Laborator_2_tema1

February 25, 2023

1 Laborator 2 - tema 1

1.1 Context

Precizari:

1. Studentii se pot consulta reciproc, dar rezolvarile vor fi individuale. Daca la o problema nu se pot furniza explicatii legate de rezolvare, nota pentru intreaga tema va fi 0.
2. Temele se predau in activitatea de elearning asociata, pana in 15 martie ora 23. Se acorda 2 puncte din oficiu.
3. Tema va fi implementata intr-un singur fisier Jupyter Notebook, avand numele: Tema1_IA_nume_prenume.ipynb. Exemplu: Tema1_IA_Popescu_Ioana.ipynb
4. Se vor folosi type annotations pentru variabile, parametrii functiilor, tipuri de retur. Se vor folosi docstrings pentru functii, in limba romana sau engleza. Neindeplinirea acestei cerinte duce la injumatatirea punctajului pe exercitiul in cauza.

1.2 Exercitii simple

1. (1 p). Se da o lista `lista` de numere intregi. Sa se depuna numerele pare in lista `lista_pare` numerele pare si in `lista_impare` cele impare.

Exemplu:

[ ]:
```
lista = [223, 1019, 4, 658, 8, 9, -4]

lista_pare = ...
lista_impare = ...

# in mod normal, lista numerelor pare/impare
assert lista_pare == [4, 658, 8, -4]
assert lista_impare == [223, 1019, 9]
```

Pentru exemplele proprii adaptati corespunzator cele doua asertiuni.

2. (1 p) Se da o lista de stringuri. Se cere ca in fata fiecarui string cu lungime para sa se insereze lungimea acelui string.

Exemplu:

[ ]:
```
my_string = "Atunci când citești un text trebuie să fii capabil să identifici␣
↪ideea principală a textului și să o poți formula în propriile tale cuvinte"
```

```
my_list = my_string.split(' ')

...
print(my_list)

assert my_list == [6, 'Atunci', 4, 'când', 'citești', 2, 'un', 4, 'text',␣
 ↪'trebuie', 2, 'să', 'fii', 'capabil', 2, 'să', 10, 'identifici', 'ideea',␣
 ↪10, 'principală', 'a', 8, 'textului', 2, 'și', 2, 'să', 'o', 4, 'poți',␣
 ↪'formula', 2, 'în', 'propriile', 4, 'tale', 'cuvinte']
```

3. (1 p) Se da o lista de cel putin 3 numere in virgula mobila. Care este al treilea cel mai mic numar?

Exemplu:

[ ]:
```
lista_numere = [1, -1, 2, -2, 3, -3, 4, -4, 100, 0]

al_treilea = ...
print(al_treilea)

assert al_treilea == -2
```

4. (1 p) Se da o lista de stringuri. Sa se determine frecventa de aparitie a fiecarui string, intr-un dictionar: cheia dintr-un dictionar este cuvantul, iar valoarea este numarul de aparitii.

Optional: dictionarul va contine cuvinetele in ordine descrescatoare a numarului de aparitii.

Exemplu:

[ ]:
```
my_string = "Sed ut perspiciatis unde omnis iste natus error sit voluptatem␣
 ↪accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab␣
 ↪illo inventore veritatis et quasi architecto beatae vitae dicta sunt␣
 ↪explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit␣
 ↪aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem␣
 ↪sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit␣
 ↪amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora␣
 ↪incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad␣
 ↪minima veniam, quis nostrum exercitationem ullam corporis suscipit␣
 ↪laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum␣
 ↪iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae␣
 ↪consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?"
my_string = my_string.lower().replace(',', '').replace('.', '').replace('?', '')
my_list = my_string.split(' ')

dict_frecvente = {}
...

print(dict_frecvente)
```

```
assert dict_frecvente == {'sed': 3, 'ut': 4, 'perspiciatis': 1, 'unde': 1,␣
↪'omnis': 1, 'iste': 1, 'natus': 1, 'error': 1, 'sit': 3, 'voluptatem': 4,␣
↪'accusantium': 1, 'doloremque': 1, 'laudantium': 1, 'totam': 1, 'rem': 1,␣
↪'aperiam': 1, 'eaque': 1, 'ipsa': 1, 'quae': 1, 'ab': 1, 'illo': 1,␣
↪'inventore': 1, 'veritatis': 1, 'et': 2, 'quasi': 1, 'architecto': 1,␣
↪'beatae': 1, 'vitae': 1, 'dicta': 1, 'sunt': 1, 'explicabo': 1, 'nemo': 1,␣
↪'enim': 2, 'ipsam': 1, 'quia': 4, 'voluptas': 2, 'aspernatur': 1, 'aut': 2,␣
↪'odit': 1, 'fugit': 1, 'consequuntur': 1, 'magni': 1, 'dolores': 1, 'eos':␣
↪1, 'qui': 4, 'ratione': 1, 'sequi': 1, 'nesciunt': 1, 'neque': 1, 'porro':␣
↪1, 'quisquam': 1, 'est': 1, 'dolorem': 2, 'ipsum': 1, 'dolor': 1, 'amet': 1,␣
↪'consectetur': 1, 'adipisci': 1, 'velit': 2, 'non': 1, 'numquam': 1, 'eius':␣
↪1, 'modi': 1, 'tempora': 1, 'incidunt': 1, 'labore': 1, 'dolore': 1,␣
↪'magnam': 1, 'aliquam': 1, 'quaerat': 1, 'ad': 1, 'minima': 1, 'veniam': 1,␣
↪'quis': 2, 'nostrum': 1, 'exercitationem': 1, 'ullam': 1, 'corporis': 1,␣
↪'suscipit': 1, 'laboriosam': 1, 'nisi': 1, 'aliquid': 1, 'ex': 1, 'ea': 2,␣
↪'commodi': 1, 'consequatur': 2, 'autem': 1, 'vel': 2, 'eum': 2, 'iure': 1,␣
↪'reprehenderit': 1, 'in': 1, 'voluptate': 1, 'esse': 1, 'quam': 1, 'nihil':␣
↪1, 'molestiae': 1, 'illum': 1, 'fugiat': 1, 'quo': 1, 'nulla': 1, 'pariatur':
↪ 1}
```

5. (2p) Se da un dictionar in care cheile sunt nume de studenti, iar valorile sunt liste de carti preferate. Sa se determine:
   1. Care sunt studentii care au in lista de preferinte o carte specificata?
   2. Care sunt perechile de studenti care au aceleasi preferinte de carti (ordinea in lista de preferinte nu e relevanta)
   3. Pentru un student dat, care sunt studentii cu care are cele mai multe carti comune in liste de preferinte; daca sunt mai multi astfel de studenti cei mai apropiati, se vor enumera toti.
   4. Pentru doi studenti, A si B, care e lista de stergeri si adaugari prin care lista lui A devine identica cu lista lui B (ordinea in lista nefiind importanta)
   5. Toate perechile de studenti A, B pentru care lista de preferinte a lui A este inclusa in (dar nu coincide cu) lista de preferinte a lui B.

Indicatie: considerati tipul de date `set`.

Exemplu:

```
preferinte = {
    'Popescu': ['carte2', 'carte1', 'carte3', 'carte4'],
    'Ionescu': ['carte1', 'carte2', 'carte3', 'carte7', 'alta carte'],
    'Georgescu': ['carte1', 'alta carte'],
    'Xulescu': ['carte2', 'carte5', 'carte6', 'carte7'],
    'Dragomir': ['carte4', 'carte1', 'carte2', 'carte3']
}
```

```
1. carte = 'alta carte' -> ['Ionescu', 'Georgescu']
2. Popescu si Dragomir au aceleasi preferinte: ['carte2', 'carte1', 'carte3', 'carte4']. Se re
3. Xulescu -> Suprapunere maxima de preferinte: 2 cu colegii: ['Ionescu']
4. A='Popescu', B='Ionescu' -> Se vor adauga cartile: {'carte7', 'alta carte'}, se vor scoate
```

`5. Preferintele lui Georgescu sunt subset propriu al preferintelor lui Ionescu`

6 (1p). Se da o lista de stringuri si o lista de intregi; nicio lista nu contine valori duplicate. Sa se determine toate combinatiile de elemente din prima lista cu elemente din a doua lista, cu valorile despartite prin '_'.

Exemplu:

[ ]:
```
lista_1 = ['laptop', 'carte', 'telefon']
lista_2 = list(range(1, 5))

...

assert lista_combinatii == ['laptop_1', 'laptop_2', 'laptop_3', 'laptop_4',
                            'carte_1', 'carte_2', 'carte_3', 'carte_4',
                            'telefon_1', 'telefon_2', 'telefon_3', 'telefon_4']
```

1.3 Exercitii cu functii

7 (1 p). Scrieti o functie care primeste un numar intreg si returneaza suma cifrelor sale.

8 (2 p). Se da un fisier de tip text, pentru care se vor considera doar acele linii din fisier care contin exact un numar intreg. Sa se scrie functii care: * returneaza doar lista numerelor 'valide' - folosita mai departe ca functie auxiliara; * determina daca numerele din fisier sunt sau nu distincte, returnand `True` respectiv `False`; * returneaza cele mai mari 3 numere din fisier, in ordine descrescatoare; * produce lista tuturor tripletelor distincte `(x, y, z)` cu `(x < y < z)`, unde `x, y, z` sunt numere din fisier.

Exemplu: pentru continutul de fisier

```
#1
2
4
-4
3
-3
10
2
abc
100
```

functiile vor returna, respectiv: * `[2, 4, -4, 3, -3, 10, 2]` * `False` * `[10, 4, 3]` * `[(-4, -3, 2), (-4, -3, 3), (-4, -3, 4), (-4, -3, 10), (-4, 2, 3), (-4, 2, 4), (-4, 2, 10), (-4, 3, 4), (-4, 3, 10), (-4, 4, 10), (-3, 2, 3), (-3, 2, 4), (-3, 2, 10), (-3, 3, 4), (-3, 3, 10), (-3, 4, 10), (2, 3, 4), (2, 3, 10), (2, 4, 10), (3, 4, 10)]`

Pentru lucrul cu fisiere, puteti consulta, de exemplu: https://www.w3schools.com/python/python_file_open.asp, https://www.geeksforgeeks.org/reading-writing-text-files-python/.

9. (2 p) Sa se scrie o functie `sum_divisors_digits`care preia un numar natural strict pozitiv n si returneaza suma cifrelor divizorilor sai. Intr-o alta functie `black_hole` se apeleaza in mod

repetat `sum_divisors_digits` pe numerele rezultate, pana cand se atinge un numar maxim de iteratii (implicit 1000) sau se ajunge la numarul 15. Puteti crea alte functii auxiliare.

Exemplul 1: se pleaca de la $n = 15$; divizorii sunt $1, 3, 5, 15$; suma cifrelor divizorilor este $1 + 3 + 5 + 1 + 5 = 15$, acesta fiind rezultatul apelului `sum_divisors_digits(15)`. Functia `black_hole` se opreste deci dupa un singur apel al functiei `sum_divisors_digits`.

Exemplul 2: se pleaca de la $n = 21$; divizorii sunt $1, 3, 7, 21$ iar `sum_divisors_digits(21)` este $1 + 3 + 7 + 2 + 1 = 14$; se reia in functia `black_hole` apelul functiei `sum_divisors_digits` pana la epuizarea numarului de apeluri sau potentiala stabilizare in 15.

1.4 Exercitii cu NumPy

In rezolvarea exercitiilor de mai jos se va folosi cod *vectorizat* si collection comprehension (preferabil insa doar cod vectorizat). Puteti folosi functii NumPy.

10. (1 p) Scrieti o functie care pentru un vector dat a dat de n componente, returneaza un vector cu diferenta perechilor de elemente adiacente: $b[i] = a[i] - a[i + 1], 0 \leq i \leq n - 2$.

Exemple: * `a = np.array([1, 2, 10, 3])`; `diff_pairs_vector(a)` va returna vectorul NumPy cu continutul `(-1, -8, 7)`.

11. (1p) Sa se scrie o functie care gaseste pozitiile minimelor locale dintr-un vector numpy. Un minim local este o valoare care are in vecinii imediati (indicele curent ± 1, fara a iesi din vector) valori strict mai mari decat ea.

Exemplu: `[-1, 3, -7, 1, 2, 6, 0, 1]` -> `[-1, -7, 0]`.

12. (1 p) Sa se scrie o functie care spune daca o matrice patratica este magica: suma pe fiecare linie, pe fiecare coloana, suma elementelor de pe fiecare diagonala este acelasi numar.

Exemplu de matrice magica:

```
11  24  7   20  3
4   12  25  8   16
17  5   13  21  9
10  18  1   14  22
23  6   19  2   15
```

13. (1 p) Sa se scrie o functie care primeste o matrice de numere si determina daca numarul de valori aflate in intervalul $(-a, b)$ (a, $b > 0$, parametri al functiei, numere reale pozitive) este peste un procent p, parametru dat. Functia va returna valoare booleana.

14. (2 p) Sa se scrie o functie care primeste o matrice `a` si returneaza o alta de aceeasi forma cu `a`, astfel incat valorile de pe fiecare coloana sa fie in intervalul [0, 1]. Se va proceda astfel: pentru o coloana oarecare se calculeaza minimul si maximul de pe ea m, M, fiecare valoare v din acea coloana va produce valoarea:

$$v' = \frac{v - m}{M - m}$$

Se presupune ca matricea `a` nu are coloane cu valori constante.

Scrieti `assert` care sa valideze functia implementata.

[ ]: