Joaquin Dominguez

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.***

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

List:

append()

Append adds items to a list one-by-one to the end.

```
>>> groceries = [‘milk’, ‘eggs’, ‘bread’]
>>>groceries.append(‘bacon’)
>>> groceries
[‘milk’, ‘eggs’, ‘bread’, ‘bacon’]
```

extend()

Extend merges one list to another.

```
>>> groceries = [‘milk’, ‘eggs’, ‘bread’]
>>> groceries2 = [‘brownies’, ‘cookies’, ‘chocolate’]
>>> groceries.extend(Groceries2)
>>> groceries
[‘milk’, ‘eggs’, ‘bread’, ‘brownies’, ‘cookies’, ‘chocolate’]
```

index()

Index finds an item’s offset by value

```
>>>  groceries = [‘milk’, ‘eggs’, ‘bread’]
>>> groceries.index(‘eggs’)
1
```

index(value, integer)

Index can also take starting and ending indices as arguments to search in a particular section of the list.

```
>>> numbers = [1,2,3,4,5,6,7]
>>> numbers.index(5,3) # Index of 5 from 3rd index till end
4
```

insert(position)

Whereas append() only adds an item to the end of a list, insert() adds an item before any offset in the list.

```
 >>> groceries = [‘milk’, ‘eggs’, ‘bread’]
>>>groceries.insert(1, ‘bacon’)
>>> groceries
[‘milk’, ‘bacon’, ‘eggs’, ‘bread’]
```

remove()

```
 >>> groceries = [‘milk’, ‘eggs’, ‘bread’]
>>>groceries.remove(‘eggs’)
>>> groceries
[‘milk’,  ‘bread’]
```

pop()

Pop() gets an item from a list and deletes it from the list at the same time.

```
 >>> groceries = [‘milk’, ‘eggs’, ‘bread’]
>>>groceries.pop(1)
‘eggs’
>>> groceries
[‘milk’, ‘bread’]
```

count()

Count() counts how many times a particular value occurs in a list.

```
 >>> groceries = [‘milk’, ‘eggs’, ‘bread’]
>>>groceries.count(‘eggs’)
1
>>> groceries.count(‘bacon’)
0
```

reverse()

This function reverses the elements of the list.

```
 >>> groceries = [‘milk’, ‘eggs’, ‘bread’]
>>> groceries.reverse()
>>> groceries
[‘bread’, ‘eggs’, ‘milk’]
```

sort()

Sorts items in a list by their values rather than their offsets.

```
>>> groceries = [‘milk’, ‘eggs’, ‘bread’]
>>> groceries.sort()
>>> groceries
[‘bread’, ‘eggs’, ‘milk’]
```

[1]+[1]
Output : [1,1]

[2]*2
Output : [2, 2]

[1,2][1:]
Output : [2]

[x for x in [2,3]]
Output : [2, 3]

[x for x in [1,2] if x ==1]
Output : [1]

[y*2 for x in [[1,2],[3,4]] for y in x]
Output : [2, 4, 6, 8]

A = [1]
Output :

Tuple:

count()
```
>>> tup_ex = (1,2,2,3,3,3,4,4,4,4,5)
>>> tup_ex.count(3)
3
```

index()
```
>>> tup_ex = (1,2,2,3,3,3,4,4,4,4,5)
>>> tup_ex.index(6)
4
```

build a dictionary from tuples
```
>>> tup = ((1, 'John'), (2,'Henry'), (3, 'Bartholomew'))
>>> tup_dict = dict((x,y) for x,y in tup)
>>> tup_dict
{1: 'John', 2: 'Henry', 3: 'Bartholomew'}
```

unpack tuples
```
>>> groceries = ('milk', 'bread', 'eggs')
>>> a,b,c = groceries
```

>>> a

‘millk’

>>> b

‘bread’

Dicts:

a_dict = {'I hate':'you', 'You should':’leave’}

keys()

Return keys from dictionary

>>> a_dict.keys()

dict_keys([‘I hate’, ‘You should’])

items()

Returns keys and values from dictionary

>>> a_dict.items()

dict_items([(‘I hate’, ‘you’), (‘You should’, ‘leave)])

hasvalues()

I didn’t find any information about this function, but if its objective is to determine whether a value exists in a dictionary, one usage could be like this:

>>> ‘you’ in a_dict.values()

True

_key()

_key is not defined ; ‘dict’ object has no attribute ‘_key’

‘never’ in a_dict’

False

del a_dict['me']

# there is no ‘me’ in a_dict

KeyError: ‘me’

a_dict.clear()

Clears dictionary items.

Output : {}

Sets:

add()

Throws another item into the set

>>> s = set((1,2,3))

>>> s

{1,2,3}

```
>>> s.add(4)
>>> s
{1,2,3,4}
```

clear()

Clears all items in set.

```
>>>s.clear()
>>> s
set()
```

copy()

Creates copy of set

```
>>> s = set((1,2,3))
>>> s2 = s.copy()
>>> s2
{1, 2, 3}
```

difference()

Returns a set that contains the difference between two sets

```
>>> x = {"apple", "banana", "cherry"}
>>> y = {"google", "microsoft", "apple"}
>>> z = x.difference(y)
>>> z
{‘banana’, ‘cherry’}
```

discard()

Removes the specified item from the set

```
>>> x = {"apple", "banana", "cherry"}
>>> x.discard(‘banana’)
>>> x
{‘apple’, ‘cherry’}
```

intersection()

Returns a set that contains the similarities between two sets

```
>>> x = {‘apple’, ‘banana’, ‘cherry’}
>>> y = {‘google’, ‘microsoft’, ‘apple’}
>>> z = x.intersection(y)
>>> z
{‘apple’}
```

issubset()

Returns True if all items in the set exist in the specified set, else False.

```
>>> x = {‘a’, ‘b’, ‘c’}
>>> y = {‘l’,’m’,’n’,’o’,’b’,’c’,’a’}
>>> z = x.issubset(y)
>>> z
True
```

pop()

Gets an item from a set and deletes it from the list at the same time.

```
>>> x = {‘apple’, ‘banana’, ‘cherry’}
>>> z =  x.pop(‘apple’)
>>> z
‘apple’
>>> x
{‘banana’, ‘cherry’}
```

remove()

Deletes item from set

```
>>> x = {‘apple’, ‘banana’, ‘cherry’}
>>> x.remove(‘apple’)
>>> x
{‘banana’, ‘cherry’}
```

union()

Returns a set that contains all items from the original set + all items from specified set

```
>>> x = {‘apple’, ‘banana’, ‘cherry’}
>>> y = {‘google’, ‘microsoft’, ‘apple’}
>>> z = x.union(y)
>>> z
{‘google’, ‘microsoft’, ‘apple’, ‘cherry’, ‘banana’}
```

update()

Inserts specified items into set

```
>>> A = {'a', 'b'}
>>> B = {1, 2, 3}
>>> A.update(B)
>>> A
{‘a’,1,2,’b’,3}
```

Strings:

capitalize()

Capitalizes the first letter

```
>>> str_ex = 'im so tired'
>>> str_ex.capitalize()
'Im so tired'
```

casefold()

Returns string where all characters are lower case

```
>>> str_ex = 'Im so Tired'
>>> str_ex.casefold()
'im so tired'
```

center()

Center aligns the string using a specified character count

```
>>> center_ex = 'apple'
>>> center_ex.center(10)
'  apple   '
```

count()

Returns number of instances a specified character is found in string

```
>>> center_ex = 'apple'
>>> center_ex.count('p')
2
```

encode()

Returns an encoded version of the given string

```
>>> center_ex.encode() #utf-8
b'apple'
```

find()

Returns the index of first occurrence of the substring, if found

```
>>> center_ex.find('l')
3
```

partition()

Searches for a specified string and splits the string into a tuple containing three elements (before, specified string, after)

```
>>> txt = 'I like to sleep upside down'
>>> x = txt.partition('sleep')
>>> x
('I like to ', 'sleep', ' upside down')
```

replace()

Replaces a specified phrase with another specified phrase

```
>>> txt = 'I like to sleep upside down'
```

>>> x = txt.replace('upside down', 'in the cereal aisle')
>>> x
('I like to sleep in the cereal aisle')

split()
Splits a string into a list of strings by specified separator
>>> string = "one,two,three"
>>> words = string.split(',')
>>> words
['one', 'two', 'three']

title()
Makes the first letter in each word uppercase
>>> ex1 = 'small spec in the universe'
>>> ex1.title()
'Small Spec In The Universe'

zfill()
Adds zeroes in the beginning of the string until it reaches specified length
>>> ex2 = '420'
>>> ex2.zfill(7)
'0000420'

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…

Counter()
Counts the number of characters in a given string
>>> Counter("mississippi")
Counter({'i': 4, 's': 4, 'p': 2, 'm': 1})

from itertools import * (**Bonus**: this one is optional, but recommended)
______Fillin relevant functions yourself…

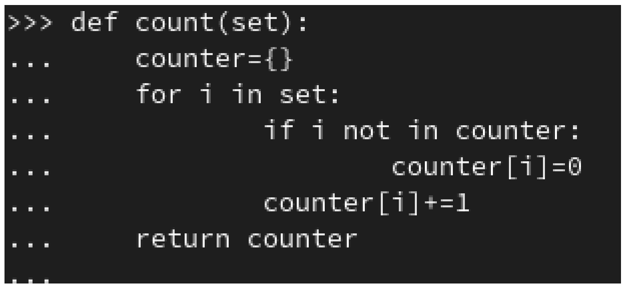
Q2.

```
flower_orders=['W/R/B','W/R/B','W/R/B
```

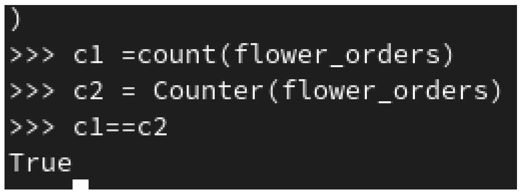
```
','W/R/B','W/R/B','W/R/B','W/R/B','W/
R/B','W/R/B','W/R/B','W/R/B','W/R/B',
'W/R/B','W/R/B','W/R/B','W/R/B','W/R/
B','W/R/B','W/R/B','W/R/B','W/R/B','W
/R/B','W/R/B','W/R/B','W/R/B','W/R/B'
,'W/R/B','W/R/B','W/R/B','W/R/B','W/R
','W/R','W/R','W/R','W/R','W/R','W/R'
,'W/R','W/R','W/R','W/R','W/R','W/R',
'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.

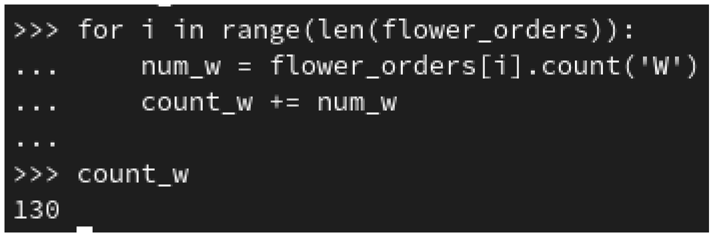


```
>>> def count(set):
...     counter={}
...     for i in set:
...             if i not in counter:
...                     counter[i]=0
...             counter[i]+=1
...     return counter
...
```



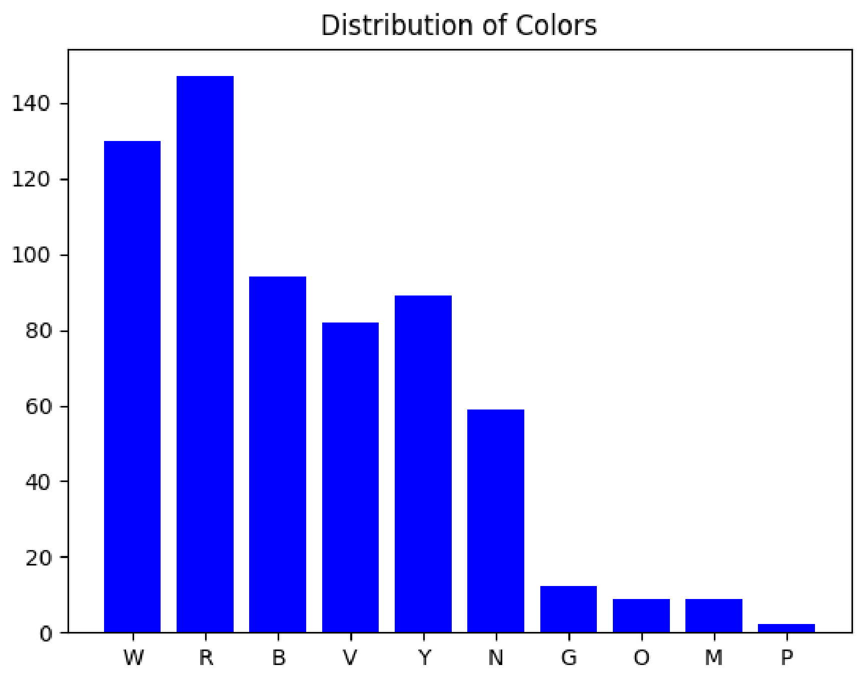
```
)
>>> c1 =count(flower_orders)
>>> c2 = Counter(flower_orders)
>>> c1==c2
True
```

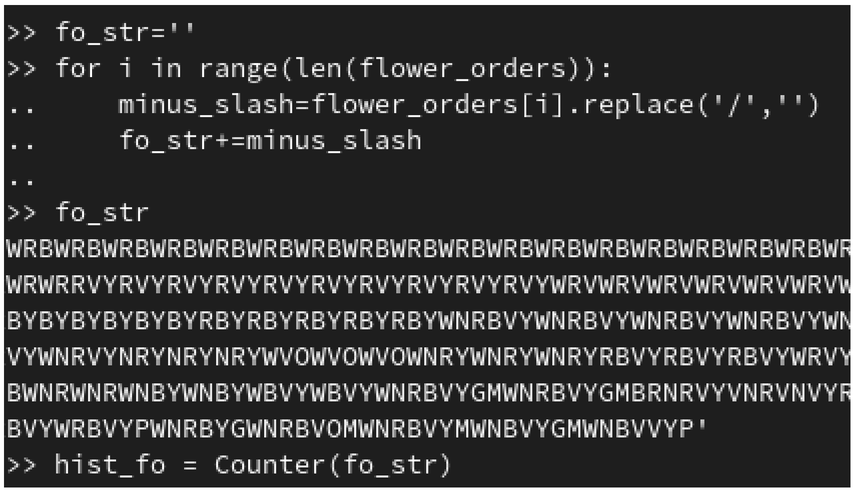
2. Count how many objects have color W in them.



```
>>> for i in range(len(flower_orders)):
...     num_w = flower_orders[i].count('W')
...     count_w += num_w
...
>>> count_w
130
```

3. Make histogram of colors

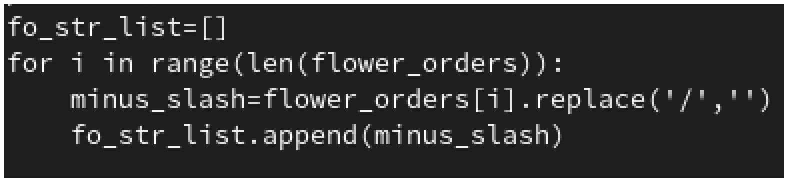




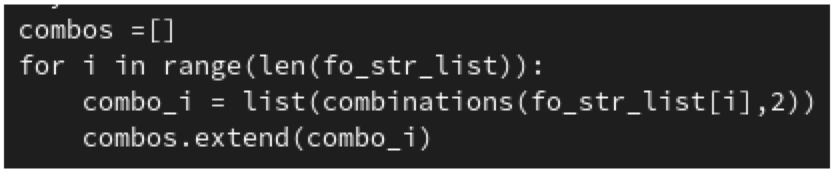
```
>> fo_str=''
>> for i in range(len(flower_orders)):
..     minus_slash=flower_orders[i].replace('/','')
..     fo_str+=minus_slash
..
>> fo_str
WRBWRBWRBWRBWRBWRBWRBWRBWRBWRBWRBWRBWRBWRBWRBWRBWRBWR
WRWRRVYRVYRVYRVYRVYRVYRVYRVYRVYRVYWRVWRVWRVWRVWRVWRVW
BYBYBYBYBYBYRBYRBYRBYRBYRBYWNRBVYWNRBVYWNRBVYWNRBVYWN
VYWNRVYNRYNRYNRYWVOWVOWVOWNRYWNRYWNRYRBVYRBVYRBVYWRVY
BWNRWNRWNBYWNBYWBVYWBVYWNRBVYGMWNRBVYGMBRNRVYVNRVNVYR
BVYWRBVYPWNRBYGWNRBVOMWNRBVYMWNBVYGMWNBVVYP'
>> hist_fo = Counter(fo_str)
```

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

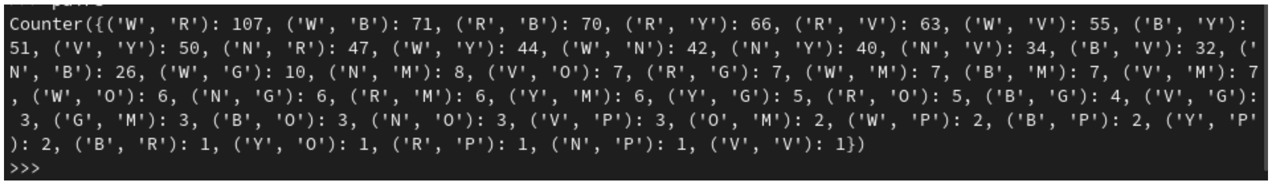
4. Rank the pairs of colors in each order regardless of how many colors are in an order.



```
fo_str_list=[]
for i in range(len(flower_orders)):
    minus_slash=flower_orders[i].replace('/','')
    fo_str_list.append(minus_slash)
```

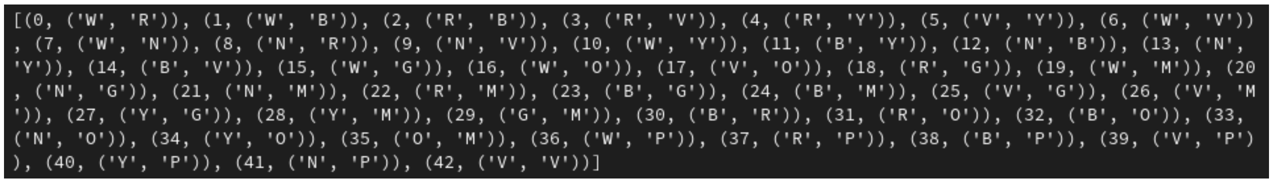


```
combos =[]
for i in range(len(fo_str_list)):
    combo_i = list(combinations(fo_str_list[i],2))
    combos.extend(combo_i)
```



```
Counter({('W', 'R'): 107, ('W', 'B'): 71, ('R', 'B'): 70, ('R', 'Y'): 66, ('R', 'V'): 63, ('W', 'V'): 55, ('B', 'Y'):
51, ('V', 'Y'): 50, ('N', 'R'): 47, ('W', 'Y'): 44, ('W', 'N'): 42, ('N', 'Y'): 40, ('N', 'V'): 34, ('B', 'V'): 32, ('
N', 'B'): 26, ('W', 'G'): 10, ('N', 'M'): 8, ('V', 'O'): 7, ('R', 'G'): 7, ('W', 'M'): 7, ('B', 'M'): 7, ('V', 'M'): 7
, ('W', 'O'): 6, ('N', 'G'): 6, ('R', 'M'): 6, ('Y', 'M'): 6, ('Y', 'G'): 5, ('R', 'O'): 5, ('B', 'G'): 4, ('V', 'G'):
 3, ('G', 'M'): 3, ('B', 'O'): 3, ('N', 'O'): 3, ('V', 'P'): 3, ('O', 'M'): 2, ('W', 'P'): 2, ('B', 'P'): 2, ('Y', 'P'
): 2, ('B', 'R'): 1, ('Y', 'O'): 1, ('R', 'P'): 1, ('N', 'P'): 1, ('V', 'V'): 1})
>>>
```

Ranking:

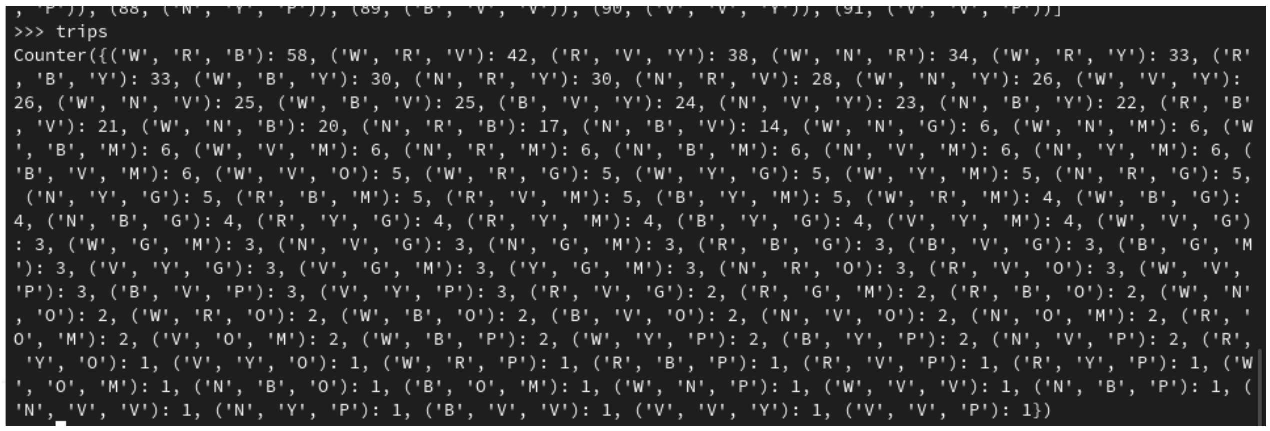


```
[(0, ('W', 'R')), (1, ('W', 'B')), (2, ('R', 'B')), (3, ('R', 'V')), (4, ('R', 'Y')), (5, ('V', 'Y')), (6, ('W', 'V'))
, (7, ('W', 'N')), (8, ('N', 'R')), (9, ('N', 'V')), (10, ('W', 'Y')), (11, ('B', 'Y')), (12, ('N', 'B')), (13, ('N',
'Y')), (14, ('B', 'V')), (15, ('W', 'G')), (16, ('W', 'O')), (17, ('V', 'O')), (18, ('R', 'G')), (19, ('W', 'M')), (20
, ('N', 'G')), (21, ('N', 'M')), (22, ('R', 'M')), (23, ('B', 'G')), (24, ('B', 'M')), (25, ('V', 'G')), (26, ('V', 'M
')), (27, ('Y', 'G')), (28, ('Y', 'M')), (29, ('G', 'M')), (30, ('B', 'R')), (31, ('R', 'O')), (32, ('B', 'O')), (33,
('N', 'O')), (34, ('Y', 'O')), (35, ('O', 'M')), (36, ('W', 'P')), (37, ('R', 'P')), (38, ('B', 'P')), (39, ('V', 'P')
), (40, ('Y', 'P')), (41, ('N', 'P')), (42, ('V', 'V'))]
```

Could have created a for loop to have ranking display vertically, but that would have taken an obnoxious amount of vertical space in this document.

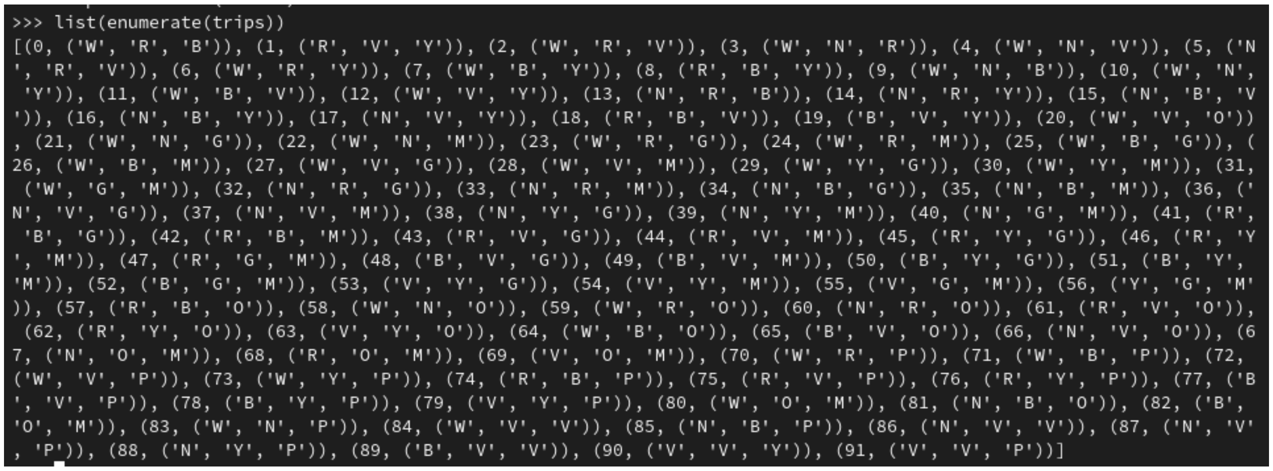
5. Rank the triplets of colors in each order regardless of how many colors are in an order

Counts:



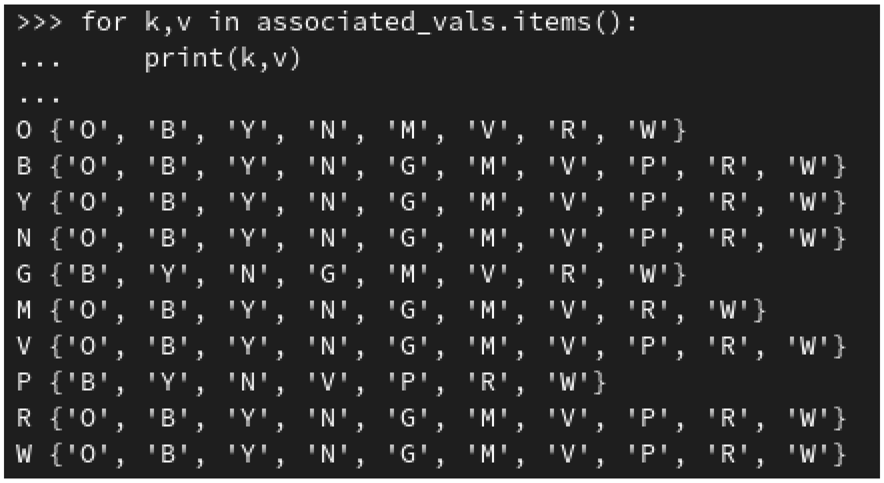
```
, 'P')), (88, ('N', 'Y', 'P')), (89, ('B', 'V', 'V')), (90, ('V', 'V', 'Y')), (91, ('V', 'V', 'P'))]
>>> trips
Counter({('W', 'R', 'B'): 58, ('W', 'R', 'V'): 42, ('R', 'V', 'Y'): 38, ('W', 'N', 'R'): 34, ('W', 'R', 'Y'): 33, ('R'
, 'B', 'Y'): 33, ('W', 'B', 'Y'): 30, ('N', 'R', 'Y'): 30, ('N', 'R', 'V'): 28, ('W', 'N', 'Y'): 26, ('W', 'V', 'Y'):
26, ('W', 'N', 'V'): 25, ('W', 'B', 'V'): 25, ('B', 'V', 'Y'): 24, ('N', 'V', 'Y'): 23, ('N', 'B', 'Y'): 22, ('R', 'B'
, 'V'): 21, ('W', 'N', 'B'): 20, ('N', 'R', 'B'): 17, ('N', 'B', 'V'): 14, ('W', 'N', 'G'): 6, ('W', 'N', 'M'): 6, ('W
', 'B', 'M'): 6, ('W', 'V', 'M'): 6, ('N', 'R', 'M'): 6, ('N', 'B', 'M'): 6, ('N', 'V', 'M'): 6, ('N', 'Y', 'M'): 6, (
'B', 'V', 'M'): 6, ('W', 'V', 'O'): 5, ('W', 'R', 'G'): 5, ('W', 'Y', 'G'): 5, ('W', 'Y', 'M'): 5, ('N', 'R', 'G'): 5,
 ('N', 'Y', 'G'): 5, ('R', 'B', 'M'): 5, ('R', 'V', 'M'): 5, ('B', 'Y', 'M'): 5, ('W', 'R', 'M'): 4, ('W', 'B', 'G'):
4, ('N', 'B', 'G'): 4, ('R', 'Y', 'G'): 4, ('R', 'Y', 'M'): 4, ('B', 'Y', 'G'): 4, ('V', 'Y', 'M'): 4, ('W', 'V', 'G')
: 3, ('W', 'G', 'M'): 3, ('N', 'V', 'G'): 3, ('N', 'G', 'M'): 3, ('R', 'B', 'G'): 3, ('B', 'V', 'G'): 3, ('B', 'G', 'M
'): 3, ('V', 'Y', 'G'): 3, ('V', 'G', 'M'): 3, ('Y', 'G', 'M'): 3, ('N', 'R', 'O'): 3, ('R', 'V', 'O'): 3, ('W', 'V',
'P'): 3, ('B', 'V', 'P'): 3, ('V', 'Y', 'P'): 3, ('R', 'V', 'G'): 2, ('R', 'G', 'M'): 2, ('R', 'B', 'O'): 2, ('W', 'N'
, 'O'): 2, ('W', 'R', 'O'): 2, ('W', 'B', 'O'): 2, ('B', 'V', 'O'): 2, ('N', 'V', 'O'): 2, ('N', 'O', 'M'): 2, ('R', '
O', 'M'): 2, ('V', 'O', 'M'): 2, ('W', 'B', 'P'): 2, ('W', 'Y', 'P'): 2, ('B', 'Y', 'P'): 2, ('N', 'V', 'P'): 2, ('R',
 'Y', 'O'): 1, ('V', 'Y', 'O'): 1, ('W', 'R', 'P'): 1, ('R', 'B', 'P'): 1, ('R', 'V', 'P'): 1, ('R', 'Y', 'P'): 1, ('W
', 'O', 'M'): 1, ('N', 'B', 'O'): 1, ('B', 'O', 'M'): 1, ('W', 'N', 'P'): 1, ('W', 'V', 'V'): 1, ('N', 'B', 'P'): 1, (
'N', 'V', 'V'): 1, ('N', 'Y', 'P'): 1, ('B', 'V', 'V'): 1, ('V', 'V', 'Y'): 1, ('V', 'V', 'P'): 1})
```

Rankings:



```
>>> list(enumerate(trips))
[(0, ('W', 'R', 'B')), (1, ('R', 'V', 'Y')), (2, ('W', 'R', 'V')), (3, ('W', 'N', 'R')), (4, ('W', 'N', 'V')), (5, ('N
', 'R', 'V')), (6, ('W', 'R', 'Y')), (7, ('W', 'B', 'Y')), (8, ('R', 'B', 'Y')), (9, ('W', 'N', 'B')), (10, ('W', 'N',
 'Y')), (11, ('W', 'B', 'V')), (12, ('W', 'V', 'Y')), (13, ('N', 'R', 'B')), (14, ('N', 'R', 'Y')), (15, ('N', 'B', 'V
')), (16, ('N', 'B', 'Y')), (17, ('N', 'V', 'Y')), (18, ('R', 'B', 'V')), (19, ('B', 'V', 'Y')), (20, ('W', 'V', 'O'))
, (21, ('W', 'N', 'G')), (22, ('W', 'N', 'M')), (23, ('W', 'R', 'G')), (24, ('W', 'R', 'M')), (25, ('W', 'B', 'G')), (
26, ('W', 'B', 'M')), (27, ('W', 'V', 'G')), (28, ('W', 'V', 'M')), (29, ('W', 'Y', 'G')), (30, ('W', 'Y', 'M')), (31,
 ('W', 'G', 'M')), (32, ('N', 'R', 'G')), (33, ('N', 'R', 'M')), (34, ('N', 'B', 'G')), (35, ('N', 'B', 'M')), (36, ('
N', 'V', 'G')), (37, ('N', 'V', 'M')), (38, ('N', 'Y', 'G')), (39, ('N', 'Y', 'M')), (40, ('N', 'G', 'M')), (41, ('R',
 'B', 'G')), (42, ('R', 'B', 'M')), (43, ('R', 'V', 'G')), (44, ('R', 'V', 'M')), (45, ('R', 'Y', 'G')), (46, ('R', 'Y
', 'M')), (47, ('R', 'G', 'M')), (48, ('B', 'V', 'G')), (49, ('B', 'V', 'M')), (50, ('B', 'Y', 'G')), (51, ('B', 'Y',
'M')), (52, ('B', 'G', 'M')), (53, ('V', 'Y', 'G')), (54, ('V', 'Y', 'M')), (55, ('V', 'G', 'M')), (56, ('Y', 'G', 'M'
)), (57, ('R', 'B', 'O')), (58, ('W', 'N', 'O')), (59, ('W', 'R', 'O')), (60, ('N', 'R', 'O')), (61, ('R', 'V', 'O')),
 (62, ('R', 'Y', 'O')), (63, ('V', 'Y', 'O')), (64, ('W', 'B', 'O')), (65, ('B', 'V', 'O')), (66, ('N', 'V', 'O')), (6
7, ('N', 'O', 'M')), (68, ('R', 'O', 'M')), (69, ('V', 'O', 'M')), (70, ('W', 'R', 'P')), (71, ('W', 'B', 'P')), (72,
('W', 'V', 'P')), (73, ('W', 'Y', 'P')), (74, ('R', 'B', 'P')), (75, ('R', 'V', 'P')), (76, ('R', 'Y', 'P')), (77, ('B
', 'V', 'P')), (78, ('B', 'Y', 'P')), (79, ('V', 'Y', 'P')), (80, ('W', 'O', 'M')), (81, ('N', 'B', 'O')), (82, ('B',
'O', 'M')), (83, ('W', 'N', 'P')), (84, ('W', 'V', 'V')), (85, ('N', 'B', 'P')), (86, ('N', 'V', 'V')), (87, ('N', 'V'
, 'P')), (88, ('N', 'Y', 'P')), (89, ('B', 'V', 'V')), (90, ('V', 'V', 'Y')), (91, ('V', 'V', 'P'))]
```

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



```
>>> for k,v in associated_vals.items():
...     print(k,v)
...
O {'O', 'B', 'Y', 'N', 'M', 'V', 'R', 'W'}
B {'O', 'B', 'Y', 'N', 'G', 'M', 'V', 'P', 'R', 'W'}
Y {'O', 'B', 'Y', 'N', 'G', 'M', 'V', 'P', 'R', 'W'}
N {'O', 'B', 'Y', 'N', 'G', 'M', 'V', 'P', 'R', 'W'}
G {'B', 'Y', 'N', 'G', 'M', 'V', 'R', 'W'}
M {'O', 'B', 'Y', 'N', 'G', 'M', 'V', 'R', 'W'}
V {'O', 'B', 'Y', 'N', 'G', 'M', 'V', 'P', 'R', 'W'}
P {'B', 'Y', 'N', 'V', 'P', 'R', 'W'}
R {'O', 'B', 'Y', 'N', 'G', 'M', 'V', 'P', 'R', 'W'}
W {'O', 'B', 'Y', 'N', 'G', 'M', 'V', 'P', 'R', 'W'}
```

7. Make a graph showing the probability of having an edge between two colors based on how often they co-occur. (a numpy square matrix)

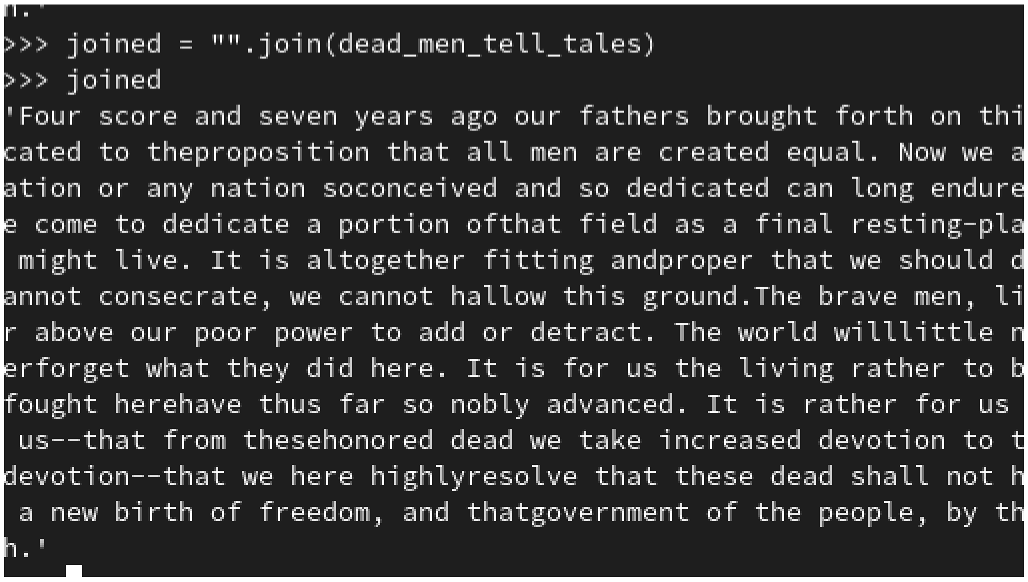
8. Make 10 business questions related to the questions we asked above.
   1. What are the probabilities that customer will order x, given that they ordered y?
   2. What other combinations can be suggested, given an initial preference?
   3. What are the top x combinations ordered?
   4. What do we think contributes to them being the most ordered?
   5. What product(s) are most fitting for promotions?

6. Are there trends to seasonality (Valentines, xmas, etc.)?
7. How is the frequency of types of color combinations affected by the size of the order?
8. Is the distribution of orders affected by type of customer?
9. Can we create clusters of types of customers and develop specific data corresponding to them?
10. What color combination is most profitable?
11. Which is least?

Q3.

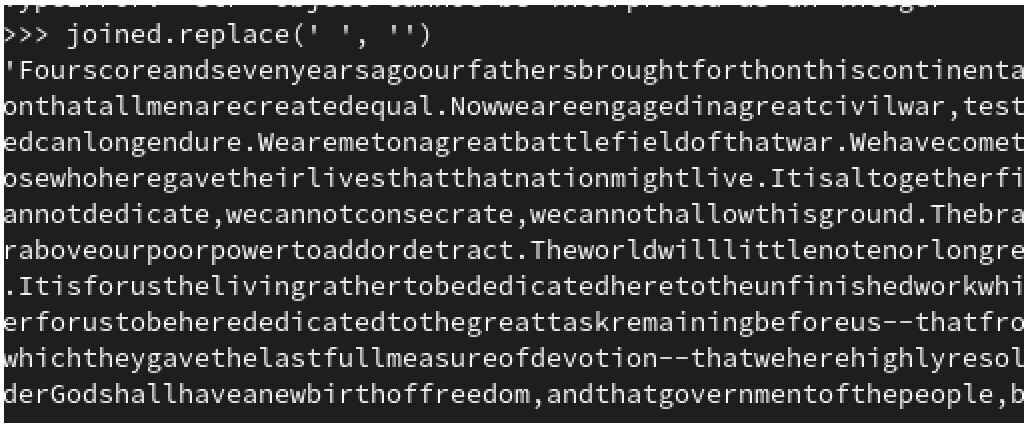
```
dead_men_tell_tales =
['Four score and seven years ago our fathers brought forth on this',
'continent a new nation, conceived in liberty and dedicated to the',
'proposition that all men are created equal. Now we are engaged in',
'a great civil war, testing whether that nation or any nation so',
'conceived 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



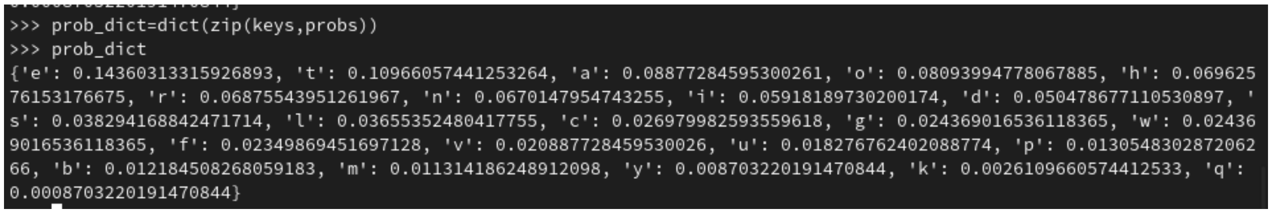
```
n.'
>>> joined = "".join(dead_men_tell_tales)
>>> joined
'Four score and seven years ago our fathers brought forth on thi
cated to theproposition that all men are created equal. Now we a
ation or any nation soconceived and so dedicated can long endure
e come to dedicate a portion ofthat field as a final resting-pla
 might live. It is altogether fitting andproper that we should d
annot consecrate, we cannot hallow this ground.The brave men, li
r above our poor power to add or detract. The world willlittle n
erforget what they did here. It is for us the living rather to b
fought herehave thus far so nobly advanced. It is rather for us 
 us--that from thesehonored dead we take increased devotion to t
devotion--that we here highlyresolve that these dead shall not h
 a new birth of freedom, and thatgovernment of the people, by th
h.'
```

2. Remove spaces



```
>>> joined.replace(' ', '')
'Fourscoreandsevenyearsagoourfathersbroughtforthonthiscontinenta
onthatallmenarecreatedequal.Nowweareengagedinagreatcivilwar,test
edcanlongendure.Wearemetonagreatbattlefieldofthatwar.Wehavecomet
osewhoheregavetheirlivesthatthatnationmightlive.Itisaltogetherfi
annotdedicate,wecannotconsecrate,wecannothallowthisground.Thebra
raboveourpoorpowertoaddordetract.Theworldwilllittlenotenorlongre
.Itisforusthelivingrathertobededicatedheretotheunfinishedworkwhi
erforustobeherededicatedtothegreattaskremainingbeforeus--thatfro
whichtheygavethelastfullmeasureofdevotion--thatweherehighlyresol
derGodshallhaveanewbirthoffreedom,andthatgovernmentofthepeople,b
```

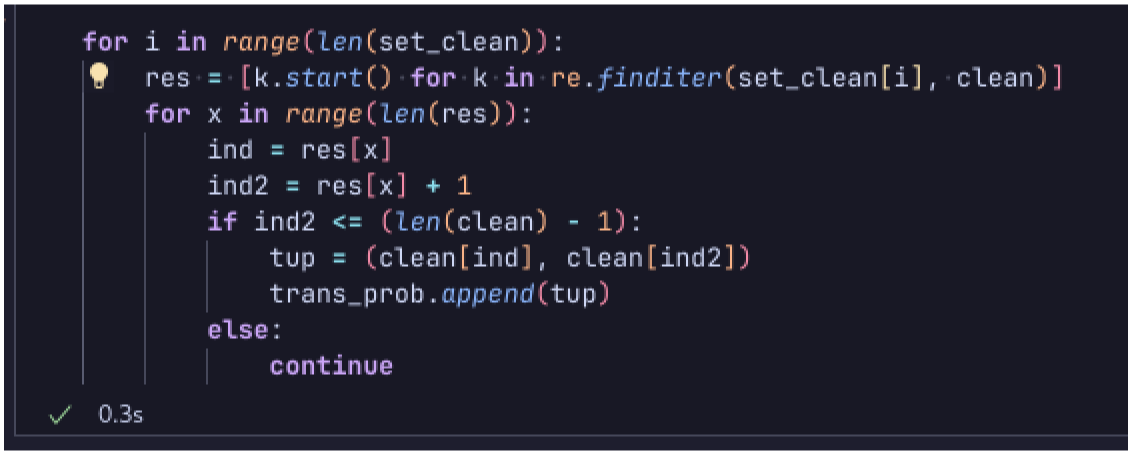
3. Occurrence probabilities for letters



```
>>> prob_dict=dict(zip(keys,probs))
>>> prob_dict
{'e': 0.14360313315926893, 't': 0.10966057441253264, 'a': 0.08877284595300261, 'o': 0.08093994778067885, 'h': 0.069625
76153176675, 'r': 0.06875543951261967, 'n': 0.0670147954743255, 'i': 0.05918189730200174, 'd': 0.050478677110530897, '
s': 0.038294168842471714, 'l': 0.03655352480417755, 'c': 0.026979982593559618, 'g': 0.024369016536118365, 'w': 0.02436
9016536118365, 'f': 0.02349869451697128, 'v': 0.020887728459530026, 'u': 0.018276762402088774, 'p': 0.0130548302872062
66, 'b': 0.012184508268059183, 'm': 0.011314186248912098, 'y': 0.008703220191470844, 'k': 0.0026109660574412533, 'q':
0.0008703220191470844}
```

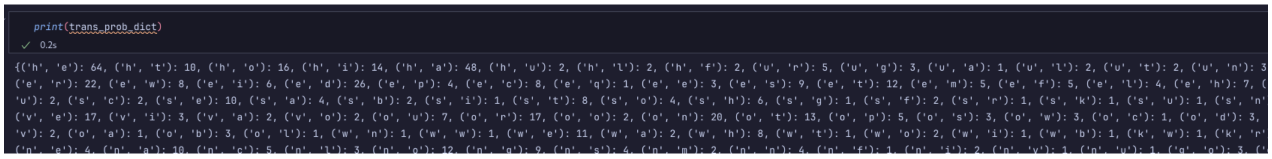
4. Tell me transition probabilities for every pair of letter

*Probabilities were calculated based on the total for each unique letter, rather than total of all letters



```
for i in range(len(set_clean)):
    res = [k.start() for k in re.finditer(set_clean[i], clean)]
    for x in range(len(res)):
        ind = res[x]
        ind2 = res[x] + 1
        if ind2 <= (len(clean) - 1):
            tup = (clean[ind], clean[ind2])
            trans_prob.append(tup)
        else:
            continue
```

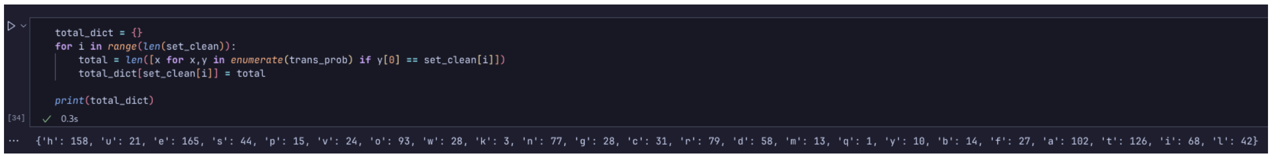
```
✓ 0.3s
```



```
print(trans_prob_dict)
```

```
✓ 0.2s
```

```
{('h', 'e'): 64, ('h', 't'): 10, ('h', 'o'): 16, ('h', 'i'): 14, ('h', 'a'): 48, ('h', 'u'): 2, ('h', 'l'): 2, ('h', 'f'): 2, ('u', 'r'): 5, ('u', 'g'): 3, ('u', 'a'): 1, ('u', 'l'): 2, ('u', 't'): 2, ('u', 'n'): 3
('e', 'r'): 22, ('e', 'w'): 8, ('e', 'i'): 6, ('e', 'd'): 26, ('e', 'p'): 4, ('e', 'c'): 8, ('e', 'q'): 1, ('e', 'e'): 3, ('e', 's'): 9, ('e', 't'): 12, ('e', 'm'): 5, ('e', 'f'): 5, ('e', 'l'): 4, ('e', 'h'): 7, (
'u'): 2, ('s', 'c'): 2, ('s', 'e'): 10, ('s', 'a'): 4, ('s', 'b'): 2, ('s', 'i'): 1, ('s', 't'): 8, ('s', 'o'): 4, ('s', 'h'): 6, ('s', 'g'): 1, ('s', 'f'): 2, ('s', 'r'): 1, ('s', 'k'): 1, ('s', 'u'): 1, ('s', 'n'
('v', 'e'): 17, ('v', 'i'): 3, ('v', 'a'): 2, ('v', 'o'): 2, ('o', 'u'): 7, ('o', 'r'): 17, ('o', 'o'): 2, ('o', 'n'): 20, ('o', 't'): 13, ('o', 'p'): 5, ('o', 's'): 3, ('o', 'w'): 3, ('o', 'c'): 1, ('o', 'd'): 3,
'v'): 2, ('o', 'a'): 1, ('o', 'b'): 3, ('o', 'l'): 1, ('w', 'n'): 1, ('w', 'w'): 1, ('w', 'e'): 11, ('w', 'a'): 2, ('w', 'h'): 8, ('w', 't'): 1, ('w', 'o'): 2, ('w', 'i'): 1, ('w', 'b'): 1, ('k', 'w'): 1, ('k', 'r'
('n', 'e'): 4, ('n', 'a'): 10, ('n', 'c'): 5, ('n', 'l'): 3, ('n', 'o'): 12, ('n', 'g'): 9, ('n', 's'): 4, ('n', 'm'): 2, ('n', 'n'): 4, ('n', 'f'): 1, ('n', 'i'): 2, ('n', 'y'): 1, ('n', 'u'): 1, ('g', 'o'): 3, ('
```

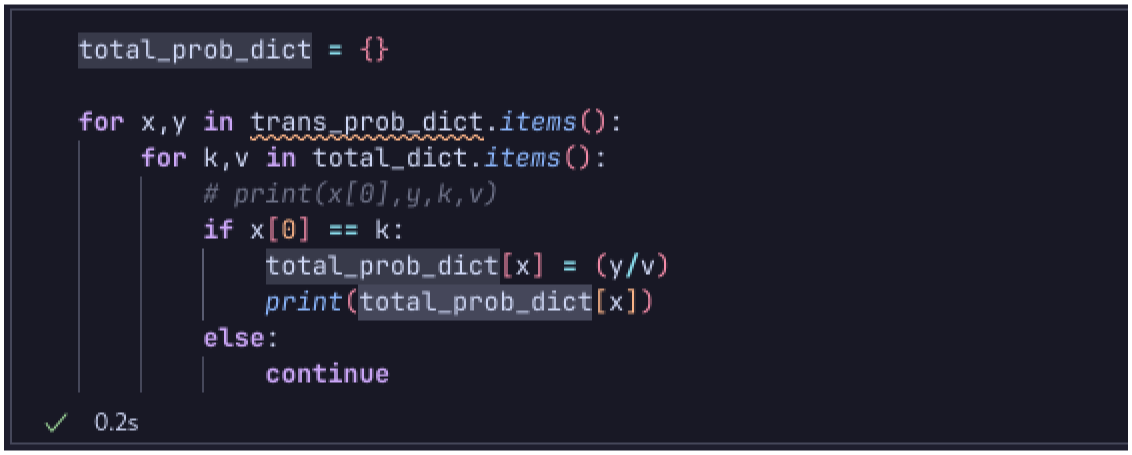


```
total_dict = {}
for i in range(len(set_clean)):
    total = len([x for x,y in enumerate(trans_prob) if y[0] == set_clean[i]])
    total_dict[set_clean[i]] = total

print(total_dict)
```

```
[34]  ✓ 0.3s
```

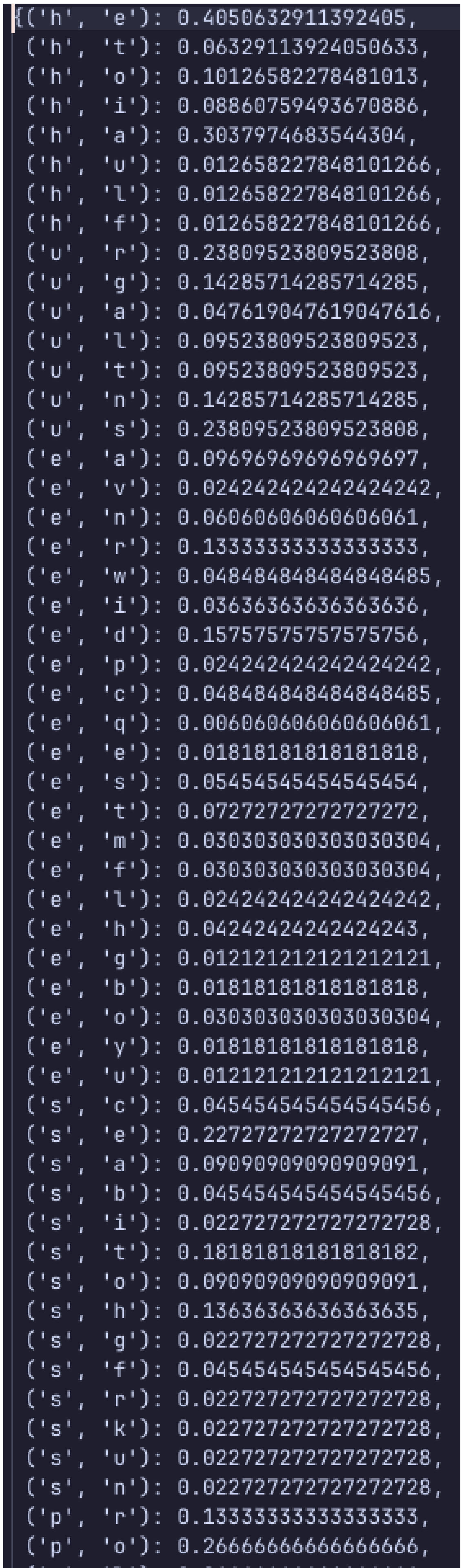
```
... {'h': 158, 'u': 21, 'e': 165, 's': 44, 'p': 15, 'v': 24, 'o': 93, 'w': 28, 'k': 3, 'n': 77, 'g': 28, 'c': 31, 'r': 79, 'd': 58, 'm': 13, 'q': 1, 'y': 10, 'b': 14, 'f': 27, 'a': 102, 't': 126, 'i': 68, 'l': 42}
```



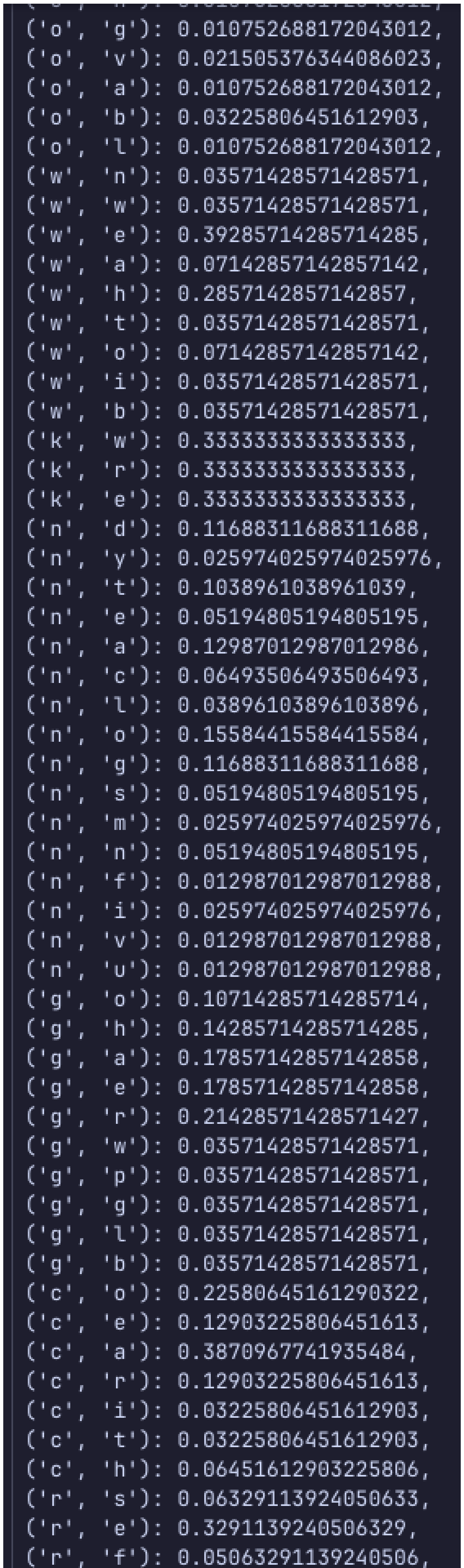
```
total_prob_dict = {}

for x,y in trans_prob_dict.items():
    for k,v in total_dict.items():
        # print(x[0],y,k,v)
        if x[0] == k:
            total_prob_dict[x] = (y/v)
            print(total_prob_dict[x])
        else:
            continue
```

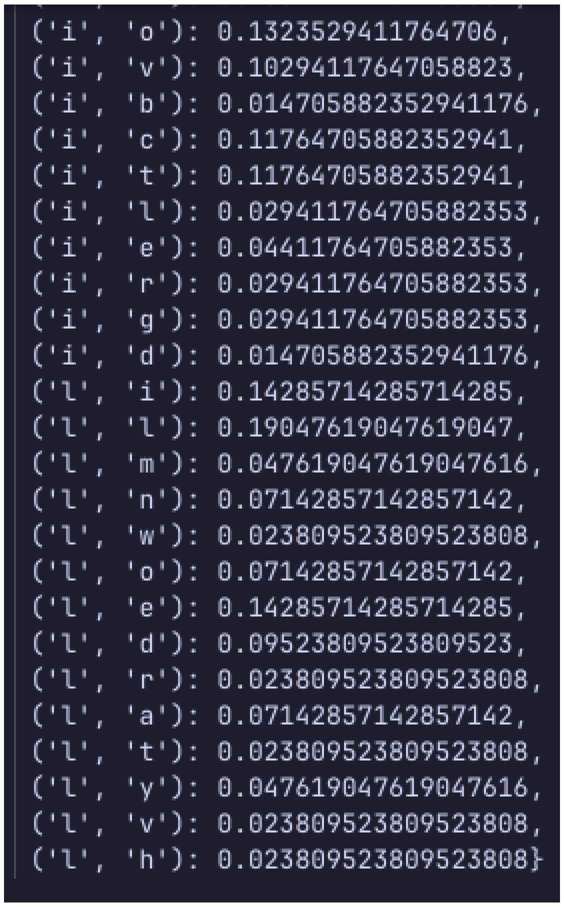
```
✓ 0.2s
```



```
{('h', 'e'): 0.4050632911392405,
 ('h', 't'): 0.06329113924050633,
 ('h', 'o'): 0.10126582278481013,
 ('h', 'i'): 0.08860759493670886,
 ('h', 'a'): 0.3037974683544304,
 ('h', 'u'): 0.012658227848101266,
 ('h', 'l'): 0.012658227848101266,
 ('h', 'f'): 0.012658227848101266,
 ('u', 'r'): 0.23809523809523808,
 ('u', 'g'): 0.14285714285714285,
 ('u', 'a'): 0.047619047619047616,
 ('u', 'l'): 0.09523809523809523,
 ('u', 't'): 0.09523809523809523,
 ('u', 'n'): 0.14285714285714285,
 ('u', 's'): 0.23809523809523808,
 ('e', 'a'): 0.09696969696969697,
 ('e', 'v'): 0.024242424242424242,
 ('e', 'n'): 0.06060606060606061,
 ('e', 'r'): 0.13333333333333333,
 ('e', 'w'): 0.048484848484848485,
 ('e', 'i'): 0.03636363636363636,
 ('e', 'd'): 0.15757575757575756,
 ('e', 'p'): 0.024242424242424242,
 ('e', 'c'): 0.048484848484848485,
 ('e', 'q'): 0.006060606060606061,
 ('e', 'e'): 0.01818181818181818,
 ('e', 's'): 0.05454545454545454,
 ('e', 't'): 0.07272727272727272,
 ('e', 'm'): 0.030303030303030304,
 ('e', 'f'): 0.030303030303030304,
 ('e', 'l'): 0.024242424242424242,
 ('e', 'h'): 0.04242424242424243,
 ('e', 'g'): 0.012121212121212121,
 ('e', 'b'): 0.01818181818181818,
 ('e', 'o'): 0.030303030303030304,
 ('e', 'y'): 0.01818181818181818,
 ('e', 'u'): 0.012121212121212121,
 ('s', 'c'): 0.045454545454545456,
 ('s', 'e'): 0.22727272727272727,
 ('s', 'a'): 0.09090909090909091,
 ('s', 'b'): 0.045454545454545456,
 ('s', 'i'): 0.022727272727272728,
 ('s', 't'): 0.18181818181818182,
 ('s', 'o'): 0.09090909090909091,
 ('s', 'h'): 0.13636363636363635,
 ('s', 'g'): 0.022727272727272728,
 ('s', 'f'): 0.045454545454545456,
 ('s', 'r'): 0.022727272727272728,
 ('s', 'k'): 0.022727272727272728,
 ('s', 'u'): 0.022727272727272728,
 ('s', 'n'): 0.022727272727272728,
 ('p', 'r'): 0.13333333333333333,
 ('p', 'o'): 0.26666666666666666,
```

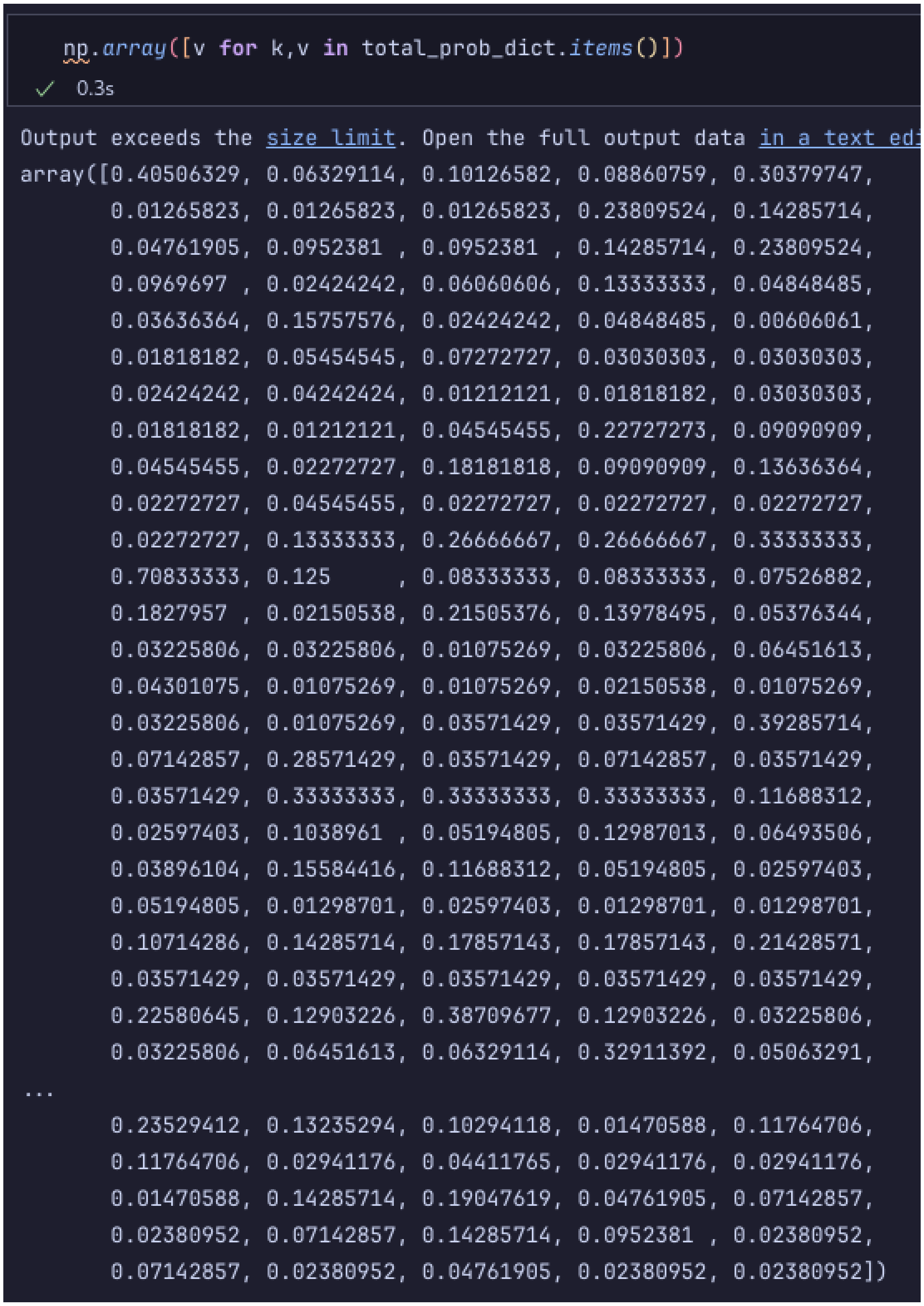


```
('o', 'g'): 0.010752688172043012,
('o', 'v'): 0.021505376344086023,
('o', 'a'): 0.010752688172043012,
('o', 'b'): 0.03225806451612903,
('o', 'l'): 0.010752688172043012,
('w', 'n'): 0.03571428571428571,
('w', 'w'): 0.03571428571428571,
('w', 'e'): 0.39285714285714285,
('w', 'a'): 0.07142857142857142,
('w', 'h'): 0.2857142857142857,
('w', 't'): 0.03571428571428571,
('w', 'o'): 0.07142857142857142,
('w', 'i'): 0.03571428571428571,
('w', 'b'): 0.03571428571428571,
('k', 'w'): 0.3333333333333333,
('k', 'r'): 0.3333333333333333,
('k', 'e'): 0.3333333333333333,
('n', 'd'): 0.11688311688311688,
('n', 'y'): 0.025974025974025976,
('n', 't'): 0.1038961038961039,
('n', 'e'): 0.05194805194805195,
('n', 'a'): 0.12987012987012986,
('n', 'c'): 0.06493506493506493,
('n', 'l'): 0.03896103896103896,
('n', 'o'): 0.15584415584415584,
('n', 'g'): 0.11688311688311688,
('n', 's'): 0.05194805194805195,
('n', 'm'): 0.025974025974025976,
('n', 'n'): 0.05194805194805195,
('n', 'f'): 0.012987012987012988,
('n', 'i'): 0.025974025974025976,
('n', 'v'): 0.012987012987012988,
('n', 'u'): 0.012987012987012988,
('g', 'o'): 0.10714285714285714,
('g', 'h'): 0.14285714285714285,
('g', 'a'): 0.17857142857142858,
('g', 'e'): 0.17857142857142858,
('g', 'r'): 0.21428571428571427,
('g', 'w'): 0.03571428571428571,
('g', 'p'): 0.03571428571428571,
('g', 'g'): 0.03571428571428571,
('g', 'l'): 0.03571428571428571,
('g', 'b'): 0.03571428571428571,
('c', 'o'): 0.22580645161290322,
('c', 'e'): 0.12903225806451613,
('c', 'a'): 0.3870967741935484,
('c', 'r'): 0.12903225806451613,
('c', 'i'): 0.03225806451612903,
('c', 't'): 0.03225806451612903,
('c', 'h'): 0.06451612903225806,
('r', 's'): 0.06329113924050633,
('r', 'e'): 0.3291139240506329,
('r', 'f'): 0.05063291139240506,
 ('d', 'u'): 0.017241379310344827,
 ('d', 'o'): 0.06896551724137931,
 ('d', 'p'): 0.017241379310344827,
 ('d', 'w'): 0.06896551724137931,
 ('d', 'h'): 0.05172413793103448,
 ('d', 'v'): 0.017241379310344827,
 ('m', 'e'): 0.5384615384615384,
 ('m', 'i'): 0.07692307692307693,
 ('m', 'b'): 0.07692307692307693,
 ('m', 'a'): 0.15384615384615385,
 ('m', 't'): 0.15384615384615385,
 ('q', 'u'): 1.0,
 ('y', 'e'): 0.1,
 ('y', 'a'): 0.2,
 ('y', 'n'): 0.1,
 ('y', 'h'): 0.1,
 ('y', 'd'): 0.1,
 ('y', 'w'): 0.1,
 ('y', 'g'): 0.1,
 ('y', 'r'): 0.1,
 ('y', 't'): 0.1,
 ('b', 'r'): 0.14285714285714285,
 ('b', 'e'): 0.35714285714285715,
 ('b', 'a'): 0.07142857142857142,
 ('b', 'u'): 0.14285714285714285,
 ('b', 'o'): 0.07142857142857142,
 ('b', 'l'): 0.07142857142857142,
 ('b', 'i'): 0.07142857142857142,
 ('b', 'y'): 0.07142857142857142,
 ('f', 'o'): 0.37037037037037035,
 ('f', 'a'): 0.1111111111111111,
 ('f', 'i'): 0.18518518518518517,
 ('f', 't'): 0.1111111111111111,
 ('f', 'r'): 0.1111111111111111,
 ('f', 'u'): 0.037037037037037035,
 ('f', 'd'): 0.037037037037037035,
 ('f', 'f'): 0.037037037037037035,
 ('a', 'n'): 0.14705882352941177,
 ('a', 'r'): 0.09803921568627451,
 ('a', 'g'): 0.0392156862745098,
 ('a', 't'): 0.35294117647058826,
 ('a', 'l'): 0.08823529411764706,
 ('a', 'v'): 0.0784313725490196,
 ('a', 'p'): 0.00980392156862745,
 ('a', 's'): 0.049019607843137254,
 ('a', 'f'): 0.00980392156862745,
 ('a', 'c'): 0.0196078431372549,
 ('a', 'd'): 0.049019607843137254,
 ('a', 'b'): 0.00980392156862745,
 ('a', 'y'): 0.00980392156862745,
 ('a', 'i'): 0.0196078431372549,
 ('a', 'k'): 0.00980392156862745,
 ('a', 'u'): 0.00980392156862745,
```



```
('i', 'o'): 0.1323529411764706,
('i', 'v'): 0.10294117647058823,
('i', 'b'): 0.014705882352941176,
('i', 'c'): 0.11764705882352941,
('i', 't'): 0.11764705882352941,
('i', 'l'): 0.029411764705882353,
('i', 'e'): 0.04411764705882353,
('i', 'r'): 0.029411764705882353,
('i', 'g'): 0.029411764705882353,
('i', 'd'): 0.014705882352941176,
('l', 'i'): 0.14285714285714285,
('l', 'l'): 0.19047619047619047,
('l', 'm'): 0.047619047619047616,
('l', 'n'): 0.07142857142857142,
('l', 'w'): 0.023809523809523808,
('l', 'o'): 0.07142857142857142,
('l', 'e'): 0.14285714285714285,
('l', 'd'): 0.09523809523809523,
('l', 'r'): 0.023809523809523808,
('l', 'a'): 0.07142857142857142,
('l', 't'): 0.023809523809523808,
('l', 'y'): 0.047619047619047616,
('l', 'v'): 0.023809523809523808,
('l', 'h'): 0.023809523809523808}
```

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



```
np.array([v for k,v in total_prob_dict.items()])
✓ 0.3s
```

```
Output exceeds the size limit. Open the full output data in a text edi
array([0.40506329, 0.06329114, 0.10126582, 0.08860759, 0.30379747,
       0.01265823, 0.01265823, 0.01265823, 0.23809524, 0.14285714,
       0.04761905, 0.0952381 , 0.0952381 , 0.14285714, 0.23809524,
       0.0969697 , 0.02424242, 0.06060606, 0.13333333, 0.04848485,
       0.03636364, 0.15757576, 0.02424242, 0.04848485, 0.00606061,
       0.01818182, 0.05454545, 0.07272727, 0.03030303, 0.03030303,
       0.02424242, 0.04242424, 0.01212121, 0.01818182, 0.03030303,
       0.01818182, 0.01212121, 0.04545455, 0.22727273, 0.09090909,
       0.04545455, 0.02272727, 0.18181818, 0.09090909, 0.13636364,
       0.02272727, 0.04545455, 0.02272727, 0.02272727, 0.02272727,
       0.02272727, 0.13333333, 0.26666667, 0.26666667, 0.33333333,
       0.70833333, 0.125     , 0.08333333, 0.08333333, 0.07526882,
       0.1827957 , 0.02150538, 0.21505376, 0.13978495, 0.05376344,
       0.03225806, 0.03225806, 0.01075269, 0.03225806, 0.06451613,
       0.04301075, 0.01075269, 0.01075269, 0.02150538, 0.01075269,
       0.03225806, 0.01075269, 0.03571429, 0.03571429, 0.39285714,
       0.07142857, 0.28571429, 0.03571429, 0.07142857, 0.03571429,
       0.03571429, 0.33333333, 0.33333333, 0.33333333, 0.11688312,
       0.02597403, 0.1038961 , 0.05194805, 0.12987013, 0.06493506,
       0.03896104, 0.15584416, 0.11688312, 0.05194805, 0.02597403,
       0.05194805, 0.01298701, 0.02597403, 0.01298701, 0.01298701,
       0.10714286, 0.14285714, 0.17857143, 0.17857143, 0.21428571,
       0.03571429, 0.03571429, 0.03571429, 0.03571429, 0.03571429,
       0.22580645, 0.12903226, 0.38709677, 0.12903226, 0.03225806,
       0.03225806, 0.06451613, 0.06329114, 0.32911392, 0.05063291,
...
       0.23529412, 0.13235294, 0.10294118, 0.01470588, 0.11764706,
       0.11764706, 0.02941176, 0.04411765, 0.02941176, 0.02941176,
       0.01470588, 0.14285714, 0.19047619, 0.04761905, 0.07142857,
       0.02380952, 0.07142857, 0.14285714, 0.0952381 , 0.02380952,
       0.07142857, 0.02380952, 0.04761905, 0.02380952, 0.02380952])
```

I’ve now realized the way I set up my dictionary system does not lend itself to this kind of array.

6. plot graph of transition probabilities from letter to letter

Unrelated:
7. Flatten a nested list
Cool intro python resources:
    https://thomas-cokelaer.info/tutorials/python/index.html