12112627 李乐平

Homework 2 Part 1

1- 5 BCAAC Actually, T[20:21] in problem 5 is a space character ' ' rather than character 'I'.

6-10 DCCBB

11-15 CABBD

16-20 CDAAA

21-25 CADBA

26-30 CDBDD

Furthermore, there are other bugs in these problems, such as Chinese quotation marks and uppercase keywords. Any of these bugs can cause exception and the program will not print the output given in the choices correctly. Hope subsequent homework can avoid these bugs.

Homework 2 Part 2

How to submit your homework:

After you complete this notebook:

Print the page, then print as pdf

Then upload the .pdf file to BlackBoard.

Defining functions

Let's write a very simple function that converts a proportion to a percentage by multiplying it by 100. For example, the value of `to_percentage(.5)` should be the number 50. (No percent sign.)

A function definition has a few parts.

`def`

It always starts with `def` (short for **def**ine):

```
def
```

Name

Next comes the name of the function. Let's call our function `to_percentage`.

```
def to_percentage
```

Signature

Next comes something called the *signature* of the function. This tells Python how many arguments your function should have, and what names you'll use to refer to those arguments in the function's code. `to_percentage` should take one argument, and we'll call that argument `proportion` since it should be a proportion.

```
def to_percentage(proportion)
```

We put a colon after the signature to tell Python it's over.

```
def to_percentage(proportion):
```

Documentation

Functions can do complicated things, so you should write an explanation of what your function does. For small functions, this is less important, but it's a good habit to learn from the start. Conventionally, Python functions are documented by writing a triple-quoted string:

```
def to_percentage(proportion):
    """Converts a proportion to a percentage."""
```

Body

Now we start writing code that runs when the function is called. This is called the *body* of the function. We can write anything we could write anywhere else. First let's give a name to the number we multiply a proportion by to get a percentage.

```
def to_percentage(proportion):
    """Converts a proportion to a percentage."""
    factor = 100
```

`return`

The special instruction `return` in a function's body tells Python to make the value of the function call equal to whatever comes right after `return`. We want the value of `to_percentage(.5)` to be the proportion .5 times the factor 100, so we write:

```
def to_percentage(proportion):
    """Converts a proportion to a percentage."""
    factor = 100
    return proportion * factor
```

Question: Define `to_percentage` in the cell below. Call your function to convert the proportion .2 to a percentage. Name that percentage `twenty_percent`.

In [162]:

```
def to_percentage(proportion):
    """ Converts a proportion to a percentage """
    factor = 100
    return proportion *  factor

twenty_percent = to_percentage(.2)
twenty_percent
```

Out[162]:

```
20.0
```

Like the built-in functions, you can use named values as arguments to your function.

Question: Use `to_percentage` again to convert the proportion named `a_proportion` (defined below) to a percentage called `a_percentage`.

Note: You don't need to define `to_percentage` again! Just like other named things, functions stick around after you define them.

In [164]:

```
a_proportion = 2**(.5) / 2
a_percentage = to_percentage(a_proportion)
a_percentage
```

Out[164]:

```
70.71067811865476
```

As we've seen with the built-in functions, functions can also take strings (or arrays, or tables) as arguments, and they can return those things, too.

Question: Define a function called `disemvowel`. It should take a single string as its argument. (You can call that argument whatever you want.) It should return a copy of that string, but with all the characters that are vowels removed. (In English, the vowels are the characters "a", "e", "i", "o", and "u".)

Hint: To remove all the "a"s from a string, you can use `that_string.replace("a", "")`. And you can call `replace` multiple times.

In [2]:

```
def disemvowel(a_string):
    for i in ("a","e","i","o","u","A","E","I","O","U"):
        a_string = a_string.replace(i,"")
    return a_string

# An example call to your function.  (It's often helpful to run
# an example call from time to time while you're writing a function,
# to see how it currently works.)
disemvowel("Can you read this without vowels?")
```

Out[2]:

```
'Cn y rd ths wtht vwls?'
```

Calls on calls on calls

Just as you write a series of lines to build up a complex computation, it's useful to define a series of small functions that build on each other. Since you can write any code inside a function's body, you can call other functions you've written.

If a function is a like a recipe, defining a function in terms of other functions is like having a recipe for cake telling you to follow another recipe to make the frosting, and another to make the sprinkles. This makes the cake recipe shorter and clearer, and it avoids having a bunch of duplicated frosting recipes. It's a foundation of productive programming.

For example, suppose you want to count the number of characters *that aren't vowels* in a piece of text. One way to do that is this to remove all the vowels and count the size of the remaining string.

Question: Write a function called `num_non_vowels` . It should take a string as its argument and return a number. The number should be the number of characters in the argument string that aren't vowels.

Hint: The function `len` takes a string as its argument and returns the number of characters in it.

In [3]:

```
def num_non_vowels(a_string):
    """The number of characters in a string, minus the vowels."""
    return len(disemvowel(a_string))

# Try calling your function yourself to make sure the output is what
# you expect.
num_non_vowels("Hymn of the Holy Land")
```

Out[3]:

```
17
```

Loops and condition

Question: Find all possible 4-digit numbers satisfying:

$$abcd \times e = dcba$$

the initial number can not be 0, and a, b, c, d are different numbers.

Print them out.

Hint: you can transfer between string and integer:

```
>>str(1234)
[out]: "1234"
>>int("1234")
[out]: 1234
```

In [160]:

```
for i in range(1,10):
    for j in range(0,10):
        if(j == i):
            continue
        for k in range(0,10):
            if(k == i or k == j):
                continue
            for p in range(0,10):
                if(p == i or p == j or p == k):
                    continue
                for fac in range(1,10):
                    if((1000*i + 100*j + 10*k + p) * fac == 1000*p + 100*k + 10*j + i):
                        print(1000*i + 100*j + 10*k + p)
                    if(1000*i + 100*j + 10*k + p >= 10000):
                        break
```

```
1089
2178
```

Question: Write a function `summation` that evaluates the following summation for $n \geq 1$:

$$\sum_{i=1}^{n} i^3 + 3i^2$$

In [25]:

```
def summation(n):
    """Compute the summation i^3 + 3 * i^2 for 1 <= i <= n."""
    """ Reduce Time Complexity by Using Formula:
        1 + 2^3 + ... + n^3 = [n(n+1)/2]^2
        1 + 2^2 + ... + n^2 = n(n+1)(2n+1)/6
        Time Complexity: O(1)
    """
    return int(n * (n + 1) * (2 * n + 1) / 2 + n * n * (n + 1) * (n + 1) / 4)

def summation_N(n):
    """Compute the summation i^3 + 3 * i^2 for 1 <= i <= n."""
    """ Calculate by Loops:
        Time Complexity: O(n)
    """
    ans = 0
    for i in range(1,n + 1):
        ans += i*i*i + 3*i*i
    return ans

# test your function:
print(summation(500))
print(summation_N(500))
```

```
15812937750
15812937750
```

Question: Recall the formula for population variance below:

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}$$

Complete the functions below to compute the population variance of `population`, an list of numbers. For this question, **do not use built in NumPy functions, such as `np.var`.**

In [5]:

```
def mean(population):
    """
    Returns the mean of population (mu)

    Keyword arguments:
    population -- a list of numbers
    """
    # Calculate the mean of a population
    ans = 0
    for i in range(0,len(population)):
        ans += population[i] / len(population)
    return ans

def variance(population):
    """
    Returns the variance of population (sigma squared)

    Keyword arguments:
    population -- a list of numbers
    """
    # Calculate the variance of a population
    return mean((population - mean(population)) ** 2)
```

In [9]:

```
# to test the function, at first we generate a list of variables
#that follow a normal distribution:
import numpy as np
population_generated = list(np.random.normal(1,0.3,500))
# test
# note that in each run, the results will differ a bit,
#because the population we generated each time is ramdom samples

print("The mean of the population is ", mean(population_generated))
print("The variance of the population is ", variance(population_generated))
```

```
The mean of the population is  0.9994264422637776
The variance of the population is  0.08992627637453299
```

Question: The **GPA**, or **Grade Point Average**, is a number that indicates how well or how high you scored in your courses on average. Based on SUSTech's grading system, the scores, Grade and GPA have the following relationship:

Grade	GPA	Score
A+	4.00	97~100
A	3.94	93~96
A-	3.85	90~92
B+	3.73	87~89
B	3.55	83~86
B-	3.32	80~82
C+	3.09	77~79
C	2.78	73~76
C-	2.42	70~72
D+	2.08	67~69
D	1.63	63~66
D-	1.15	60~62
F	0	<60

Define a function `Score2GradeGPA` to convert a score (of total 100) to Grade and GPA according to the table above:

In [1]:

```python
def Score2GradeGPA(score):
    """
    Returns the Grade and GPA into a tuple.

    Keyword arguments:
    score -- an integer or float value in the range [0,100]
    """
    scores = [96.5,92.5,89.5,86.5,82.5,79.5,76.5,72.5,69.5,66.5,62.5,59.5]
    GPAs = [4.00,3.94,3.85,3.73,3.55,3.32,3.09,2.78,2.42,2.08,1.63,1.15]
    for i in range(12):
        if (score >= scores[i]):
            return GPAs[i]
    return 0.00

# test
# here is the scores of 10 students:
Scores = [50, 60.1, 71, 74.2, 99, 88, 79.5, 89, 83, 94]
# apply the function:
[Score2GradeGPA(score) for score in Scores]
```

Out[1]:

```
[0.0, 1.15, 2.42, 2.78, 4.0, 3.73, 3.32, 3.73, 3.55, 3.94]
```

Set

The **Jaccard index**, also known as the **Jaccard similarity coefficient**, is a statistic used for gauging the similarity and diversity of sample sets. It was developed by Paul Jaccard, and independently formulated again by T. Tanimoto. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Now we have list A_l and list B_l:

In [57]:

```
A_l = ['h','e','r','a','m','m','y','w','a','r','d','i','s','a','n','a','w','a','r','d',
      'p','r','e','s','e','n','t','e','d','b','y','t','h','e','e','c','o','r','d','i',
      'n','g','c','a','d','e','m','y','t','o','r','e','c','o','g','n','i','z','e','a',
      'c','h','i','e','v','e','m','e','n','t','i','n','t','h','e','m','u','s','i','c',
      'i','n','d','u','s','t','r','y','h','e','t','r','o','p','h','y','d','e','p','i']

B_l = ['s','o','r','i','g','i','n','a','l','a','n','u','a','r','y','d','a','t','e','d',
      'u','e','t','o','t','h','e','i','m','p','a','c','t','o','f','t','h','e','p','a',
      'n','d','e','m','i','c','o','n','t','h','e','m','u','s','i','c','i','n','d','u',
      's','t','r','y','i','n','a','n','d','a','r','o','u','n','d','t','h','e','o','s',
      'n','g','e','l','e','s','o','n','v','e','n','t','i','o','n','e','n','t','e','r']
```

Question: Try to define set A and set B based on the corresponding lists A_l and B_l, for set A (or B), the elements are unique members in list A_l (or B_l):

In [65]:

```
A = set()
B = set()
for i in A_l:
    A |= {i}
for i in B_l:
    B |= {i}
```

Question: Try to calculate the Jaccard index of A and B, $J(A,B)$:

In [67]:

```
def Jaccard_Index(A,B):
    return len(A & B) / len(A | B)

Jaccard_Index(A,B)
```

Out[67]:

```
0.7727272727272727
```

String

The Grammy Award, is an award presented by the Recording Academy to recognize achievement in the music industry. The trophy depicts a gilded gramophone. The annual award ceremony features performances by prominent artists and presentation of awards that showcase achievements made by industry recording artists. The first Grammy Awards ceremony was held on May 4, 1959, to honor the musical accomplishments of performers for the year 1958. After the 2011 ceremony, the Academy overhauled many Grammy Award categories for 2012. The 63rd Annual Grammy Awards were held on March 14, 2021 (after it was postponed from its original January 31, 2021 date due to the impact of the COVID-19 pandemic on the music industry), in and around the Los Angeles Convention Center.

We have the full transcripts of this ceremony, see attached file "Transcripts_63RD_ANNUAL_GRAMMY_AWARDS_2021.txt".

Question: Read the text from the file, extract all words and save them into a list called **word_list**, remove all non-alphabet characters from each word.

In [3]:

```
# coding=utf-8
file = open("Transcripts_63RD_ANNUAL_GRAMMY_AWARDS_2021.txt",encoding = 'utf-8')
word_list = []
tmpList = ''
for i in file.read():
    if(i.isalpha()):
        tmpList = ''.join([tmpList,i])
    else:
        if(len(tmpList) == 0 or not i.isspace()):
            continue
        word_list.append(tmpList)
        tmpList = ''
word_list
```

Out[3]:

```
['WELCOME',
 'TO',
 'THE',
 'rd',
 'ANNUAL',
 'GRAMMY',
 'AWARD',
 'IM',
 'TREVOR',
 'NOAH',
 'AND',
 'IM',
 'YOUR',
 'HOST',
 'AS',
 'WE',
 'CELEBRATE',
```

Question: Count the frequency of each unique word in **word_list**, and save the results into a dictionary called **word_count**: each key in the dictionary is a word, and the corresponding value is the frequency.

In [4]:

```
word_count = {}
for i in word_list:
    if(i in word_count):
        word_count[i] += 1
    else:
        word_count[i] = 1
word_count
```

Out[4]:

```
{'WELCOME': 21,
 'TO': 447,
 'THE': 679,
 'rd': 8,
 'ANNUAL': 1,
 'GRAMMY': 86,
 'AWARD': 30,
 'IM': 161,
 'TREVOR': 4,
 'NOAH': 2,
 'AND': 543,
 'YOUR': 55,
 'HOST': 3,
 'AS': 39,
 'WE': 147,
 'CELEBRATE': 6,
 'LAST': 14,
```

Question: Based on **word_count**, what's the top 10 frequently used words in this ceremony? Try to print the top 10 words and their frequency below:

```
For example:
    word     frequency
    of       100
    happy    20
    ...
```

In [6]:

```
most_frequent = sorted(word_count.items(), key = lambda x: -x[1])
print("word     frequency")
for i in most_frequent[:10]:
    print(i[0],"\t",i[1])
```

```
word     frequency
THE      679
I        569
AND      543
TO       447
YOU      434
A        378
ME       243
OF       233
MY       233
IT       208
```

Question: For the top 100 frequently used words, what's top 5 frequently used alphabets? Try to print them out with the percentage they appear in the 100 words:

```
 For example:
    alphabet percentage
    a    10.25%
    b    20.01%
    ...
```

In [145]:

```
alpha_count = {}
tot_len = 0
for k in most_frequent[:100]:
    tot_len += len(k[0])
    for i in k[0]:
        if(i.lower() in alpha_count):
            alpha_count[i.lower()] += 1
        else:
            alpha_count[i.lower()] = 1
for i in alpha_count:
    alpha_count[i[0]] /= tot_len
frequent_alpha = sorted(alpha_count.items(),key = lambda x:-x[1])
print("alphabet percentage")
for i in frequent_alpha[:5]:
    print(i[0],"\t",round(i[1] * 100,2),"%")
```

```
alphabet percentage
e        10.86 %
a        9.43 %
t        9.14 %
o        8.57 %
h        6.29 %
```

Flip a coin

Python's built-in `random` module, can do the pseudorandom number generation. You can get more info. about it here:https://docs.python.org/3/library/random.html (https://docs.python.org/3/library/random.html)

You can import the module:

In [72]:

```
import random
```

The method `random` in the module `random` will return a random values between the interval 0 and 1. For example:

In [149]:

```
random.random() # you can run this code multiple times to see how it works.
```

Out[149]:

```
0.4367744321253153
```

Question: Define a function called `flip_coin` that return 0 or 1. Each time we call the function, it will return 0 with probability $p = 0.2$, and or return 1 with probability $1 - p = 0.8$. Try to use the method `random.random()` :

In [152]:

```
def flip_coin():
    """
    Return 0 with probability 0.2, or return 1 with probability 0.8
    """
    p = 0.2
    return random.random() >= p
```

In [159]:

```
# this code is used for verifying your flip coin function
flip_times = 10000
flip_coin_records = [flip_coin() for i in range(flip_times)]
flip_1 = sum(flip_coin_records)
print ("Flip coins {0} times, in which {1:.3%} are 0, and {2:.3%} are 1.".format(
    flip_times,1-flip_1/flip_times, flip_1/flip_times))
```

```
Flip coins 10000 times, in which 20.020% are 0, and 79.980% are 1.
```