```
In [1]:
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Homework 1

Fill in your name and the names of any students who helped you below.

> I affirm that I personally wrote the text, code, and comments in this homework assignment.

> I received help from [other student] who gave me suggestions on [problem]. - [Maxim Matkovski]

This homework focuses on topics related to basic data types, collections, and iterations. For each problem, place your solution below the **Solution** headers.

Submitting the Assignment

Refer to Expectations for Assignments (https://nbviewer.jupyter.org/github/PhilChodrow/PIC16A/blob/master/content/preliminaries/expectati
for instructions on submitting this assignment. In particular, please attend to the following:

1. Complex blocks of code have comments to help the reader understand what is going on. Blocks of code of under five lines usually don't require comments.
2. Your assignment is submitted as a PDF on Gradescope. This PDF must be:
   - Paginated (split into multiple pages of normal size).
   - Marked on Gradescope. This entails indicating the page(s) on which each problem lies.
   - **The reader may decline to grade your homework if you do not paginate and mark your submission!**

Problem 1

(a)

Create a string variable `s` such that printing it yields the following result:

Hint: If you use triple quote delimiters `"""`, you'll be able to stretch the string across multiple lines without manually typing special characters. However, careful use of special characters is also acceptable.

```
print(s)
```

```
Tired    : Doing math on your calculator.
Wired    : Doing math in Python.
Inspired : Training literal pythons to carry out long division using an
 abacus.
```

In [14]:

```
# solutions using \n are also acceptable.

s = """
Tired    : Doing math on your calculator.
Wired    : Doing math in Python.
Inspired : Training literal pythons to carry out long division using an abacu

print(s)
```

```
Tired    : Doing math on your calculator.
Wired    : Doing math in Python.
Inspired : Training literal pythons to carry out long division using an aba
cus.
```

Next, write Python commands **which use `s`** to print the specified outputs. Feel free to use loops and comprehensions; however, keep your code as concise as possible. Each solution should require at most three short lines of code.

For full credit, you should minimize the use of positional indexing (e.g. `s[5:10]`) when possible.

(b)

Output:

```
Tired
Doing math on your calculator.
Wired
Doing math in Python.
Inspired
Training literal pythons to carry out long-division using an abacus.
```

In [15]:

```
# your solution here.

for word in s.split(":"):
    print(word)
```

```
Tired
 Doing math on your calculator.
Wired
 Doing math in Python.
Inspired
 Training literal pythons to carry out long division using an abacus.
```

(c)

Output:

```
Tired
Wired
Inspired
```

Hint: look at the endings of words. A small amount of positional indexing might be handy here.

In [4]:
```python
# your solution here.

s1 = s.split()

i = 0
while i < len(s1):
    if s1[i][-2:] == 'ed':
        print (s1[i])
    i = i + 1
```

```
Tired
Wired
Inspired
```

(d)

Output:

```
Tired     : Doing math on your calculator.
Wired     : Doing math in Python.
```

Hint: These two lines are shorter than the other one.

In [23]:
```python
# your solution here.

s2 = s.split('.')

for i in s2:
    if len(i) < 70:
        print(i)
```

```

Tired     : Doing math on your calculator

Wired     : Doing math in Python
```

(e)

Output (note the lack of quotation marks).

```
Tired    : Doing data science on your calculator.
Wired    : Doing data science in Python.
Inspired : Training literal pythons to carry out machine learning using
 an abacus.
```

Hint: `str.replace` .

In [8]:
```
# your solution here.

s = s.replace("math", "data science")
s = s.replace("long division", "machine learning")

print(s)
```

```
Tired    : Doing data science on your calculator.
Wired    : Doing data science in Python.
Inspired : Training literal pythons to carry out machine learning using an
abacus.
```

Problem 2

Write code which, given a list `L` , produces a dictionary `D` where:

- The *keys* of `D` are the unique elements of `L` (i.e. each element of `L` appears only once).
- The value `D[i]` is the number of times that `i` appears in list `L` .

Demonstrate that your code works for lists of strings, lists of integers, and lists containing both strings and integers. (it's fine to copy and paste your code for this, although if you happen to know how to define functions then you might find that more efficient). **You should test your code on all three cases and visibly show the results in your submitted PDF.**

Example

```
L = ["a", "a", "b", "c"]
# your code here
# ...
D
# output
{"a" : 2, "b" : 1, "c" : 1}
```

Attend to Efficiency

A good way to solve this problem is using the `list.count()` method. However, you should carefully check the structure of your code to ensure that you are not calling `list.count()` an unnecessary number of times. Consider the supplied example above: how many times should `list.count()` be called?

There are also other good solutions to this problem which do not use `list.count()` . Here as well, make sure that you are not performing unnecessary computations.

Solution

In [9]:

```
# your solution here.

L = ["a", "a", "b", "c"]

D = {}

for x in L:
    if (x in D):
        D[x] += 1
    else:
        D[x] = 1

D
```

Out[9]:

```
{'a': 2, 'b': 1, 'c': 1}
```

Problem 3

The `input()` function allows you to accept typed input from a user as a string. For example,

```
x = input("Please enter an integer. ")
# user types 7
x
# output
'7'
```

Write code that prompts a user to input an odd `int`. If the user inputs an even `int`, the code should re-prompt them for an odd integer. After the user has entered an odd integer, the code should print a list of all numbers that the user input.

The code `int("1")` will convert strings composed of numerals into integers. You may assume that the user will only input such strings.

Hint: Try `while` and associated tools.

Example

```
# your code

> Please enter an integer. 6
> Please enter an integer. 8
> Please enter an integer. 4
> Please enter an integer. 9
> [6, 8, 4, 9]
```

Solution

In [24]:
```python
# your solution here.

# create an empty list and a flag condition
List = []
flag = 0

# prompt for an intial input
print('Please enter an integer.')
x = int(input())

# while loop will repeat until odd integer entered
while(flag != 1):

    if (x%2 == 0):
        List.append(x)
        print('Please enter an integer.')
        x = int(input())

    else:
        List.append(x)
        print(List)
        flag = 1
```

```
Please enter an integer.
6
Please enter an integer.
8
Please enter an integer.
4
Please enter an integer.
9
[6, 8, 4, 9]
```

Problem 4

Write list comprehensions which produce the specified list. Each list comprehension should fit on one line and be no longer than 80 characters.

In [ ]:
```python
# 80 characters
```

(a)

A list of the first 10 triangular numbers (https://en.wikipedia.org/wiki/Triangular_number). The ith triangular number is the sum of the integers up to and including i. For example, the sixth triangular number is

$$1 + 2 + 3 + 4 + 5 + 6 = 21 .$$

The first element in your list should be 1, and the last one should be 55.

Note

Some of you may be familiar with the formula

$$1 + 2 + 3 + \cdots + (n-1) + n = \frac{n(n+1)}{2} \ ,$$

which is usually attributed to an 8-year-old Carl Friedrich Gauss (the same one after whom Gaussian elimination, Gauss's Theorem, Gaussian (normal) distributions, and many other mathematical constructs are named). **For full credit, you should not use this formula.**

Solution

In [25]:
```
# solution

triangle_numbers = [(n**2 + n)/2 for n in range (1,11)]

print(triangle_numbers)
```

```
[1.0, 3.0, 6.0, 10.0, 15.0, 21.0, 28.0, 36.0, 45.0, 55.0]
```

In [ ]:
```
# try without using the formula
```

(b)

A list of the letters in a predefined string `s`, **except for vowels.** For the purposes of this example, an English vowel is any of the letters `["a", "e", "i", "o", "u"]`. For example:

```
s = "make it so, number one"
# your code here
# ...
L
# output
["m", "k", "t", "s", "n", "m", "b", "r", "n"]
```

You should also exclude spaces. It is ok to include punctuation.

Hint: Consider the following code:

```
l = "a"
l not in ["e", "w"]
```

Solution

In [26]:
```
# your solution here.
# longer solutions acceptable if they also handle
# uppercase letters

s = s = "make it so, number one"

new_s = [letter for letter in s if letter not in ['a','e','i','o','u']]

print(new_s)
```

```
['m', 'k', ' ', 't', ' ', 's', ',', ' ', 'n', 'm', 'b', 'r', ' ', 'n']
```

(c)

A list `L` whose elements are themselves lists. The `i` th element of `L` contains the powers of `X[i]` from `0` to `k`, where `X` is a list and `k` an integer that can be specified by the user. For example,

```
X = [5, 6, 7]
k = 2
# your code here
# ...
L
# output
[[1, 5, 25], [1, 6, 36], [1, 7, 49]]
```

Solution

In [27]:
```
# your solution here.

X = [5,6,7]
k = 2

L = [[number**x for x in range (0,k+1)] for number in X]

L
```

Out[27]:
```
[[1, 5, 25], [1, 6, 36], [1, 7, 49]]
```

(d)

As in **(c)**, but now include only even powers of `X`. For example,

```
X = [5, 6, 7]
k = 8
# your code here
# ...
L
# output
[[1, 25, 625, 15625, 390625],
 [1, 36, 1296, 46656, 1679616],
 [1, 49, 2401, 117649, 5764801]]
```

Solution

In [28]:
```
# your solution here.

X = [5,6,7]
k = 8

L = [[number**x for x in range (0,k+1) if x%2==0] for number in X]

L
```

Out[28]:
```
[[1, 25, 625, 15625, 390625],
 [1, 36, 1296, 46656, 1679616],
 [1, 49, 2401, 117649, 5764801]]
```

Problem 5

In this problem, we'll simulate the *simple random walk*, perhaps the most important discrete-time stochastic process. Random walks are commonly used to model phenomena in physics, chemistry, biology, and finance. In the simple random walk, at each timestep we flip a fair coin. If heads, we move foward one step; if tails, we move backwards. Let "forwards" be represented by positive integers, and "backwards" be represented by negative integers. For example, if we are currently three steps backwards from the starting point, our position is `-3` .

Write code to simulate a random walk. Your code should:

- Specify an upper and lower bound -- once the walk reaches one of these two numbers, the process terminates.
- Keep a log of the results of the coin flips.
- Keep a log of the position of the walk at each time step.
- When the walk reaches the upper or lower bound, print a message and terminate the walk.

To simulate a coin flip, run the following code once:

```
import random
```

Then, you can let `x` be the value of a fairly flipped coin by writing:

```
x = random.choice(["heads","tails"])
```

You don't have to use `"heads"` and `"tails"` when using this function -- can you think of a more useful choice set?

Your code should include at least one instance of an `elif` statement and at least one instance of a `break` statement. (It is possible to write a good version of this code without these constructs; however, I am specifically asking you to demonstrate their use).

Example

```
import random

upper_bound = 10
lower_bound = -10

# your code here
# ...

# message
Upper bound at 10 reached

positions
# output
[1, 2, 1, 2, 1, 0, 1, 2, 3]
```

For full credit, you must display the list of previous positions by typing out the `positions` variable as above.

Optionally, you may also enjoy visualizing the random walk. After your main code, run the following two lines to produce a plot. When the bounds are set very large, the resulting visualization can be quite intriguing and attractive. It is not necessary for you to understand the syntax of these commands at this stage -- we'll cover them later in the course.

Solution

In [84]:

```
# your solution here.

import random

#create bounds and positions
upper_bound = 10
lower_bound = 10

positions = []
position = 0



while True:
    x = random.choice ([-1, 1])
    position = position + x
    positions.append(x)

    if position == upper_bound:
        print('Upper bound at', upper_bound, 'reached')
        break
    if position == lower_bound:
        print('Lower bound at', lower_bound, 'reached')
        break

print('positions length: ', len(positions))
```

```
Upper bound at 10 reached
positions length:  60
```

In [85]:
```
# run this to check that your positions list looks ok

print('positions length: ', len(positions))
positions
```

```
positions length:  60
```

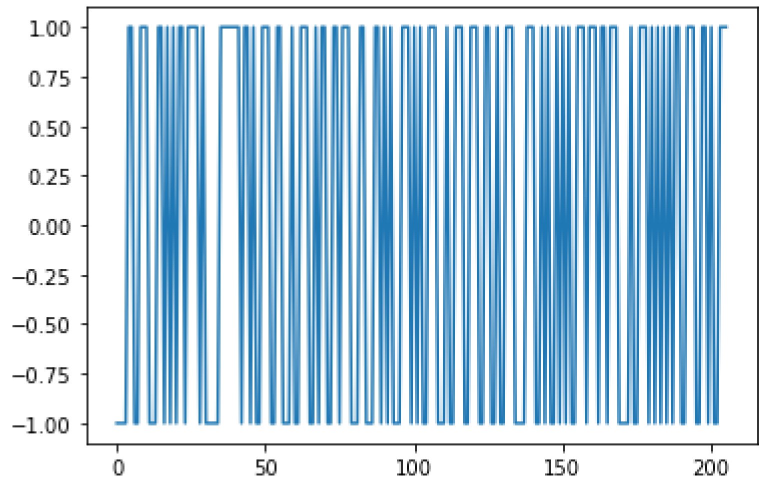
Out[85]:
```
[1,
 1,
 1,
 -1,
 1,
 -1,
 1,
 1,
 1,
 1,
 -1,
 -1,
 1,
 -1,
 -1,
 -1,
 -1,
 1,
 -1,
 1,
 1,
 -1,
 1,
 1,
 1,
 1,
 -1,
 1,
 -1,
 1,
 1,
 1,
 -1,
 -1,
 1,
 1,
 1,
 -1,
 1,
 -1,
 -1,
 1,
 -1,
 1,
 1,
 -1,
 1,
 -1,
 -1,
 -1,
 1,
 -1,
 -1,
 -1,
 1,
 1,
 1,
 1,
 1,
 1]
```

In [95]:
```python
# A longer walk
# run your code here again and increase the number of steps (if you'd like)
# then, run the code below
# your solution here.

import random

#create bounds and positions
upper_bound = 10
lower_bound = 10

positions = []
position = 0



while True:
    x = random.choice ([-1, 1])
    position = position + x
    positions.append(x)

    if position == upper_bound:
        print('Upper bound at', upper_bound, 'reached')
        break
    if position == lower_bound:
        print('Lower bound at', lower_bound, 'reached')
        break

print('positions length: ', len(positions))
```

```
Upper bound at 10 reached
positions length:  206
```

In [96]:
```
from matplotlib import pyplot as plt
plt.plot(positions)
```

Out[96]:
```
[<matplotlib.lines.Line2D at 0x281e828a790>]
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200