Tutoriatul 03 - tupluri, dicționare

Probleme cu tupluri

Problema 1

Se citește de la tastatură un număr natural *n* nenul. Să se creeze și afișeze un tuplu format din numere naturale divizibile cu 2 și cu cifra unităților mai mică decât restul împărțirii la 9 a numărului respectiv mai mici sau egale decât *n*.

Exemplu: pentru n = 27, se va afișa (10, 12, 14, 16, 20, 22, 24, 26)

Indicație: folosiți comprehensiunea de tupluri

```
n = int(input())
t = tuple(x for x in range(0, n + 1, 2) if x % 2 == 0 and x % 10 < x % 9 )
print(t)
 27
(10, 12, 14, 16, 20, 22, 24, 26)
```

Problema 2

În scopul acestei probleme, considerați reprezentarea de *(a, b)* pentru un număr complex. Se citesc de la tastatură numerele întregi *a*, *b*, *c*, *d*. Să se afișeze numărul complex *(x,y)* reprezentat de produsul numerelor complexe reprezentate de *(a,b)* și *(c,d)* și modulul acestuia (calculat cu funcția *sqrt( )*).

Exemplu: pentru (1,2) și (3,4), produsul va fi (-5, 10) și modulul acestuia va fi aproximativ 11,18

```
import math
a = int(input())
b = int(input())
c = int(input())
d = int(input())
p = (a * c - b * d, a * d + c * b)
print(p, math.sqrt(sum(x ** 2 for x in p)))
 1
 2
 3
 4
(-5, 10) 11.180339887498949
```

Problema 3

Se citesc de la tastatură 4 numere întregi, *a*, *b*, *c* și *d*, cu *a*<*b* și *c*<*d*. Să se afișeze intersecția și reuniunea intervalelor *(a, b)* și *(c, d)* sub formă de interval sau reuniune de intervale sau mulțime.

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
if a >= d or b <= c:
    inter = {}  #vom considera aceasta ca fiind multimea vida, desi este un abuz de atr
ibuire
    reuni = "reuniune disjuncta"
elif c < a:
    if b < d:
        inter = (a, b)
        reuni = (c, d)
    else:
        inter = (a, d)
        reuni = (c, b)
else:
    if b < d:
        inter = (c, b)
        reuni = (a, d)
    else:
        inter = (c, d)
        reuni = (a, b)
print(inter, reuni)
```

```
 0
 1
 1
 7
{} reuniune disjuncta
```

Problema 4

Se citesc de la tastatură, pe o singură linie,un număr întreg *a*, un șir de numere întregi de lungime cel puțin 3 și un număr întreg *b*, *b>a*. Să se afișeze câte elemente din interiorul șirului sunt cuprinse în intervalul *(a, b)*. Scoateți elementele cu valoare maximă din șir și afișați-l sub formă de tuplu.

Exemplu: pentru șirul 1 2 3 4 5 6 7 8 8 8 8 4 2 3 4

se va afișa

4

[2, 3, 4, 5, 6, 7, 4, 2, 3]

```
a, *t, b = (int(x) for x in input().split())
c = 0
mx = t[0]
for i in t:
    if a < i < b:
        c += 1
    if i > mx:
        mx = i
print(c)
ct = t.copy()
for i in ct:  #folosim o copie pentru a nu modifica lista pe masura ce o parcurgem
    if i == mx:
        t.remove(i)
print(t)
```

```
 1 2 3 4 5 6 7 8 8 8 8 4 2 3 4
4
[2, 3, 4, 5, 6, 7, 4, 2, 3]
```

Problema 5

Se citesc de la tastatură două numere întregi *a*, *b* cu *a<b*, și un număr natural nenul *r*. Să se afișeze sub formă de tuplu termenii seriei aritmetice cu primul termen *a* și rația *r* care se găsesc în intervalul *[a, b]*. Să se afișeze suma lor.

Indicație: nu folosiți formule matematice pentru sumă

```
a = int(input())
b = int(input())
r = int(input())
t = tuple(a + r * i for i in range((b - a) // r + 1))
print(t, sum(t))
```

```
 2
 10
 1
(2, 3, 4, 5, 6, 7, 8, 9, 10) 54
```

Problema 6

Apreciați rezultatul următoarei secvențe de cod, cu precizările că $a<b$, $c<d$, $b<d$ și a este diferit de c.

Un indiciu poate fi găsit în comentariile acestei celule.

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
if a < c:
    if c < b < d:
        e, f = c, b
    elif d < b:
        e, f = c, d
    else:
        e, f = 0, -1
else:
    e, f = a, b
print(b - a + d - c - f + e + 1)
```

```
 1
 4
 6
 9
8
```

Dicționare

Dicționarele în python sunt colecții de elemente neordonate, mutabile și alcătuite din perechi de tipul *cheie - valoare*. Cheia unei valori (diverse colecții de date pot reprezenta o valoare în contextul ăsta) trebuie să fie unică, imutabilă și hashable. De exemplu, numerele întregi, șirurile de caractere, tuplurile (formate din elemente imutabile) sunt elemente hashable.

Pe scurt,

dicționarul este o colecție de elemente, unde fiecare element este indexat în mod diferit pentru o parcurgere mai eficientă a datelor.

Exemplu:

Să presupunem că avem următorul tabel de date care conține informații despre studenții de la opționalul ASC (trei la număr):

- 2501910000034 Ionescu Ion 10 8 7 8
- 2402900000041 Marinica Maria 9 10 8 8 8
- 1412900000041 Petrescu Petrica 8 10 4 7

Scopul nostru este să organizăm datele din tabel astfel încât să putem accesa notele fiecărui elev folosind numai CNP-ul (un exemplu bun de cheie pentru dictionar).

Soluția este să creăm un dicționar care are drept chei CNP-ul fiecărui student și valoarea asociată fiecărei cheie să fie o listă care conține notele studentului respectiv.

```
note_studenti_ASC = {"2501910000034":
[10, 8, 7, 8], "2402900000041" : [9, 10, 8, 8, 8], "1412900000041" : [8, 10, 4, 7]}
print(note_studenti_ASC)
for CNP in note_studenti_ASC:
    print(note_studenti_ASC[CNP]) #parcurgere neobisnuita pentru o colectie de elemente
 in python
```

```
{'2501910000034': [10, 8, 7, 8], '2402900000041': [9, 10, 8, 8, 8], '1412900000041': [8,
[10, 8, 7, 8]
[9, 10, 8, 8, 8]
[8, 10, 4, 7]
```

Crearea dicționarelor

Dicționarele pot fi create în general folosind una dintre următoare trei metode:

1. Atribuire directă (*mot à mot*), adică înșiruire de "*cheie* : element" între acolade.

```
d = {"numere prime" : [2, 3, 5, 7, 11], "numere fibonacci" : [1, 2, 3, 5, 8, 13]}
print(d)
```

```
{'numere prime': [2, 3, 5, 7, 11], 'numere fibonacci': [1, 2, 3, 5, 8, 13]}
```

2. Prin funcția *dict()*, fie prin definirea cheilor ca tip string *(vezi d1)*, fie prin înșiruirea unor secvențe de perechi de ti 'cheie-valoare' *(vezi d2)*.

Atribuirea perechilor prin mulțimi nu este recomandată. Mulțimile sunt neordonate, deci oricare dintre cele două elemente poate deveni cheia.

```
d1 = dict(a = [0, 1, 2], ziua = "luni")
print(d1)
tuplu_d2 = ([12, "marian"], ('valul', 'traian'), {123, 'nasol'})
d2 = dict(tuplu_d2)
print(d2) #123 nu a fost luat ca cheie, lucru care nu e convenabil
```

```
{'a': [0, 1, 2], 'ziua': 'luni'}
{12: 'marian', 'valul': 'traian', 'nasol': 123}
```

3. Prin popularea succesivă, adică simpla atribuire de forma *dictionar[cheie_nouă]=valoare_nouă*

```
dict_studenti = {}
print(dict_studenti)
dict_studenti['Tudor'] = [6, 7, 8]
print(dict_studenti)
dict_studenti['Marius'] = [3, 8, 9]
print(dict_studenti)
```

```
{}
{'Tudor': [6, 7, 8]}
{'Tudor': [6, 7, 8], 'Marius': [3, 8, 9]}
```

Accesare, modificarea și adăugarea elementelor

Accesarea unui element dintr-un dicționar se realizează pe bază de *cheie*, adică respectiva *cheie* va acționa similar unui index, cum se folosește la structuri de date de tip listă sau tuplu.

```
print(dict_studenti['Tudor'])
print(dict_studenti['Tudor'][0]) #putem apela elemente din lista in mod intuitiv
```

```
[6, 7, 8]
6
```

Deoarece dicționarele sunt mutabile, elementele pot suferi modificări prin simple atribuiri, iar ștergerea unei perechi *cheie-valoare* se realizează prin instrucțiunea *del*.

```
dict_studenti['Tudor'][0] = "anulat"
del dict_studenti['Marius'] #Marius a fost exmatriculat
print(dict_studenti)
```

```
{'Tudor': ['anulat', 7, 8]}
```

Parcurgerea dicționarelor

Dacă ar fi aplicată unui dicționar o parcurgere uzuală pentru colecțiile de date, anume *for el in colectie: ...*, ce ar fi parcurs de fapt ar fi *cheile* dicționarului respectiv *(vezi primul exemplu)*. Această metoda este echivalentă cu parcurgerea colecției *dict.keys()*

```
note_studenti_ASC = {"2501910000034":
[10, 8, 7, 8], "2402900000041" : [9, 10, 8, 8, 8], "1412900000041" : [8, 10, 4, 7]}
print(note_studenti_ASC)
for CNP in note_studenti_ASC:
    print(CNP)
    print(note_studenti_ASC[CNP])
```

```
{'2501910000034': [10, 8, 7, 8], '2402900000041': [9, 10, 8, 8, 8], '1412900000041': [8,
2501910000034
[10, 8, 7, 8]
2402900000041
[9, 10, 8, 8, 8]
```

```
1412900000041
[8, 10, 4, 7]
```

```
for CNP in note_studenti_ASC.keys():
    print(CNP)
    print(note_studenti_ASC[CNP])
```
```
2501910000034
[10, 8, 7, 8]
2402900000041
[9, 10, 8, 8, 8]
1412900000041
[8, 10, 4, 7]
```

Astfel, pentru o parcurgere eficientă, strict pe valorile dicționarului, se parcurge colecția *dict.values()*, care conține strict valorile respective.

```
for element in note_studenti_ASC.values():
    print(element)
```
```
[10, 8, 7, 8]
[9, 10, 8, 8, 8]
[8, 10, 4, 7]
```

În plus, pentru a parcurge perechile *cheie-element* se utilizează *dict.items()*.

```
for pereche in note_studenti_ASC.items():
    print(pereche)
```
```
('2501910000034', [10, 8, 7, 8])
('2402900000041', [9, 10, 8, 8, 8])
('1412900000041', [8, 10, 4, 7])
```

Operații cu sau asupra dicționarelor

Metode comune care funcționează și pe dicționare sunt

- len(d)
- sum(d) (adună cheile dicționarului *d*, dă eroare dacă nu sunt de tip numeric)
- min, max (dacă valorile sunt comparabile!)
- del (poate șterge perechea *cheie-valoare*, sau tot dicționarul dacă nu este specificat vreo cheie)

```
d1 = {"a":2, "b":3 , "c":4}
print(len(d1))
print(max(d1))
print(max(d1.values()))
```
```
3
c
4
```

Operatorii *in*, *not in* acționează asupra *cheilor*, nu asupra valorilor.

```
print("a" in d1)
del d1["a"]
print("a" not in d1)
```
```
True
True
```

Metode pentru dicționare

Convenție de notare: functie(p1[, p2]) implică că parametru 1 este necesar pentru apelarea funcției, iar parametrul 2 nu este obligatoriu. În schimb, pentru a avea un parametru 2 este obligatoriu un parametru 1.

- d.clear()

Șterge toate perechile din *d*, iar *d* rămâne un dicționar vid (*d* nu dispare complet din memorie, cum se întâmplă la *del d*).

- d.copy()

Returnează o copie simplă a lui *d*.

- d.fromkeys(it[, val])

Returnează un dicționar nou având cheile date de iterabilul *id* și toate valorile egale cu *val*. *val* este implicit None.

- d.get(key[, default])

Returnează valoarea cheii *key*. Dacă nu există, returnează valoarea *default*. *default* este implicit None.

- d.pop(k[, d])

Elimină din *d* perechea asociată cheii *k* și returnează valoarea cheii asociate. Dacă nu există, este returnată valoarea *d*, iar dacă *d* nu a fost dat ca parametru, funcția dă eroare.

- d.popitem()

Elimină arbitrar o pereche *cheie-valoare*. Dacă *d* este vid, funcția dă eroare.

- d.setdefault(key[, default])

Inserează perechea *key-default* în d dacă cheia *key* nu există. *default* este implicit None.

- d.update([E,]**F)

Adaugă la d perechile definite de E sau **F, unde E poate fi un dicționar și **F poate fi o enumerare de perechi *cheie-valoare*. Notația mai ciudată a parametrilor vrea să semnifice că atunci când sunt folosiți și dicționare și perechi *cheie-valoare*, dicționarele sunt trecute înainte.

Pentru a înțelege mai ușor metodele specifice dicționarelor, recomand parcurgerea următoarelor

Probleme Propuse

Problema 1

Ați fost angajați de un magazin de electrocasnice pentru a reorganiza eficient într-o colecție date despre

produsele magazinului. Din păcate, fostul informatician nu știa de colecția de tip *dicționar* (de unde și concedierea imediată) și a organizat datele în liste imbricate sub forma:

[tip_produs1, [unitati_vandute, pret_unitate], tip_produs2, [unitati_vandute, pret_unitate], ...]

Reorganizați informațiile într-un dicționar, care să aibă drept cheie tipul produsului și valoare asociată încasările din vânzarea respectivului produs și afișați-l.

Exemplu: pentru

ls = ["frigider", [10, 600], "cuptor cu microunde", [20, 300], "masina de spalat rufe", [80, 1000]]

se va crea în memoria și afișa

{"frigider" : 6000, "cuptor cu microunde" : 6000, "masina de spalat rufe" : 80000}

În scopul acestei probleme, considerați că lista este asociată variabilei ls în program.

```
ls = ["frigider", [10, 600], "cuptor cu microunde", [20, 300], "masina de spalat rufe",
 [80, 1000]]
dict_magazin = {}
for element in ls:
    if type(element) is str:
        dict_magazin[element] = 0
        precedent = element
    else:
        dict_magazin[precedent] = element[0] * element[1]
print(dict_magazin)
```

```
{'frigider': 6000, 'cuptor cu microunde': 6000, 'masina de spalat rufe': 80000}
```

Problema 2

Facultatea de Matematică și Informatică a decis ca studenții care nu au media notelor de la Algebră I strict mai mare decât 5 să fie exmatriculați. În calitate de informatician șef al facultății, treaba ta este să scoți din dicționarul anului I datele asociate studenților exmatriculați și să îl afișezi pe ecran. Dicționarul este de forma

dict_an1 = {CNP_student1 : lista_note, CNP_student2 : lista_note, ...}

Exemplu: pentru dicționarul

dict_an1 = {1234 : [3, 4, 5, 6, 7], 1235 : [4, 5, 4, 4], 1236 : [10, 10, 1, 1, 2], 1237 : [4, 10]}

se va afișa

{1234 : [3, 4, 5, 6, 7], 1237 : [4, 10]}

În scopul acestei probleme, considerați că lista este asociată variabilei ls în program.

Indicație: folosiți o copie a dicționarului pentru a elimina chei progresiv

```
dict_an1 = {1234 : [3, 4, 5, 6, 7], 1235 : [4, 5, 4, 4], 1236 : [10, 10, 1, 1, 2], 1237
 : [4, 10]}
copie_dict = dict_an1.copy() #este necesara o copie pentru a elimina progresiv cheile
for i in dict_an1.keys():
    ma = sum(dict_an1[i])/len(dict_an1[i])
```

```
    if ma < 5:
        copie_dict.pop(i)
dict_an1 = copie_dict.copy()
print(dict_an1)
```
```
{1234: [3, 4, 5, 6, 7], 1237: [4, 10]}
```

Problema 3

Participanții unui concurs au avut punctajele și vârstele salvate într-o structură de date de tip dicționar (cheia asociată valorilor fiind numele de utilizator cu care au participat la concursul respectiv, iar valoarea asociată o listă cu două elemente). Să se construiască și afișeze un dicționar cu aceeași structură cu cei calificați în etapa a doua, adică cei cu un punctaj mai mare sau egal cu 80 și cu vârsta cuprinsă în (15, 25). Valoarea asociată unei chei va fi 0.

Exemplu: pentru

d_etapa1 = {"cosmos_dan":[81, 20], "marian123":[87, 26], "supermen":[65, 16]}

se va construi și afișa

{"cosmos_dan":0}

```
d_etapa1 = {"cosmos_dan":[81, 20], "marian123":[87, 26], "supermen":[65, 16]}
ls = []
for nume in d_etapa1.items():
    if nume[1][0] > 80 and 15 < nume[1][1] < 25:
        ls.append(nume[0])
d_etapa2 = {}.fromkeys(ls, 0)
print(d_etapa2)
```
```
{'cosmos_dan': 0}
```