## Homework 1

*For all questions, please use* ***only*** *python, numpy and matplotlib (if needed for plotting).* ***Do not use pandas or other higher level libraries.***

Click [here](http://github.com/ChadKwong) to view the repository folder containing the full python file

**Q1. Please fill in an explanation of each function and an example of how to use it below.**

**List**:

append() – This function will add the specified parameter to the end of the specified list as a single element.

X = [1,2,3]

Y = [4,5,6]

X.append(Y)

**X => [1,2,3,[4,5,6]]**

extend() – This function takes input that is an iteration and adds the elements within the input to the end of the specified list as separate elements

X = [1,2,3]

Y = [4,5,6]

X.extend(Y)

**X => [1,2,3,4,5,6]**

index()– This function returns the index of the list containing the specified input.

X = list(range(10,20))

X.index(15) => 5

index(value, integer) – This function will list the index of the first occurrence of the first input after the index specified by the second output

x = [1,2,3,2,1,2,3,2]

**x.index(3,4) => 6**

insert(position,element) – This function will insert the element input at the specified position

x = [1,2,3,2,1,2,3,2]

**x.insert(4,'midpoint') => [1,2,3,2,’midpoint’,1,2,3,2]**

remove() – This function deletes the first instance of the specified element

x=[1,2,3,2,1,2,3,2]

**x.remove(2) => [1,3,2,1,2,3,2]**

pop() – This function will remove and store the element at the position specified by the input. Giving no input will return the last element in the selected list

x = ['a','b','c','d','e','f','g']

**y = x.pop() => g**

**z = x.pop(3) => d**

count(element) – This function will return a count of the specified element

x = ['a','b','c','d','e','f','g','a','c','a','g','e','f','f','f','f','d']

x.count('f') => 5

reverse() – This function will reverse the order of the specified list

x = [1,2,3,4,5]

**x.reverse() => [5,4,3,2,1]**

sort() – This function will sort the specified list. By default it will sort lists alphabetically for strings and in ascending order for numerical data types

num = [3,6,4,2,8,9,6,3,17,8,90]

words = ['hi', 'there', 'how', 'do', 'you', 'sort', 'again', '?']

**num.sort() => [2, 3, 3, 4, 6, 6, 8, 8, 9, 17, 90]**

**words.sort() => ['?', 'again', 'do', 'hi', 'how', 'sort', 'there', 'you']**

[1]+[1] = **[1,1],** This adds the contents of the lists together

[2]\*2 = **[2,2],** This will duplicate the list and add the contents together

[1,2][1:] = **[2]**, This lists the items in the list after the index preceding the colon

[x for x in [2,3]] = **[2,3]**, This will return each item in the list

[x for x in [1,2] if x ==1] = **[1]**, this will return each item in the list as long as the item is equal to 1

[y\*2 for x in [[1,2],[3,4]] for y in x] = **[2,4,6,8]**, This is an iterative arithmetic that loops through the items in the parent list, then loops through the items in each of the child lists and returns the value in each child list multiplied by 2

A = [1] – This will create a list containing 1 element equal to 1 and store it to the variable A

**Tuple:**

count() –This function will return the number of times that the input appears in the tuple.

myTuple = (1,3,5,4,5)

**myTuple.count(5) => 2**

index() –This function will search through the tuple for the input and return its position.

myTuple = (1,3,5,4,5)

**myTuple.index(4) => 3**

build a dictionary from tuples – A Dictionary can be constructed from a list of tuples using the dict() function

myTuple = ((1,'yes'), (2,'the'), (3,'index'), (4,'is'), (5,'one'))

**dict((x,y) for x,y in myTuple) => {1: 'yes', 2: 'the', 3: 'index', 4: 'is', 5: 'one'}**

unpack tuples –You can unpack the contents of a tuple by setting it equal to a tuple of the same length containing variables to store the contents of the target tuple

myTuple = ('yes','no','maybe','so','damn')

(x,y,z,a,b) = myTuple

**X => ‘yes’**

**Dicts:**

a\_dict = {'I hate':'you', 'You should':’leave’} –This creates a dictionary called “a\_dict” containing 2 key value pairs where the keys are ‘I hate’ and ‘You should’, and the corresponding values are ‘you’ and ‘leave’.

keys() –This function will return the keys of the dictionary as a view object

myDict = {1: 'yes', 2: 'the', 3: 'index', 4: 'is', 5: 'one'}

**myDict.keys() => dict\_keys([1, 2, 3, 4, 5])**

items() –This function will return the items of the dictionary as a view object, where each item is a key value pair

myDict = {1: 'yes', 2: 'the', 3: 'index', 4: 'is', 5: 'one'}

**myDict.items() => dict\_items([(1, 'yes'), (2, 'the'), (3, 'index'), (4, 'is'), (5, 'one')])**

values() –This function will return the values of the dictionary as a view object. To check for a specific value, a simple **in** condition to return a boolean value.

myDict = {1: 'yes', 2: 'the', 3: 'index', 4: 'is', 5: 'one'}

**myDict.values() => dict\_values(['yes', 'the', 'index', 'is', 'one'])**

has\_key() –This function will return a boolean value based on whether the input exists as a key within the specified dictionary. With python 3, this function was removed and we should use the in operator instead

‘never’ in a\_dict –This will return a bool value based on whether ‘never’ exists as a key in a\_dict

del a\_dict['me'] –This will delete the item using the key ‘me’ from the dictionary a\_dict

a\_dict.clear() –This function will delete the contents of a dictionary

**Sets:**

add() –This function will add an element to the specified set

x = {'first', 'second'}

x.add('third')

**x => {'first', 'second', 'third'}**

clear() –This function will remove all elements from the specified set

x = {'first', 'second'}

x.clear()

**x => set()**

copy() –This function returns a duplicate of the specified set

x = {'first', 'second'}

**x.copy() => {‘first’, ‘second’}**

difference() –This function will return a set containing items that are present in the specified set but that are not present in the input

x = {"apple", "banana", "cherry"}

y = {"google",'microsoft', 'apple'}

z = x.difference(y)

**z => {'banana', 'cherry'}**

discard() –This function removes the input item from the specified set

x = {'first', 'second', 'third'}

x.discard('second')

**x => {'first', 'third'}**

intersection() –This function will return a set containing common items between the input set and the specified set

x = {"apple", "banana", "cherry"}

y = {"google",'microsoft', 'apple'}

z = x.intersection(y)

**z => {“apple”}**

issubset() –This function will return a bool value based off of whether the specified set exists completely within the input set

x = {'first', 'second', 'third', 'fourth','fifth'}

y = {'second', 'third'}

**y.issubset(x) => True**

pop() –This function will remove and store a random element of the specified set.

x = ['a','b','c','d','e','f','g']

**y = x.pop() => ‘a’**

**z = x.pop() => ‘d’**

**aa = x.pop() => ‘b’**

remove() –This function will remove the element that matches the specified input.

x = {'first', 'second', 'third', 'fourth','fifth'}

x.remove('second')

**x => {‘first’, ‘third’, ‘fourth’,’fifth’}**

union() –This function will add the input to the specified set

x = {"apple", "banana", "cherry"}

y = {"google",'microsoft', 'apple'}

**x.union(y) => {'apple', 'banana', 'cherry', 'google', 'microsoft'}**

update() –This function will merge the input to the specified set and create a new set as the output

setA = {'apple', 'orange', 'kiwi'}

setB = {'mango', 'papaya'}

setA.update(setB)

**setA => {'apple', 'kiwi', 'mango', 'orange', 'papaya'}**

**Strings:**

capitalize() –This function will capitalize the first character within the specified string

txt = ‘where am I’

x = txt.capitalize()

**x => ‘Where am I’**

casefold() –This function will convert all the characters of the specified string to lower case

x = ‘AHHHHHHHH’

x.casefold()

**x => ‘ahhhhhhhh’**

center() –This function will place the characters of the specified string in the center position of a new string of length specified by the first input where the surrounding characters are the second input.

myString = 'yes'

**myString.center(11,'-') => '----yes----'**

count() –This function will return a count of the occurrence of the input within the specified string

txt = 'Hello World'

**txt.count('o') => 2**

encode() –This function will return the specified string encoded in the specified input. By default, this input is UTF8

myString = 'åå©'

**myString.encode() => b'\xc3\xa5\xc3\xa5\xc2\xa9'**

find() –This function will return the index position of the input value within the specified string

txt = 'Hello World'

**txt.find('W') => 6**

partition() –This function will locate the input in the specified string and split the latter into a tuple containing 3 string elements: contents to the left of the input, the input, and contents to the right of the input

txt = "where am i"

x = txt.partition('am')

**x => ('where ', 'am', ' i')**

replace() –This function will return the specified string with the first input replaced by the second input

txt = ‘hallo’

**txt.replace(‘a’,’e’) => ‘hello**

split() –This function will split the specified string based off of the input. The output will be a list of string elements.

entry = ‘1/2/3/4/5’

**entry.split(‘/’) => ['1', '2', '3', '4', '5']**

title() –This function will capitalize the first letter of each word within the specified string

txt = ‘where am I’

**txt.title() => ‘Where Am I’**

zfill() –This function adds n number of zeros to the beginning of the specified string where n is equal to the input. If n is less than the length of the string, no zeros will be added.

txt = "My name is Ståle"

**txt.zfill(17) => "0My name is Ståle"**

*Ok enough by me do the rest on your own!*  Use dir() to get built- in functions\*\*\*

**from collections import Counter**

Fill in relevant functions yourself…

Elements –This function returns an iterative list of the contents of the counter object

c = Counter(a=4, b=2, c=0, d=-2)

**sorted(c.elements()) => ['a', 'a', 'a', 'a', 'b', 'b']**

Most\_common –This function returns a list of the most common elements within the counter object. If no input is specified, then every element will be returned in the list.

**Counter(‘abracadabra’).most\_common(3) => [(a,5),(b,2),(r,2)]**

Subtract –This function is used to subtract elements between two counter objects

a = Counter(a=4, b=2, c=0, d=-2)

b = Counter(a=2,b=3,c=4,d=5)

b.subtract(a)

**b => Counter({'a': -2, 'b': 1, 'c': 4, 'd': 7})**

Total –This function will return the sum of the number of elements within the counter object

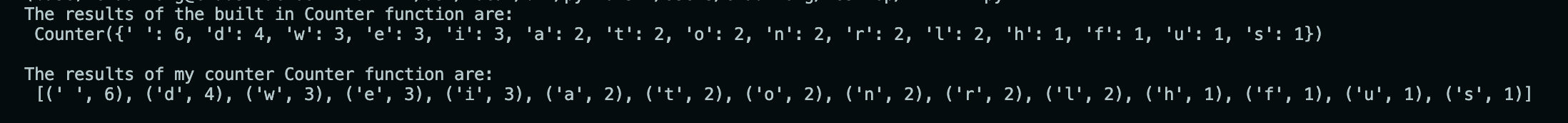
**from itertools import \*** (**Bonus**: this one is optional, but recommended)

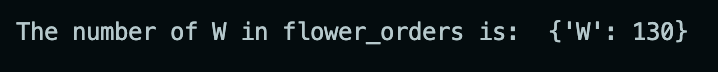
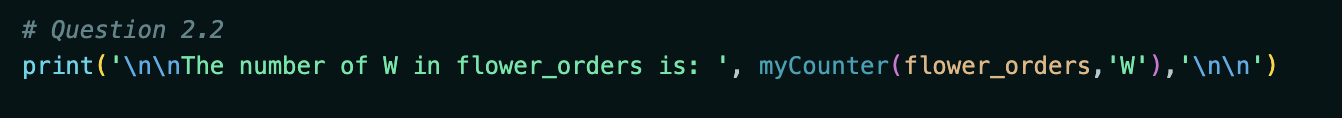
Fillin relevant functions yourself…

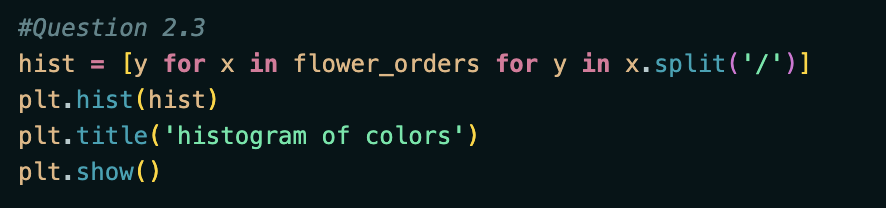
**Q2.**

|  |  |
| --- | --- |
| flower\_orders=['W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R/B','W/R','W/R','W/R','W/R','W/R','W/R','W/R','W/R','W/R','W/R','W/R','W/R','W/R','W/R','W/R','W/R','R/V/Y','R/V/Y','R/V/Y','R/V/Y','R/V/Y','R/V/Y','R/V/Y','R/V/Y','R/V/Y','R/V/Y','W/R/V','W/R/V','W/R/V','W/R/V','W/R/V','W/R/V','W/R/V','W/R/V','W/R/V','W/R/V','W/N/R/V','W/N/R/V','W/N/R/V','W/N/R/V','W/N/R/V','W/N/R/V','W/N/R/V','W/N/R/V','W/R/B/Y','W/R/B/Y','W/R/B/Y','W/R/B/Y','W/R/B/Y','W/R/B/Y','B/Y','B/Y','B/Y','B/Y','B/Y','R/B/Y','R/B/Y','R/B/Y','R/B/Y','R/B/Y','W/N/R/B/V/Y','W/N/R/B/V/Y','W/N/R/B/V/Y','W/N/R/B/V/Y','W/N/R/B/V/Y','W/G','W/G','W/G','W/G','R/Y','R/Y','R/Y','R/Y','N/R/V/Y','N/R/V/Y','N/R/V/Y','N/R/V/Y','W/R/B/V','W/R/B/V','W/R/B/V','W/R/B/V','W/N/R/V/Y','W/N/R/V/Y','W/N/R/V/Y','W/N/R/V/Y','N/R/Y','N/R/Y','N/R/Y','W/V/O','W/V/O','W/V/O','W/N/R/Y','W/N/R/Y','W/N/R/Y','R/B/V/Y','R/B/V/Y','R/B/V/Y','W/R/V/Y','W/R/V/Y','W/R/V/Y','W/R/B/V/Y','W/R/B/V/Y','W/R/B/V/Y','W/N/R/B/Y','W/N/R/B/Y','W/N/R/B/Y','R/G','R/G','B/V/Y','B/V/Y','N/B/Y','N/B/Y','W/B/Y','W/B/Y','W/N/B','W/N/B','W/N/R','W/N/R','W/N/B/Y','W/N/B/Y','W/B/V/Y','W/B/V/Y','W/N/R/B/V/Y/G/M','W/N/R/B/V/Y/G/M','B/R','N/R','V/Y','V','N/R/V','N/V/Y','R/B/O','W/B/V','W/V/Y','W/N/R/B','W/N/R/O','W/N/R/G','W/N/V/Y','W/N/Y/M','N/R/B/Y','N/B/V/Y','R/V/Y/O','W/B/V/M','W/B/V/O','N/R/B/Y/M','N/R/V/O/M','W/N/R/Y/G','N/R/B/V/Y','W/R/B/V/Y/P','W/N/R/B/Y/G','W/N/R/B/V/O/M','W/N/R/B/V/Y/M','W/N/B/V/Y/G/M','W/N/B/V/V/Y/P'] |  |

1. Build your own counter object, then use the built-in Counter() and confirm they have the same values.Text

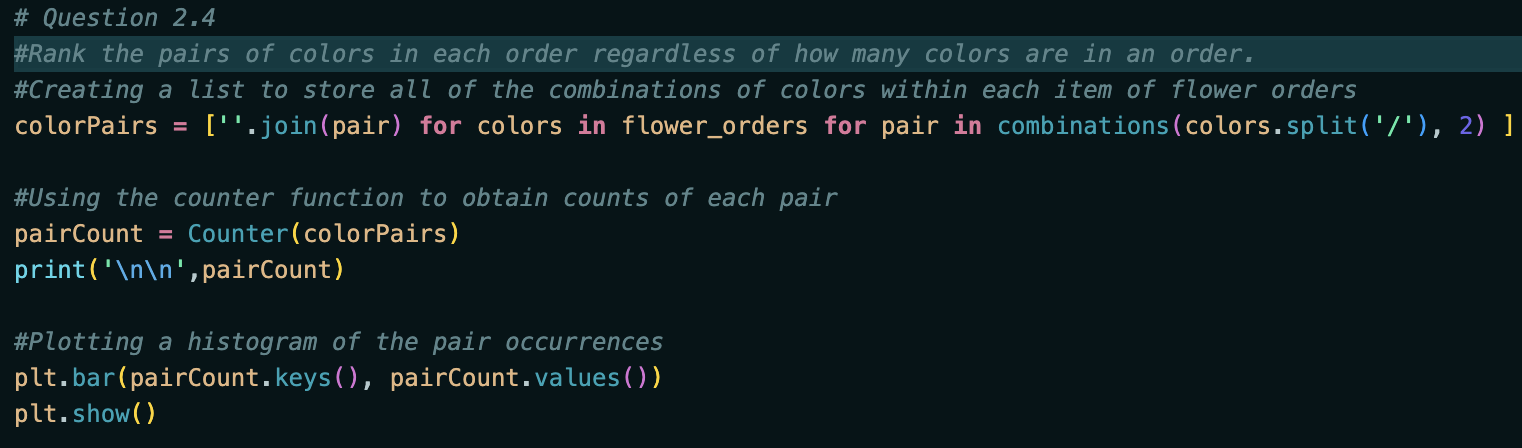
   Description automatically generated
2. Count how many objects have color W in them.

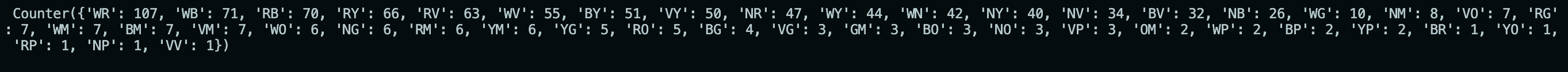


1. Make histogram of colorsChart, histogram

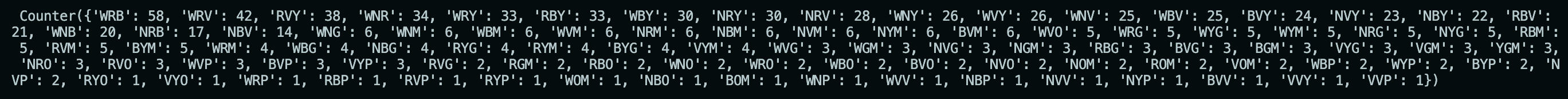
   Description automatically generated

# Hint from JohnP - Itertools has a permutation function that might help with these next two.

1. Rank the pairs of colors in each order regardless of how many colors are in an order.

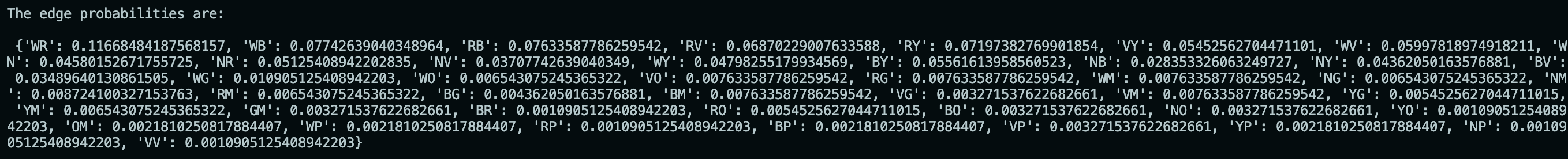


1. Rank the triplets of colors in each order regardless of how many colors are in an order.

Text

Description automatically generated

1. Make a dictionary with key=”color” and values = “what other colors it is ordered with”.Text

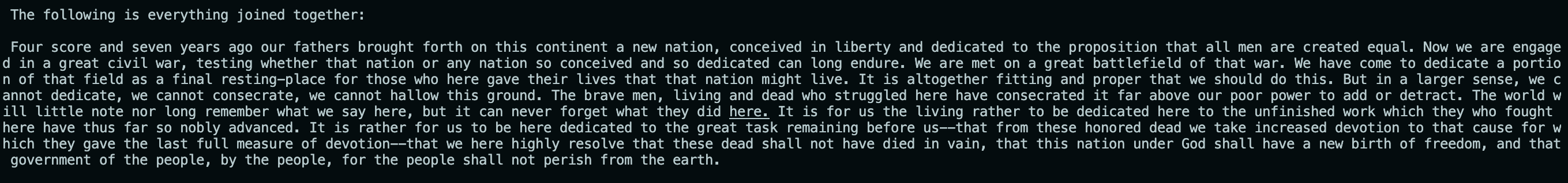
   Description automatically generated
2. Make a graph showing the probability of having an edge between two colors based on how often they co-occur. (a numpy square matrix)
3. Make 10 business questions related to the questions we asked above.
   1. What are the top 5 individual colors?
   2. What are the top 5 color pairings?
   3. What are the top 5 color triplets?
   4. How often do we expect white and red flowers with violet?
   5. Are our orders larger or smaller on average?
   6. What colors are ideal for single pairings?
   7. What colors are ideal for triplets?
   8. Is it worth increasing the limit to the number of flowers?
   9. Is it worth increasing the required amount of flowers for an order?
   10. Are there any colors worth discarding from inventory?

**Q3.**

|  |
| --- |
| dead\_men\_tell\_tales =  ['Four score and seven years ago our fathers brought forth on this', 'continent a new nation, conceived in liberty and dedicated to the', 'proposition that all men are created equal. Now we are engaged in', 'a great civil war, testing whether that nation or any nation so', 'conceived and so dedicated can long endure. We are met on a great', 'battlefield of that war. We have come to dedicate a portion of', 'that field as a final resting-place for those who here gave their', 'lives that that nation might live. It is altogether fitting and', 'proper that we should do this. But in a larger sense, we cannot', 'dedicate, we cannot consecrate, we cannot hallow this ground.', 'The brave men, living and dead who struggled here have consecrated', 'it far above our poor power to add or detract. The world will', 'little note nor long remember what we say here, but it can never', 'forget what they did here. It is for us the living rather to be', 'dedicated here to the unfinished work which they who fought here', 'have thus far so nobly advanced. It is rather for us to be here', 'dedicated to the great task remaining before us--that from these', 'honored dead we take increased devotion to that cause for which', 'they gave the last full measure of devotion--that we here highly', 'resolve that these dead shall not have died in vain, that this', 'nation under God shall have a new birth of freedom, and that', 'government of the people, by the people, for the people shall', 'not perish from the earth.'] |

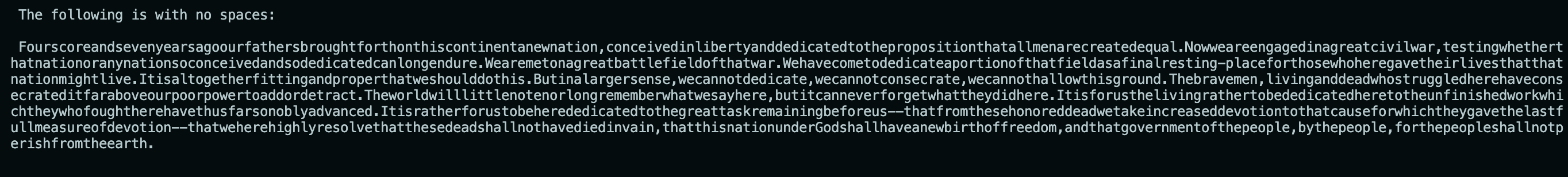
1. Join everything

Text

Description automatically generated

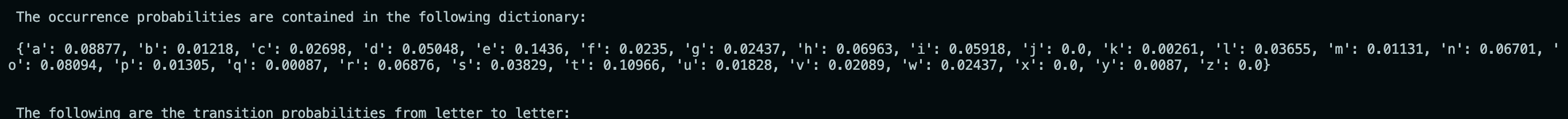
1. Remove spaces

A screenshot of a computer

Description automatically generated with medium confidence

1. Occurrence probabilities for letters

Text

Description automatically generated

1. Tell me transition probabilities for every pair of letters

Text

Description automatically generatedText

Description automatically generated with medium confidence

1. Make a 26x26 graph of 4. in numpy

Text

Description automatically generatedGraphical user interface, text

Description automatically generated

1. plot graph of transition probabilities from letter to letter

Text

Description automatically generatedA picture containing graphical user interface

Description automatically generated

Unrelated:

7. Flatten a nested list

Cool intro python resources:

<https://thomas-cokelaer.info/tutorials/python/index.html>