Lab08

COMP 125 Programming with Python

Recording Disclaimer

The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

Q1: Word Frequencies

- In this assignment we will count the frequency of words occurring in a given text.
- Let's look at the famous Charles Dickens quote (all lower case and no punctuation and digits)

"**it was** the best of times **it was** the worst of times **it was** the age of wisdom **it was** the age of foolishness **it was** the epoch of belief **it was** the epoch of incredulity **it was** the season of light **it was** the season of darkness **it was** the spring of hope **it was** the winter of despair"

- Unique **words** and **counts** (frequencies) :

words: ['it', 'was', 'the', 'best', 'of', 'times', 'worst', 'age', 'wisdom', 'foolishness', 'epoch', 'belief', 'incredulity', 'season', 'light', 'darkness', 'spring', 'hope', 'winter', 'despair']

counts:[10, 10, 10, 1, 10, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1]

Implement **word_count** function

- Write a function **word_count** that takes a string containing the text to be processed as an argument. **word_count** function should return two lists: **words** and **counts**, where
  words contains all the unique words that are present in the given text.
  counts contains the frequencies of the words

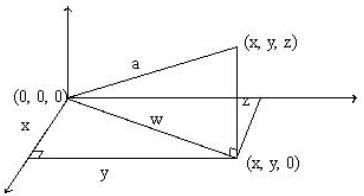
- Split the text into words
- Create two empty lists; list of **words** and a list of **counts**. We will use the index of a given word in the list **words**, to find the frequency of this word in the list **counts**. Hint: lookup the list method **index**.
- Iterate over the split words
  - Check if the word is in the list of **words** (remember the **`in`** keyword)
  - If it is an existing word, figure out its index and increment the corresponding **counts** list item by 1
  - If it is a new word, append it to the **words** list and append 1 to the **counts** list

Q2: Vector functions

- A vector in three dimensions can be represented as a list of length three **[x, y, z]**, where x, y and z are the components of the vector.
- Suppose that **vec1** and **vec2** are two vectors (with components **[x_1, y_1, z_1]** and **[x_2, y_2, z_2]**)
- Implement the following functions for vectors
    - **vector_length**: receives a vector (**vec1)**, returns the magnitude of the vector (**m**), where **m** is defined as **m=sqrt(x_1*x_1+ y_1*y_1+ z_1*z_1)**

    - **vector_add**: receives two vectors, returns a single vector containing the sum of the input vectors
      Ex: if **vec1=[x_1, y_1, z_1]** and **vec2=[x_2, y_2, z_2]**, then the resulting vector is
      vec3= [x_1+x_2, y_1+y_2, z_2+z_2]

    - **vector_dot**: receives two vectors (**[x_1, y_1, z_1]** and **[x_2, y_2, z_2])**, returns a floating point number (**d**) as the dot product of these vectors **d=(x_1*x_2 + y_1*y_2 + z_1*z_2)**



Vector functions

vector_demo: receives two vectors, has no return. By using the functions **vector_length**, **vector_add**, and **vector_dot** prints out information regarding the given vectors.
Implement this function by using f-string so that it contains **maximum 5 lines** of code including the definition. It should work as follows:

```
#Implement the following functions
def vector_add():

def vector_length():

def vector_dot():

def vector_demo():


vec1=[1.,2.,3.]
vec2=[3.,1.,4.]
vector_demo(vec1, vec2)
```

Output
Note the format for the floating point values.

```
In [20]: runcell(0, './08-vectors.py')
The length of the vector [1.0, 2.0, 3.0] is 3.742
The length of the vector [3.0, 1.0, 4.0] is 5.099
The sum of the vectors [1.0, 2.0, 3.0] and [3.0, 1.0, 4.0] is [4.0, 3.0, 7.0]
The dot product of the vectors [1.0, 2.0, 3.0] and [3.0, 1.0, 4.0] is 17.0
```

Formatting

Old School Formatting vs An Improved String Formatting Syntax

Option #1: %-formatting

```
In [32]: name="Comp. 125"

In [33]: "Hello %s" % name
Out[33]: 'Hello Comp. 125'
```

If we update “name”

```
In [34]: name="Comp. 125 Sec. 2"

In [35]: "Hello %s" % name
Out[35]: 'Hello Comp. 125 Sec. 2'
```

Option #2: str.format()

```
In [43]: name="Greta"

In [44]: lastname="Thunberg"

In [45]: "Hello, {0}. You are doing a great job {0} {1}.".format(name, lastname)
Out[45]: 'Hello, Greta. You are doing a great job Greta Thunberg.'
```

Formatting

f-string: A new formatting approach

```
In [46]: f"Hello, {name}. You are doing a great job {name} {lastname}."
Out[46]: 'Hello, Greta. You are doing a great job Greta Thunberg.'
```

```
In [47]: name="Atlas"

In [48]: lastname="Sarrafoğlu"

In [49]: f"Hello, {name}. You are doing a great job {name} {lastname}."
Out[49]: 'Hello, Atlas. You are doing a great job Atlas Sarrafoğlu.'
```

How do you want to align your text?

```
In [73]: f"{name:<10}"
Out[73]: 'Atlas     '

In [74]: f"{name:>10}"
Out[74]: '     Atlas'

In [75]: f"{name:^10}"
Out[75]: '  Atlas   '
```

Formatting

f-string: Further control

Type declaration not required in f-string !

```
In [58]: import math

In [59]: f"{2 * math.pi}"
Out[59]: '6.283185307179586'

In [60]: f"{2 * math.pi:10.5}"
Out[60]: '    6.2832'
```

Old style formatting for floating point numbers

```
In [63]: "%f" % (2 * math.pi)
Out[63]: '6.283185'

In [64]: "%10.5f" % (2 * math.pi)
Out[64]: '   6.28319'
```

Switch to different formats:

```
In [5]: f"{2 * math.pi:e}"
Out[5]: '6.283185e+00'

In [6]: f"{2 * math.pi:g}"
Out[6]: '6.28319'
```