## 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():**adds an element to end of list**

list = [1,2,3]

list.append(4)

returns -> [1,2,3,4]

extend(): **similar to append but for multiple elements**

list = [1,2,3]

list.extend([4,5,6])

returns -> [1,2,3,4,5,6]

index(): **used to find index of first occurrence of given element**

list = [1,2,3,4,5,5,6]

list.index(5)

returns -> 4

index(value, integer): **version of index() that allows a specific start of search**

list = [1,2,3,4,5,5,6]

list.index(5,5)

returns -> 5

insert(position): **inserts element in specific position of list**

list = [1,2,3]

list.insert(‘test’,2)

returns -> [1,2,’test’,3]

remove(): **used to remove first occurrence of specific element from list**

list = [1,2,2,3,4]

list.remove(2)

returns -> [1,2,3,4]

pop(): **removes element at specific position of list**

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

list.pop(2)

returns -> [1,2,4,5]

count(): **used to count number of occurrences of specific element in list**

list = [1,2,2,3,3,3,4]

list.count(2)

returns -> 2

reverse(): **used to reverse the elements in the list**

list = [1,2,3,4]

list.reverse()

returns -> [4,3,2,1]

sort(): **used to sort elements in ascending order, can be used with sort(reverse=True) for descending**

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

list.sort()

returns -> [1,2,3,4,5]

[1]+[1]: **concatenates both lists and returns a new list**

Returns -> [1,1]

[2]\*2: **creates new list that is repetition of original list**

Returns - > [2,2]

[1,2][1:]: **returns a new list from a slice of original**

Returns -> [2]

[x for x in [2,3]]: **list comprehension that creates new list by performing operation on each element**

Returns -> [2,3]

[x for x in [1,2] if x ==1**]: list comprehension creates new list and only adding elements equal to 1**

Returns -> [1]

[y\*2 for x in [[1,2],[3,4]] for y in x]: **nested list comprehension. Iterates over each element in original list and performs operation on each element**

Returns -> [2,4,6,8]

A = [1]: **sets variable ‘A’ to a list equal to [1]**

**Tuple:**

count(): similar to count() for lists. Returns number of occurrences of specific element

tuple = (1,2,2,2,3,4)

tuple.count(2)

returns -> 3

index(): similar to index() in lists, it returns the index of a specific element in tuple

tuple = (1,2,3,4,5)

tuple.index(4)

returns -> 3

build a dictionary from tuples

tuple\_list = [(‘a’,2),(‘b’,4),(‘c’,6)]

dictionary = {key: value for key, value in tuple\_list}

OR

dict(tuple\_list)

OR

dictionary = {}

dictionary.update(tuple\_list

returns -> {‘a’ : 2, ‘b’ : 4, ‘c’ : 6}

unpack tuples

tuple = (1,2,3,4)

q,w,e,r = tuple

returns -> q = 1 , w = 2, e = 3, r = 4

**Dicts:**

a\_dict = {'I hate':'you', 'You should':’leave’}: creates dictionary with two key value pairs

keys(): used to return keys from dictionary

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

dict.keys()

returns -> dict\_keys([‘I hate’,’You should’])

items(): used to view key value tuple pairs of a dictionary

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

dict.items()

returns -> dict\_items([(‘I hate’,’you’),(‘You should’,’leave’)])

hasvalues(): not a python function

key(): used to return key for specific value in dictionary

key(dict, ‘you’)

returns -> ‘I hate’

‘never’ in a\_dict: **checks if the word ‘never’ appears in the dictionary, a\_dict. Returns boolean**

Returns -> False

del a\_dict['me']: deletes key value pair for key == ‘me’ if key exists

Returns -> KeyError as ‘me’ does not exist as key in a\_dict

a\_dict.clear(): deletes all key value pairs from dictionary

Returns -> {}

**Sets:**

add(): adds a new element to the set, will not add if element is already present

set = {1,2,3}

set.add(4)

returns -> {1,2,3,4)

clear(): removes all elements from set

set = {1,2,3}

set.clear()

Returns -> {}

copy(): returns shallow copy of set

set = {1,2,3}

new\_set = set.copy()

Returns -> new\_set = {1,2,3}

difference(): returns new set that contains elements present in first set but not second

set\_1 = {1,2,3}

set\_2 = {2,3,4}

set\_1.difference(set\_2)

Returns -> {1}

discard(): removes specific element of set

set = {1,2,3}

set.discard(2)

Returns -> {1,3}

intersection(): returns new set that contains elements that are present in both sets

set\_1 = {1,2,3}

set\_2 = {2,3,4}

set\_1.intersection(set\_2)

Returns -> {2,3}

issubset(): returns True if set is a subset of the other set

set\_1 = {1,2,3}

set\_2 = {1,2,3,4,5}

set\_1.issubset(set\_2)

Returns -> True

pop(): removes an arbitrary element from the set

set = {1,2,3,4}

set.pop()

Returns -> 2

remove(): removes a specific element from the set

set = {1,2,3,4}

set.remove(3)

Returns -> {1,2,4}

union(): returns new set that contains elements from both sets

set\_1 = {1,2,3}

set\_2 = {1,2,3,4,5}

set\_1.union(set\_2)

Returns -> {1,2,3,4,5}

update(): adds elements from one set to the original set

set\_1 = {1,2,3}

set\_2 = {7}

set\_1.update(set\_2)

Returns -> {1,2,3,7}

**Strings:**

capitalize(): method returns copy of string with first character capitalzed

string = ‘hello world’

string.capitalize()

returns -> ‘Hello world’

casefold(): returns copy of string that is suitable for case-insensitive comparisons

string = ‘hELLO wOrLD’

string.casefold()

Returns -> ‘hello world’

center(): returns copy of string padded with fill character with centered string

string = ‘hello’

string.center(10,’-‘)

Returns -> ‘--hello---’

count(): returns number of occurrences of specific substring in string

string = ‘hello world. hello world’

string.count(‘world’)

returns -> 2

encode(): returns encoded version of string as bytes object

string = ‘hello world’

string.encode(‘utf-32’)

returns - > b'\xff\xfeh\x00e\x00l\x00l\x00o\x00 \x00w\x00o\x00r\x00l\x00d\x00'

find(): searches for substring within string and returns index of substring

string = ‘hello world’

string.find(‘world’)

Returns -> 6

partition(): searches string for specified separater in string and returns tuple of part before, separator and part after

string = ‘hello, world’

string.partition(‘,’)

Returns -> (“hello”,”,”,”world”)

replace():returns copy of string with specified substrings replaced with specified substrings

string = ‘hello world’

string.replace(‘hello’,’bye’)

returns -> ‘bye world’

split(): splits string into list of substrings based on specified separator

string = ‘hello,world’

string.split(‘,’)

returns - > [‘hello’,’world]

title(): returns copy of string with each word’s first letter capitalized

string = ‘hello world’

string.title()

returns -> ‘Hello World’

zfill() : returns a copy of string padded with zeros on left to specified length

string = ‘test’

string.zfill(8)

returns -> ‘0000test’

*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…

**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.

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

3. Make histogram of colors

# 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.

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

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

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.

**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

2. Remove spaces

3. Occurrence probabilities for letters

4. Tell me transition probabilities for every pair of letters

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

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>