)
```

```
    print(t)
```

```
print(t.count(8), t.index(8))
```

```
print(sorted(t))
```



Tupluri

Particularități

- ▶ Tuplu cu un element

`t = (0) * 8`

`t = (0,) * 8`

Tupluri

Particularități

- ▶ Nu comprehensiune:

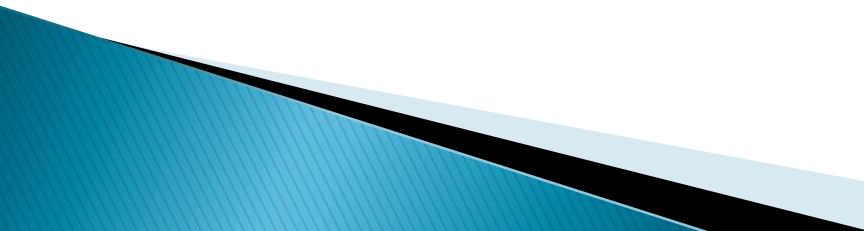
`(expresie for x in iterabil)` este **generator**

```
t = (i*i for i in range(10))
```

```
print(t, type(t))
```

```
print(sum(t)) #nu lista suplimentara
```

```
print(sum(t))
```



Tupluri

Particularități

- ▶ Nu comprehensiune:

`(expresie for x in iterabil)` este **generator**

```
l = [1,0,3,0,0,4]
```

```
l[:] = [x for x in l if x!=0]
```

```
l[:] = (x for x in l if x!=0) #nu lista suplimentara
```

Tupluri

- ▶ Împachetare/despachetare

```
(x,y) = (1,2)
```

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

```
print(x,y)
```

```
x, y = y, x
```

```
print(x,y)
```

Tupluri

- ▶ Împachetare/despachetare

```
t = 1,2,3
```

```
print(type(t))
```

```
a,b,c = t
```

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

Tupluri

- ▶ Împachetare/despachetare

```
def f():
```

```
    x = 1
```

```
    y = [10,20]
```

```
    return x,y
```


Tupluri

- ▶ Împachetare/despachetare

```
def f():
```

```
    x = 1
```

```
    y = [10,20]
```

```
    return x,y
```

```
r = f()
```

```
print(f"rezultat {r} de tip {type(r)}")
```

Tupluri

- ▶ Împachetare/despachetare

```
def f():
```

```
    x = 1
```

```
    y = [10,20]
```

```
    return x,y
```

```
r = f()
```

```
print(f"rezultat {r} de tip {type(r)}")
```

```
x, ls = f()
```

```
print(f"rezultate {x} de tip {type(x)} si {ls}  
de tip {type(ls)}")
```

Tupluri

- ▶ Împachetare/despachetare

```
a, *b, c = 1, 2, 3, 5
```

```
print(a,b,c,type(b))
```

Tupluri

Utilitate

- Atribuire multiplă
- Returnare valori multiple
- Pack/unpack
- Chei în dicționare
- Există tupluri cu nume

