TUTORIAT 6

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. Dicționare (**clasa dict**)
   1. Creare
   2. Accesare elemente
   3. Metode uzuale

II. Mulțimi (clasa **set**)
   1. Creare
   2. Accesare elemente
   3. Metode uzuale
   4. Clasa **frozenset**

III. Exerciții

I.Clasa **DICT**

- ➔ **MUTABILE** (li se poate modifica valoarea după creare)
- ➔ colecție de **perechi cheie:valoare**
- ➔ **cheia este IMUTABILĂ** (toate componentele cheii trebuie să fie imutabile) **și UNICĂ**
- ➔ nu sunt permise valorile duplicate
- ➔ elementele sunt indexate după cheie

I.1. **Creare**

Metoda de creare	Exemplu
cu ajutorul constructorului **dict()**	nume_dictionar = dict()
enumerarea **perechilor cheie:valoare între {}**	nume_dictionar = {cheie1:valoare1, cheie2:valoare2,..., cheien:valoaren}
cu ajutorul fromkey(dict.fromkeys(iterabil_chei, valoare)) implicit valoare = None	nume_dictionar = dict.fromkeys(range(4), 0)

comprehensiune	
{cheie: valoare for cheie in interabil}	nume_dictionar = {x: 1 for x in "abcde"}

!ATENȚIE! my_dict = {} -> dicționar vid

my_dict = {[2,3] = 2, 1:5} -> EROARE, cheile sunt IMUTABILE

Exemple:

dict1 = {"b":2, "c":3, "d":4}

dict2 = {1:[3,4,5], 2:[1], 3:{1,2,3}}

dict3 = dict([("b", 2), ("c",3), ("d", 4)])

dict4 = dict.fromkeys(range(4), 0)

II.2.Accesare elemente

Pentru a accesa elementele unui dicționar se utilizează structura de forma dictionar[cheie] (**EROARE** dacă nu există cheia)

dictionar.get(cheie, valoare_default) -> returnează valoarea asociată cheii, dacă nu există cheie returnează None sau valoarea default dacă a fost specificată

Exemple:

dict1 = {"b":2, "c":3, "d":4}

print(dict1["c"]) => 3

print(dict1["f"]) => EROARE

print(dict1.get["f"]) => None

print(dict1.get("f", 0)) => 0

II.3. Metode uzuale

Funcții comune : **len**(nume_dictionar) => returnează numărul de perechi cheie-valoare din dicționar, **min**(nume_dictionar) => returnează valoarea celei mai mici chei din dictionar, **max**(nume_dictionar) => returnează valoarea celei mai mari chei din dicționar.

Apartenența: in, not in (după chei)

Exemplu: my_dict = {1:[3,4,5], 2:[1], 3:{1,2,3}}

```
if 4 not in my_dict:
    print("4 nu se gaseste in dictionar")
```

Metode (funcție a unei clasei)

Metoda	Apel	Efect
setdefault(cheie, valoare)	dict1.setdefault(cheie,valoare)	returnează valoarea cheii existente dacă exista deja în dicționar sau cheii noi adăugate cu valoarea None sau cu cea speficicată
update	dict1.update(dictionar) dict1.update(iterabil cheie-valoare)	reuniune de dicționare, cheile deja existente sunt actualizate
pop	dict1.pop(cheie, valoare)	returnează valoarea ascociată cheii și elimină cheie din dicționar(EROARE dacă cheia nu există sau returnează valoare primită ca al doilea parametru)
del	del dict1 [cheie]	elimină cheie și valoarea asociată din dicționar(EROARE dacă nu există cheia)
clear	dict1.clear()	elimină toate perechile cheie-valoare din dicționar

Lucrul cu dicționare

val = dict1[cheie]	**atribuie valoarea asociata cheii din dictionar variabilei val**
dict1[cheie] = valoare	**atribuie cheii valoarea corepunzătoare(dacă cheie nu există o inserează)**
dict1.get(cheie, default)	**returnează valoarea asociată cheii(dacă cheie nu există în dicționar, returnează valoarea default)**
dict1.keys()	**returnează o colecție a cheilor din dicționar**
dict1.values()	**returnează o colecție a volorilor asociate cheilor din dicționar**
dict1.items()	**returnează o colecție de tupluri cheie-valoare**

Exemple:

```
dict1 = {"b":2, "c":3, "d":4}
dict1.update({"f":4, "d":5})
print(dict1) => {"b":2, "c":3, "d":4, "f":4, "d":5}
dict1.pop("D") => EROARE
```

```
val = dict1.pop("d")
print(dict, val) => {"b":2, "c":3, "f":4, "d":5} None
val = dict1.pop("D", 0)
print(dict1, val) {"b":2, "c":3, "f":4, "d":5} 0
del dict1["d"] => EROARE


my_dict = {"b":2, "c":-3, "d":4}
print(my_dict.keys(), type(my_dict.keys()))
print(my_dict.values(), type(my_dict.values()))
print(my_dict.items(), type(my_dict.items()))
ls = my_dict.keys()
print(ls, type(ls))
my_dict.update({"b":10})
print(ls)
```

II. Clasa **SET**

- ➔ **MUTABILE** (li se poate modifica valoarea după creare)
- ➔ elementele sunt **imutabile, unice și neindexate**
- ➔ **neordonate** (nu se păstrează ordinea de introducere a elementelor)

II.1. **Creare**

Metoda de creare	Exemplu
cu ajutorul constructorului **set()**	nume_multime = set()
enumerarea **elementelor între {}**	nume_multime = {20, 30, 40, 50}
iterabil în constructorul set(iterabil)	nume_multime = set("tutoriat")
comprehensiune **{expresie for nume_varianila in iterabil}** **{expresie for nume_varianila in iterabil if conditie}** **{ expresie1 if conditie else expresie2 for nume_variabila in iterabil}**	nume_multime = {a**2 for a in range(2,5)} nume_multime = {a for a in range(1, 20) if a % 2 == 0} nume_multime = {a if a > 0 else –a for a in range(-5, 4)}

!ATENȚIE!

- **multime = { }** **NU** este mulțimea vidă, ci dicționar, **multime = set()** mulțimea vidă
- **NU** multime[i] (elementele **NU** sunt indexate)
- **NU** se poate utiliza **felierea**, **NU** multime[i:j], multime[i:j:k], multime[:j], multime[i:]
- mulțimile nu au elemente de tip mulțime, listă (MUTABILE)

II.2. **Accesare elemente**

Element cu element

Exemplu:

```
multime =  {1, 2, 3, 15, 28, -5}
for element in multime:
        print(element)
```

Apartenența: in, not in

```
Exemplu:  my_set = {1, 5, 9, 13}
          if -4 not in my_set:
             print("-4 nu se gaseste in multime")
```

Comparare: ==, !=

Exemplu:

```
my_set1 = {1, 2, 5, 8}
my_set2 = { 2, 8, 5, 1}
print(my_set1 == my_set2) => True
```

Operatorii <, >, <=, >= (sunt utilizați pentru testatrea incluziunii)

Exemplu:

print({3, 5, 6} > {3, 5}) => True (mulțimea {3, 5, 6} include mulțimea {3, 5})

print({14, 15} < {14, 18}) => False (mulțimea {14,15} nu este inclusă în mulțimea {14, 18})

print({14, 15} < {14, 15, 16, 18}) => True (mulțimea {14, 15} este inclusă în mulțimea {14, 15, 16, 18})

Operatori specifici

OPERATOR	DENUMIRE	EFECT

|	reuniune	operația de reuniune pe mulțimi
&	intersecție	operația de intersecție pe mulțimi
-	diferență	diferența pe mulțimi
^	diferență simetrică	diferența simetrică pe mulțimi

Exemple:

set1 = {4, 5, 12, 18, 25, 36}

set2 = {12, 19, 25}

print(set1 | set2) => {4, 5, 36, 12, 18, 19, 25}

print(set1 & set2) => {25, 12}

print(set1 – set2) => {18, 4, 5, 36}

print(set1 ^ set2) => {4, 5, 36, 18, 19}

II.3. Metode uzuale

Metoda	Apel	Efect
issubset(set2)	set1.issubset(set2)	returnează True dacă set1 este submulțime a mulțimii set2 (False în caz contrar)
issuperset(set2)	set1.issuperset(set2)	returnează True dacă set2 este submulțime a mulțimii set1 (False în caz contrar)
union(set2)	set1.union(set2)	returnează reuniunea mulțimilor set1 și set2
intersection(set2)	set1.intersection(set2)	returnează intersecția mulțimilor set1 și set2
difference(set2)	set1.difference(set2)	returnează diferența mulțimilor set1 și set2
symetric_difference(s2)	set1.symetric_difference(set2)	returnează diferența simetrică mulțimilor set1 și set2
update(iterabil)	my_set.update(iterabil1 iterabil2, ..., iterabiln)	adaugă elementele colecțiilor iterabile primite ca parametru la mulțimea my_set
add(element)	my_set.add(element)	adaugă element la mulțimea my_set
remove(element)	my_set.remove(element)	elimină element din mulțimea my_set (**EROARE** dacă elementul nu se găsește în mulțime)
discard(element)	my_set.discard(element)	elimină un element din mulțimea my_set

Exemple:

set1 = {4, 5, 12, 18, 25, 36}

set2 = {12, 25}

print(set2.issubset(set1)) => True

print(set1.issuperset(set2)) => True

print(set1.union(set2)) => {18, 4, 5, 36, 25, 12}

print(set1.intersection(set2)) => {25, 12}

print(set1.difference(set2)) => {18, 4, 5, 36}

print(set1.symetric_difference(set2)) => {4, 5, 36, 18}

set1.add(16)

print(set1) => {16, 18, 4, 5, 36, 25, 12}

set1.remove(3) => EROARE, 3 nu se găsește în mulțime

set1.dicard(6) => 6 nu se găsește în mulțime, dar nu produce EROARE

set1.discard(16)

print(set1) => {18, 4, 5, 36, 25, 12}

set1.update(range(10, 14), [1, 11, 25], [36, 4])

print(set1) => {1, 4, 5, 10, 11, 12, 13, 18, 25, 36}

II.4. Clasa frozenset

- ➔ **IMUTABILE** (nu li se poate modifica valoarea după creare)
- ➔ elementele sunt **imutabile, unice și neindexate**
- ➔ **neordonate** (nu se păstrează ordinea de introducere)
- ➔ creare cu ajutorul constructorului **frozenset**()
- ➔ se pot aplica metodele clasei set, mai puțin cele care modifică mulțimea

Exemplu:

vid_frozenset = frozenset() – frozenset vid

my_frozenset = frozenset([1, 4, 5, 9, 18, 26])

print(my_frozenset) => frozenset({1, 4, 5, 9, 18, 26})

!ATENȚIE!

- reuniunea dintre frozenset si set => frozenset

- reuniunea dintre set si frozenset => set

III.Exerciții

i) Dicționare

1.Se citește n, număr natural și elementele a două liste, fiecare având n elemente. Să se convertească cele două liste într-un dicționar având drept chei elementele primei liste și drept valori elementele celei de-a doua liste.

Exemplu:

n = 4, lista1 = [`rosu', 'albastru', 'verde', 'alb'], lista2 = ['mar', 'cer', 'frunza', 'zapada']

2.Se dă un dicționar dict1 și o listă de chei din acest dicționar. Se cere să se creeze un nou dicționar ce conține doar cheile din lista dată.

Exemplu: dict1 = { "nume": "Ana", "varsta": 30, "salariu":10000, "oras": "Cluj" , "ore": 45}

lista = ["nume", "salariu", "ore"]

=>new_dict = { "nume": "Ana", "salariu":10000, "ore": 45}

3.În urma susținerii unui examen la un curs de programare, studenții au primit, pe lângă punctele obținute(punctele pot lua valori între 10 și 100) și un număr de ordine, reprezentând a câta sursă trimisă a fost rezolvarea lor. Scopul numărului de ordine este aflarea celui mai scurt timp de rezolvare pentru obținerea celei mai mari note.

Să se rezolve următoarele cerințe:

a) Se citesc de la tastatură n, număr natural, reprezentând numărul de elevi care au trimis rezolvările examenului și n perechi de forma punctaj obținut-nume student. Ordinea citirii stabilește clasamentul trimiterii surselor (primul student citit este cel care a trimis primul rezolvarea, al doilea student este cel care a trimis al doilea rezolvarea, etc). Să se memoreze datele citite într-o listă de tupluri de forma (punctaj, nume_student, numar_ordine).

Exemplu: n = 8 80 Popescu Ana 65 Mihai Alexandru 95 Petrescu Stefan
70 Aldea Ioana 95 Stanescu Maria 80 Radu Alexandra 65 Oprea Matei 85 Craciun Stefan

[(80, Popescu Ana, 1), (65, Mihai Alexandru, 2), (95, Petrescu Stefan, 3), (70, Aldea Ioana, 4), (95, Stanescu Maria, 5),(80, Radu Alexandra, 6), (65, Oprea Matei, 7), (85, Craciun Stefan, 8)]

b) Să se afișeze punctajele distincte obținute de studenți la examenul susținut.

Pentru datele de intrare de mai sus se obține: {80, 95, 65, 70, 85} nu neapărat în această ordine.

c) Folosind un dicționar, să se stocheze pentru fiecare punctaj distinct o listă cu tuplurile ce conțin numele și numărul de ordine al sursei studenților care au obținut punctele respective.

Pentru datele de intrare de mai sus se obține dicționarul:

{ 80: [(Popescu Ana, 1), (Radu Alexandra, 6)], 65:[(Mihai Alexandru, 2), (Oprea Matei, 7)], 95:[(Petrescu Stefan 3), (Stanescu Maria, 5)], 70:[(Aldea Ioana, 4)], 85:[(Craciun Stefan, 8)]

d) Să se afișeze numele și numărul de ordine al studentului care a obținut cel mai mare punctaj în cel mai scurt timp.

Pentru datele de mai sus se obține (Petrescu Stefan, 3)

4. Spiridușii lui Moș Crăciun au fiecare asociat câte un cod format cu litere și cifre. Până în luna noiembrie spiridușii au avut la dispoziție o agendă în care să completeze ce jucării pot face până la Crăciun și câte bucăți. O linie din agendă conține codul spiridușului, numărul de bucăți (număr natural) și numele jucăriei (numele este format din cuvinte separate prin câte un spațiu). Un spiriduș poate adăuga de mai multe ori o linie în agendă, chiar și cu aceeași jucărie, dacă se hotărăște că poate face mai multe.

Un exemplu de date corespunzătoare agendei :

S1 1 papusa	S2 1 papusa
S2 1 papusa	S2 2 masinuta
S3 1 masinuta	S1 10 ponei
S1 10 trenulet	S3 15 ponei

a) Să se memoreze datele din agendă, citite de la tastatură, astfel încât Moș Crăciun să poată afla cât mai repede informațiile cerute la punctele următoare.

b) Dat codul unui spiriduș, care sunt jucăriile pe care le poate face și ce cantitate din fiecare?

Pentru aceasta scrieți o funcție despre_spiridus cu 2 parametri: în primul parametru se transmite structura în care s-au memorat datele la punctul a), iar al doilea este codul unui spiriduș. Funcția returnează o lista cu elementele tupluri de 2 elemente – primul fiind numele jucăriei, iar al doilea cantitatea – ordonată descrescător după cantitate și, în caz de egalitate, crescător după nume. Apelați funcția pentru codul S1 și afișați lista returnata de funcție. Pentru datele din agenda de mai sus se va afișa [('ponei', 10), ('trenulet', 10), ('papusa', 1)]

c) Care este mulțimea jucăriilor pe care spiridușii le pot produce?

Pentru aceasta scrieți o funcție jucarii care primește ca parametru structura în care s-au memorat datele la punctul a) și returnează o mulțime cu numele jucăriilor care pot fi produse de spiriduși. Apelați funcția și afișați pe ecran elementele mulțimii returnate (pe o linie, separate prin virgula). Pentru datele din agenda de mai sus o posibilă ieșire este (jucăriile se pot afișa în orice ordine): ponei,trenulet,masinuta,papusa

d) Care este lista spiridușilor harnici: ordonați descrescător după numărul de jucării diferite pe care le pot face și, în caz de egalitate, descrescător după cantitatea de jucării pe care o vor produce și, în caz de egalitate, crescător după cod?

Pentru aceasta scrieți o funcție spiridusi care primește ca parametru structura în care s-au memorat datele și returnează o lista cu elementele tupluri de 3 elemente – primul fiind codul spiridușului, al doilea numărul de jucării diferite pe care le poate produce, iar al treilea numărul total de bucăți de jucării pe care le poate produce spiridușul – ordonată după criteriile cerute de Moș Crăciun (precizate anterior). Apelați funcția și afișați pe ecran elementele listei obținute, fiecare tuplu din listă fiind afișat pe o linie separată.

Pentru datele din agenda de mai sus se va afișa ('S1', 3, 21) ('S3', 2, 16) ('S2', 2, 4)

(Problemă adaptată după problemă recapitulare (curs/laborator) test laborator grupa 244, an universitar 2020-2021)

5.https://pynative.com/python-dictionary-quiz/

ii) Mulțimi

1.Se citesc n și m, numere naturale și n, respectiv m, numere întregi, elementele a două mulțimi. Să se afișeze mulțimea formată din elementele ce se află în prima mulțime sau în cea de-a doua mulțime, dar nu se află în ambele.

Exemplu: n = 5, m = 5, set1 = {10, 20, 30, 40, 50}, set2 = {30, 40, 50, 60, 70} =>

{20, 70, 10, 60}

2. https://pynative.com/python-set-quiz/