


Tema 6. Seturi și dicționare în Python



Ce ne așteaptă?

1. Seturi în Python

2. Dicționare în Python

1. Care sunt caracteristicile unui set în Python?
2. Cum se creează un set?
3. Cum se adaugă elemente într-un set?
4. Cum șterg elemente din set?
5. Care sunt metodele de interacțiune între 2 seturi?
6. Cum se elimină dublurile cu ajutorul unui set?
7. Care sunt caracteristicile unui dicționar în Python?
8. Cum se creează dicționarele?
9. Cum se accesează elementele unui dicționar?
10. Cum se adaugă și se actualizează elementele unui dicționar?
11. Cum se șterg datele unui dicționar?
12. Cum se accesează cheile, valorile și perechile unui dicționar?
13. Cum se creează dicționare prin procedura “comprehensions”?

1. Seturi în Python

Particularitățile seturilor în Python

- **Set – o colecție neordonată de obiecte unice**
- **Un set este definit de următoarele caracteristici:**
 - este heterogen – poate conține elemente de același tip sau de tip diferit
 - este neordonat – ordinea în care sunt incluse elementele nu va fi și ordinea de afișare a acestora
 - este mutabil – permite modificarea conținutului după crearea acestuia
 - nu permite prezența repetată a obiectelor
 - nu permite apelarea obiectelor folosind indexul sau operatorul slicing
 - elementele setului se includ în cadrul parantezelor acolade “{ }” separate prin virgulă

Crearea unui set

- Crearea unui set cu elemente utilizând parantezele acolade

```
s = {10,20,30,40}  
print(s)  
print(type(s))
```

- Crearea unui set gol utilizând funcția `set()`

```
s=set()  
print(s)  
print(type(s))
```

- Crearea unui set cu elemente utilizând funcția `set()`

```
t=(20, 40, 30 ,20, 50, 20, 10)  
s=set(t)  
print(s)  
print(type(s))
```

Metode de adăugare a elementelor în set

- **Metoda add()** – permite adăugarea unui element

```
s={10,20,30}  
s.add(40)  
print(s)
```

- **Metoda update()** – permite adăugarea mai multor elemente din una sau mai multe secvențe

- adăugarea mai multor elemente dintr-o singură secvență

```
s = {10,20,30}  
l = [40,50,60,10]  
s.update(l)  
print(s)
```

- adăugarea mai multor elemente din mai multe secvențe

```
s = {10,20,30}  
l = [40,50,60,10]  
st = "Pavel"  
s.update(l, st)  
print(s)
```

- **Metoda copy()** – returnează o copie a setului

```
s={10,20,30}  
s1=s.copy()  
print(s1)
```

Metode de ștergere a elementelor din set

- Metoda `pop()` - șterge un element aliator și returnează acel element

```
s = {40,10,30,20}  
print(s.pop())  
print(s)
```

- Metoda `remove()` - șterge elementul specificat, dacă acesta lipsește – `KeyError`

```
s={40,10,30,20}  
s.remove(30)  
print(s)  
s.remove(50)
```

- Metoda `discard()` - șterge elementul specificat, dacă acesta lipsește nu returnează eroare

```
s={40,10,30,20}  
s.discard(30)  
print(s)  
s.discard(50)
```

- Metoda `clear()` - șterge toate elementele setului

```
s={10,20,30}  
s.clear()  
print(s)
```

Metode de interacțiune între 2 seturi (1)

- Metoda `union()` returnează elementele ambelor seturi

```
x={10,20,30,40}  
y={30,40,50,60}  
print(x.union(y))
```

- Metoda `intersection()` returnează doar elementele comune

```
x = {10,20,30,40}  
y = {30,40,50,60}  
print(x.intersection(y))
```


Metode de interacțiune între 2 seturi (2)

- Metoda `difference()` returnează elementele dintr-un set neprezente în celălalt

```
x={10,20,30,40}
y={30,40,50,60}
print(x.difference(y))
print(y.difference(x))
```

- Metoda `symmetric_difference()` returnează elementele care nu sunt comune ambelor seturi

```
x={10,20,30,40}
y={30,40,50,60}
print(x.symmetric_difference(y))
```

Operatori de apartenență și set comprehensions

- Operatorii *in* și *not in* permit verificare prezenței sau lipsei elementului în set

```
s = {10, 20, 30, "Pavel"}  
print(s)  
print(10 in s)  
print(50 in s)  
print(2 not in s)  
print('Pavel' not in s)
```

- Set comprehensions – o cale simplificată de creare a unui set prin iterația unei secvențe

```
x = [1,2,3,4,5,6]  
y = {i*i for i in x}  
print(y)  
print(type(y))
```

Eliminarea dublurilor și frozen set

- **Eliminarea dublurii elementelor unei liste**

```
l=[10,20,30,10,20,40]
print(l)
s=set(l)
print(s)
l=list(s)
print(l)
```

- **Frozen set – un set imutabil creat dintr-o secvență utilizând funcția frozenset()**

```
vocale = ('a', 'e', 'i', 'o', 'u', 'î', 'â', 'ă')
fs = frozenset(vocale)
print(fs)
print(type(fs))
```

2. Dicționare în Python

Particularitățile dicționarelor în Python

- **Dictionar** – o colecție de elemente de tip cheie-valoarea
- **Un dicționar este definit de următoarele caracteristici:**
 - conține date în forma perechilor chei-valori
 - cheia și valorile se separă prin simbolul două puncte “:”
 - perechile se includ în cadrul parantezelor acolade “{ }” separate prin virgulă
 - nu se admite dublarea cheilor însă se permite repetarea valorilor
 - atât cheile cât și valorile sunt heterogene, adică pot fi de diferite tipuri
 - este mutabil – permite modificare conținutului după crearea acestuia
 - este dinamic – lungimea acestuia poate fi modificată după creare
 - este neordonat – ordinea în care sunt incluse perechile nu va fi și ordinea de afișare a acestora
 - nu permite apelarea obiectelor folosind indexul sau operatorul de feliere

Crearea dicționarilor folosind paranteze acolade

- Crearea dicționarilor folosind sintaxa

nume_dict = {cheie1:valoare1, cheie2:valoare2, ...}

```
d = {"nume": "Nicolaev", "prenume": "Pavel", "varsta": 32, "job": "conf"}  
print(d)  
print(type(d))
```

- Crearea unui dicționar gol utilizând parantezele acolade

```
d = {}  
print(d)  
print(type(d))
```

- Crearea unui dicționar prin adăugarea elementelor în dicționarul gol

```
d = {}  
d['nume'] = "Nicolaev"  
d['prenume'] = "Pavel"  
print(d)  
print(type(d))
```

Crearea dicționarilor folosind funcția *dict()*

- Crearea unui dicționar gol utilizând funcția *dict()*

```
d = dict()  
print(d)  
print(type(d))
```

- Crearea unui dicționar cu elemente utilizând sintaza:

nume_dict = dict([tuple1, tuple2 ...])

- tuple va avea doar 2 elemente – primul cheie, al doilea valoarea

```
t1 = ("nume", "Nicolaev")  
t2 = ("prenume", "Pavel")  
d = dict([t1, t2])  
print(d)  
print(type(d))
```

Accesarea datelor și actualizarea lor cu ajutorul cheii

- **Accesarea datelor prin intermediul cheilor**

```
d = {"nume": "Nicolaev", "prenume": "Pavel", "varsta": 32, "job": "conf"}  
print(d["nume"])  
print(d["varsta"])  
print(d["ocupatie"])
```

- **Actualizarea datelor**

```
d = {"nume": "Nicolaev", "prenume": "Pavel", "varsta": 32}  
print(d)  
d["prenume"] = "Ion"  
d["varsta"] = 25  
print(d)
```

- **Adaugare valori**

```
d = {"nume": "Nicolaev", "prenume": "Pavel", "varsta": 32}  
print(d)  
d["prenume"] = "Ion"  
d["job"] = "conf"  
print(d)
```

Metode de accesare a datelor

- Metoda `get()` cu sintaxa: **`nume_dict.get(cheie)`** – reîntoarce valoarea corespunzătoare cheii specificate. Dacă cheia lipsește reîntoarce **`None`**

```
d={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32}
print(d.get("varsta"))
print(d.get("job"))
```

- Metoda `get()` cu sintaxa: **`nume_dict.get(cheie, val_implic)`** – reîntoarce valoarea conform cheii specificate. Dacă cheia lipsește reîntoarce **`val_implic`**

```
d={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32}
print(d.get("varsta", "cheie greșită"))
print(d.get("job", "cheie greșită"))
```

- Funcția `len()` – reîntoarce numărul perechilor dicționarului

```
d={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32, "job":"conf"}
print(len(d))
```

- Metoda `copy()` – realizează o copie a dicționarului (clonare)

```
d1={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32, "job":"conf"}
d2 = d1.copy()
print(d2)
```


Ștergerea datelor dicționarului

- Ștergerea unei perechi utilizând cuvântul cheie *del* cu sintaza: ***del nume_dict[cheie]***

```
d={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32}
print(d)
del d["varsta"]
print(d)
del d["job"]
```

- Ștergerea tuturor perechilor utilizând metoda *clear()*

```
d={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32}
print(d)
d.clear()
print(d)
```

- Ștergerea întregului dicționar sintaza: ***del nume_dict***

```
d={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32}
print(d)
del d
print(d)
```

Metode de ștergere a datelor

- Metoda `pop()` – șterge valoarea asociată cheii și returnează această valoare. Dacă cheia lipsește `KeyError`

```
d={"nume":"Nicolae", "prenume":"Pavel", "varsta":32, "job":"conf"}
print(d)
print(d.pop("nume"))
print(d)
print(d.pop("ocupatie"))
```

- Metoda `popitem()` – șterge o pereche aleatoare și returnează valoarea acestuia sub formă de tuple. Dacă dicționarul gol `KeyError`

```
d={"nume":"Nicolae", "prenume":"Pavel", "varsta":32, "job":"conf"}
print(d)
print(d.popitem())
print(d)
d.clear()
d.popitem()
```

Metode de accesare a cheilor, valorilor și perechilor

- **Metoda keys()** – returnează toate cheile asociate dicționarului

```
d={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32, "job":"conf"}
keys = d.keys()
for k in keys:
    print(k)
```

- **Metoda values()** – returnează toate valorile asociate dicționarului

```
d={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32, "job":"conf"}
values = d.values()
for v in values:
    print(v)
```

- **Metoda items()** – returnează un tip de date cu tupluri ce conțin fiecare cheie și valoarea asociată

```
d={"nume":"Nicolaev", "prenume":"Pavel", "varsta":32, "job":"conf"}
items = d.items()
for k, v in items:
    print(k, "---", v)
```

Dictionary Comprehensions

- Dictionary comprehensions – o cale simplificată de creare a unui dicționar prin iterația unei secvențe

```
x = [1,2,3,4,5,6]
y = {i:3*i for i in x}
print(y)
print(type(y))
```