Stats 21 - HW 3

The questions have been entered into this document. You will modify the document by entering your code.

Make sure you run the cell so the requested output is visible. Download the finished document as a PDF file. If you are unable to convert it to a PDF, you can download it as an HTML file and then print to PDF.

Homework is an opportunity to practice coding and to practice problem solving. Doing exercises is where you will do most of your learning.

Copying someone else's solutions takes away your learning opportunities. It is also academic dishonesty.

Reading

- Chapters 11, 12, and 14

Please keep up with the reading. The chapters are short.

Exercise 11.1

Write a function that reads the words in `words.txt` and stores them as keys in a dictionary. It doesn't matter what the values are. Then you can use the in operator as a fast way to check whether a string is in the dictionary.

Use the same `words.txt` file from HW2.

Do Exercise 10.10 but this time searching the dictionary using the `in` operator. You can see how much faster it is to search a dictionary.

In [1]:
```
def make_word_dict():
    words = open("words.txt")
    word_dict = {}
    for line in words:
        word_dict[line[:-1]] = 0
    return word_dict
```

In [2]:
```
word_dict = make_word_dict()
```

In [3]:
```
"hello" in word_dict
```

Out[3]:
```
True
```

Exercise 11.4

In Exercise 10.7, you created a function called `has_duplicates()`. It takes a list as a parameter and returns `True` if there is any object that appears more than once in the list.

Use a dictionary to write a faster, simpler version of `has_duplicates()`.

In [4]:
```
def has_duplicates(t):
    words_dict = dict(zip(t, t))
    return len(words_dict) == len(t)
```

In [5]:
```
has_duplicates(['a','b','b','c'])
```

Out[5]:
```
False
```

In [6]:
```
has_duplicates(['a','b','c','a'])
```

Out[6]:
```
False
```

Exercise 11.5

A Caesar cipher is a weak form of encryption that involves 'rotating' each letter by a fixed number of places. To rotate a letter means to shift it through the alphabet, wrapping around to the beginning if necessary. "A" rotated by 3 is "D". "Z" rotated by 1 is "A".

Two words are "rotate pairs" if you can rotate one of them and get the other. For example, "cheer" rotated by 7 is "jolly".

Write a script that reads in the wordlist `words.txt` and finds all the rotate pairs of words that are 5 letters or longer.

One function that could be helpful is the function `ord()` which converts a character to a numeric code. Keep in mind that numeric codes for uppercase and lowercase letters are different.

Some hints:

- you can write helper functions, such as a function that will rotate a letter by a certain number and/or another function that will rotate a word by a number of letters
- to keep your script running quickly, you should use the wordlist dictionary from exercise 11.1

In [7]:
```
# write your code here.
def rotate_word(word, shift):
    lword = word.lower()
    new_word = ""
    for letter in lword:
        num = ord(letter) + shift
        if num > 122:
            num -= 26
        new_word += chr(num)
    return new_word
```

In [8]:
```
# test case
rotate_word("abcdefghijklmnopqrstuvwxyz", 6)
```

Out[8]:
```
'ghijklmnopqrstuvwxyzabcdef'
```

In [9]:
```
new_dict = {}
new_list = []
for word in word_dict:
    if len(word) >= 5:
        for i in range(1, 25):
            find_word = rotate_word(word, i)
            if find_word in word_dict:
                new_dict[word] = find_word
                if find_word not in new_dict:
                    new_list = new_list + [(word, find_word)]
print(new_list)
```

```
[('abjurer', 'nowhere'), ('adder', 'beefs'), ('ahull', 'gnarr'), ('alkyd', 'epoch'),
('alula', 'tenet'), ('ambit', 'setal'), ('anteed', 'bouffe'), ('aptly', 'timer'), ('a
rena', 'river'), ('baiza', 'tsars'), ('banjo', 'ferns'), ('benni', 'ruddy'), ('biff
s', 'holly'), ('bolls', 'hurry'), ('bombyx', 'hushed'), ('bourg', 'viola'), ('buffi',
'hallo'), ('bulls', 'harry'), ('bunny', 'sleep'), ('butyl', 'hazer'), ('chain', 'ingo
t'), ('cheer', 'jolly'), ('clasp', 'raphe'), ('cogon', 'sewed'), ('commy', 'secco'),
('corky', 'wiles'), ('craal', 'penny'), ('credo', 'shute'), ('creel', 'perry'), ('cub
ed', 'melon'), ('curly', 'wolfs'), ('cushy', 'wombs'), ('danio', 'herms'), ('dated',
'spits'), ('dawted', 'splits'), ('dazed', 'spots'), ('didos', 'pupae'), ('dolls', 'wh
eel'), ('drips', 'octad'), ('ebony', 'uredo'), ('eches', 'kinky'), ('fadge', 'toru
s'), ('fagot', 'touch'), ('fills', 'lorry'), ('fizzy', 'kneed'), ('frena', 'wiver'),
('frere', 'serer'), ('fusion', 'layout'), ('ganja', 'kerne'), ('gassy', 'smeek'), ('g
inny', 'motte'), ('gnarl', 'xeric'), ('golem', 'murks'), ('golly', 'murre'), ('gree
n', 'terra'), ('gulfs', 'marly'), ('gulls', 'marry'), ('gummy', 'masse'), ('gunny',
'matte'), ('hints', 'styed'), ('hoggs', 'tasse'), ('hotel', 'ovals'), ('inkier', 'pur
ply'), ('jerky', 'snath'), ('jiffs', 'polly'), ('jimmy', 'posse'), ('jinni', 'pott
o'), ('jinns', 'potty'), ('johns', 'punty'), ('lallan', 'pepper'), ('later', 'shal
y'), ('latex', 'shale'), ('linum', 'rotas'), ('luffs', 'rally'), ('lutea', 'pyxie'),
('manful', 'thumbs'), ('mills', 'sorry'), ('mocha', 'suing'), ('molas', 'surgy'), ('m
uffs', 'sally'), ('mulch', 'sarin'), ('mumms', 'sassy'), ('munch', 'satin'), ('muumu
u', 'weewee'), ('noggs', 'tummy'), ('nulls', 'tarry'), ('nutty', 'tazze'), ('oxter',
'vealy'), ('pecan', 'tiger'), ('primero', 'sulphur'), ('pulpy', 'varve'), ('ratan',
'vexer'), ('sheer', 'tiffs'), ('sneer', 'toffs'), ('steeds', 'tuffet'), ('steer', 'tu
ffs'), ('teloi', 'whorl')]
```

In [10]:
```
has_duplicates(['a','b','b','c'])
```

Out[10]:
```
False
```

Exercise 12.2

Write a program that reads a word list from a file (see Section 9.1) and finds all the sets of words that are anagrams.

After finding all anagram sets, print the list of all anagram sets that have 6 or more entries in it.

Here is an example of what the output might look like:

```
['abets', 'baste', 'bates', 'beast', 'beats', 'betas', 'tabes']
['acers', 'acres', 'cares', 'carse', 'escar', 'races', 'scare',
'serac']
['acred', 'arced', 'cadre', 'cared', 'cedar', 'raced']
...
```

Hint:

First traverse the entire wordlist to build a dictionary that maps from a collection of letters to a list of words that can be spelled with those letters. The question is, how can you represent the collection of letters in a way that can be used as a key? i.e. The word "eat" and the word "tea" should be in a list associated with a key ('a','e','t').

In [11]:
```
def anagrams(wordlist):
    collection = {}
    for word in wordlist:
        key = tuple(sorted(word))
        if key in collection:
            collection[key] = collection[key] + [word]
        else:
            collection[key] = [word]
    for keys in collection:
        if len(collection[keys]) >= 6:
            print(collection[keys])
```

NumPy Exercises

For the following exercises, be sure to print the desired output. You will not receive credit for problems that do not print the desired output.

Some of these exercises will require the use of functions in numpy that may not have been covered in class. Look up documentation on how to use them. I always recommend checking the official reference at https://numpy.org/doc/stable/reference/index.html

In [12]:
```
import numpy as np
np.random.seed(1)
```

Exercise NP.1:

Task: Create an array `b` of 10 random integers selected between 0-99

Desired output: `[37 12 72  9 75  5 79 64 16  1]` of course yours might be different

In [13]:
```
b = np.random.randint(0, 100, 10)
print(b)
```

```
[37 12 72  9 75  5 79 64 16  1]
```

NP.1b:

Task: reverse the elements in `b`. Hint: Try slicing the array, but backwards

Desired output: `[ 1 16 64 79  5 75  9 72 12 37]` yours will be different

In [14]:
```
b = b[::-1]
print(b)
```

```
[ 1 16 64 79  5 75  9 72 12 37]
```

NP.2a:

Task: Create an array `c` of 1000 random values selected from a normal distribution centered at 100 with sd = 15, rounded to 1 decimal place. Print only the first 10 values.

Desired output: `[ 92.1  83.9 113.   65.5 126.2  88.6 104.8  96.3 121.9  69.1]` of course your values may be different

In [15]:
```
c = np.random.normal(100, 15, 1000).round(decimals = 1)
print(c[:10])
```

```
[ 92.1  83.9 113.   65.5 126.2  88.6 104.8  96.3 121.9  69.1]
```

NP.2b:

Perform a Shapiro-Wilk test to see if the values in c come from a normal distribution. Report the p-value and appropriate conclusion.

Look up `scipy.stats.shapiro` for usage and details.

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html

In [16]:
```
import scipy
from scipy import stats
```

In [17]:
```
scipy.stats.shapiro(c)
```

Out[17]:
```
ShapiroResult(statistic=0.9977032542228699, pvalue=0.17940226197242737)
```

The p-value is greater than 0.05, so there is not statistically significant evidence to reject the null hypothesis that the data was drawn from a normal distribution.

NP.2c:

Identify and print the values in `c` that are more than 3 standard deviations from the mean of `c`. Report the proportion of values that are more than 3 sd from the mean.

Desired output: `[ 54.1 ... 148.3 ]`

```
0.32 of the values are beyond 3 sd from the mean.
```

In [18]:
```
mean = 100
sd = 15
```

```
dist = abs(c - mean)
index = dist > 3*sd
outs = c[index] # values in c that are > 3 standard deviations from the mean
print(outs)
prop = len(outs)/1000 # proportion of values that are > 3 standard deviations from the
print(f"{prop} of the values are beyond 3 sd from the mean.")
```

```
[145.5 159.4 149.8  54.2]
0.004 of the values are beyond 3 sd from the mean.
```

NP.3:

Task: Make a 3x3 identity matrix called `I3`

Desired output:

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [19]:
```
from numpy import matlib
I3 = np.matlib.identity(3)
print(I3)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

NP.4a:

Task: Make a 10x10 array of values 1 to 100. Call it X

Desired output:

```
[[  1   2   3   4   5   6   7   8   9  10]
 [ 11  12  13  14  15  16  17  18  19  20]
 [ 21  22  23  24  25  26  27  28  29  30]
 [ 31  32  33  34  35  36  37  38  39  40]
 [ 41  42  43  44  45  46  47  48  49  50]
 [ 51  52  53  54  55  56  57  58  59  60]
 [ 61  62  63  64  65  66  67  68  69  70]
 [ 71  72  73  74  75  76  77  78  79  80]
 [ 81  82  83  84  85  86  87  88  89  90]
 [ 91  92  93  94  95  96  97  98  99 100]]
```

In [20]:
```
X = np.arange(1, 101)
X = X.reshape(10, 10)
print(X)
```

```
[[  1   2   3   4   5   6   7   8   9  10]
 [ 11  12  13  14  15  16  17  18  19  20]
 [ 21  22  23  24  25  26  27  28  29  30]
 [ 31  32  33  34  35  36  37  38  39  40]
 [ 41  42  43  44  45  46  47  48  49  50]
 [ 51  52  53  54  55  56  57  58  59  60]
 [ 61  62  63  64  65  66  67  68  69  70]
 [ 71  72  73  74  75  76  77  78  79  80]
 [ 81  82  83  84  85  86  87  88  89  90]
 [ 91  92  93  94  95  96  97  98  99 100]]
```

NP.4b:

Task: Make a copy of X, call it Y (1 line). Replace all values in Y that are not squares with 0 (1 line). see `numpy.isin()`

Desired output:

```
[[  1   0   0   4   0   0   0   0   9   0]
 [  0   0   0   0   0  16   0   0   0   0]
 [  0   0   0   0  25   0   0   0   0   0]
...
 [  0   0   0   0   0   0   0   0   0 100]]
```

In [21]:
```
Y = X
squares = np.arange(11) ** 2
mask = np.isin(Y, squares, invert = True)
Y[mask] = 0
print(Y)
```

```
[[  1   0   0   4   0   0   0   0   9   0]
 [  0   0   0   0   0  16   0   0   0   0]
 [  0   0   0   0  25   0   0   0   0   0]
 [  0   0   0   0   0  36   0   0   0   0]
 [  0   0   0   0   0   0   0   0  49   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0  64   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [ 81   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 100]]
```

NP.5:

Task: Use `np.tile()` to tile a 2x2 diagonal matrix of integers to make a checkerboard pattern. Call the matrix `checkers`

Desired output:

```
[[1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 ...
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]]
```

In [22]:
```
pattern = np.matlib.identity(2)
```

```
checkers = np.tile(pattern, reps = (4, 4)).astype(int)
print(checkers)
```

```
[[1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]]
```

NP.6:

Task: convert the values in `f_temp` from Farenheit to celsius. The conversion is subtract 32, then multiply by 5/9. Round to 1 decimal place.

Desired output:

```
[[21.1 21.7 22.2 22.8 23.3 23.9]
 ...
 [34.4 35.  35.6 36.1 36.7 37.2]]
```

In [23]:
```
# do not modify
f_temp  = np.arange(70,100).reshape((5,6))
```

In [24]:
```
c_temp = (f_temp - 32)*(5/9)
c_temp = c_temp.round(decimals = 1)
print(c_temp)
```

```
[[21.1 21.7 22.2 22.8 23.3 23.9]
 [24.4 25.  25.6 26.1 26.7 27.2]
 [27.8 28.3 28.9 29.4 30.  30.6]
 [31.1 31.7 32.2 32.8 33.3 33.9]
 [34.4 35.  35.6 36.1 36.7 37.2]]
```

NP.7:

Task: Convert values in the matrix `x` into z-scores by column, call it matrix `z`. That is: each column should have a mean of 0 and std of 1. (subtract the column mean, and divide by the column std). (not required, but see if you can do this in one line)

Print the column means and column std to show that they have been standardized.

Desired output:

```
[[-1.09996745 -0.47901666  0.8816739 ]
 [ 0.9495002   1.18844641  0.11324252]
...
 [-0.60705751 -1.08536687 -0.57430135]
 [-1.28156585 -0.81250928  1.52877401]]

 [ 6.66133815e-17  1.11022302e-16 -1.11022302e-16]
[1. 1. 1.]
```

In [25]:
```
# do not modify
np.random.seed(100)
x = np.random.randint(100,size = 30).reshape(10,3)
```

In [26]:
```
z = (x - np.tile(x.mean(axis = 0), reps = (10, 1))) / np.tile(x.std(axis = 0), reps =
```

In [27]:
```
print(z)
print(z.mean(axis = 0))
print(z.std(axis = 0))
```

```
[[-1.09996745 -0.47901666  0.8816739 ]
 [ 0.9495002   1.18844641  0.11324252]
 [-1.04808219  1.64320907  0.27501755]
 [ 1.23486912  0.40019114  0.84123014]
 [ 1.23486912 -0.78219177 -0.45297008]
 [-0.68488539 -0.75187426  0.5985676 ]
 [ 0.19716398 -0.72155675 -1.46406399]
 [ 1.10515598  1.40066898 -1.74717029]
 [-0.60705751 -1.08536687 -0.57430135]
 [-1.28156585 -0.81250928  1.52877401]]
[ 4.44089210e-17  1.11022302e-16 -1.11022302e-16]
[1. 1. 1.]
```

NP.8:

Task: Convert values in the matrix `x` into scaled values from 0 to 10. That is take each column and scale values linearly so that the largest value is 10, and the smallest value in the column is 0. Round results to 2 decimal places. Call the result `y`

(Not required, but see if you can do the calculations in one line.)

Desired output:

```
[[ 0.72  2.22  8.02]
 [ 8.87  8.33  5.68]
...
 [ 2.68  0.    3.58]
 [ 0.    1.   10.  ]]
```

In [28]:
```
# do not modify
np.random.seed(100)
x = np.random.randint(100,size = 30).reshape(10,3)
```

In [29]:
```
y = (x * 10) / np.tile(x.max(axis = 0), reps = (10, 1))
y = y.round(decimals = 2)
```

In [30]:
```
print(y)
```

```
[[ 0.82  2.55  8.07]
 [ 8.88  8.4   5.78]
 [ 1.02 10.    6.27]
 [10.    5.64  7.95]
 [10.    1.49  4.1 ]
 [ 2.45  1.6   7.23]
 [ 5.92  1.7   1.08]
 [ 9.49  9.15  0.24]
 [ 2.76  0.43  3.73]
 [ 0.1   1.38 10.  ]]
```

NP.9:

Task: Replace all NaN values in the matrix `x` with 0.

Desired output:

```
[[ 8. 24. 67.]
 [87. 79.  0.]
 [10.  0. 52.]
 [ 0. 53. 66.]]
```

In [31]:
```
# do not modify
np.random.seed(100)
x = np.random.randint(100,size = 12).reshape(4,3).astype('float')
row = np.array([1, 2, 3])
col = np.array([2, 1, 0])
x[row, col] = np.nan
```

In [32]:
```
mask = np.isnan(x)
x[mask] = 0
print(x)
```

```
[[ 8. 24. 67.]
 [87. 79.  0.]
 [10.  0. 52.]
 [ 0. 53. 66.]]
```