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 Rashi and Lauren who gave me suggestions on 1 and 4.

- David Nguyen

This homework focuses on topics related to basic data types, collections, and iterations. For each problem, place your solution below the **Solution** headers. Refer to Expectations for Assignments (https://nbviewer.jupyter.org/github/PhilChodrow/PIC16A/blob/master/content/preliminaries/expectations_for_assignments.ipynb) for instructions on submitting this assignment. For this assignment, it is not necessary to write docstrings for any of your code, but you should include comments and descriptions of your general approach.

Submit this assignment on GradeScope.

Problem 1

(a)

Create a string variable `s` such that the following works:

You may need to use one or more special characters.

```
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 [1]:
```
# your solution
a = 'Tired'
b = 'Wired'
c = 'Inspired'
aa = 'Doing math on your calculator.'
bb = 'Doing math in Python.'
cc = 'Training literal pythons to carry out long division using an abacus.'

#combine all the strings above to make s
s = a + '\t : ' + aa + '\n' + b + '\t : ' + bb + '\n' + c + ' : ' + cc
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.
```

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 [2]:
```
# your solution here.
s = a + '\n' + aa + '\n' + b + '\n' + bb + '\n' + c + '\n' + cc
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.
```

(c)

Output:

```
Tired
Wired
Inspired
```

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

In [3]:
```
# your solution here.
s = a+'\n'+b+'\n'+c
print(s)
```

```
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 [4]:
```
s = a + '\t : ' + aa + '\n' + b + '\t : ' + bb
print(s)
```

```
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 [9]:
```
# your solution here.
s = a + '\t : ' + aa + '\n' + b + '\t : ' + bb + '\n' + c + ' : ' + cc
temp1 = s.replace('math', 'data science')
temp2 = temp1.replace('long division', 'machine learning')
s = temp2
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).

For example:

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

Solution

In [133]:
```
A = input('please input a list: ')
#L = A.replace(" ",'')
L = A.split()
myList = [] #initialize empty list

# go through the current list and create list for keys
for x in range(len(L)):
    if(L[x] not in myList):
        myList.append(L[x])


values = [] #empty list for values

# go through list L and create list for values
for x in range(len(myList)):
    count = 0

    #for loop to check for values
    for y in range(len(L)):
        if L[y] == myList[x]:
            count = count + 1 #increment

    values.append(count)



#putting the two list together to make a dict
D = {myList[i]:values[i] for i in range(len(myList))}


print(D)
```

```
please input a list: a a a a b b c c c c d e e e
{'a': 4, 'b': 2, 'c': 4, 'd': 1, 'e': 3}
```

Problem 3

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

```
x = input("Please input a number.")
# 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 [69]:
```
print('Note: Input an odd number to end the process')

#initialize empty list
newList = []

#while loop
while True:
    num = input('Please enter an integer: ') #take input
    num = int(num) #change from string to int
    newList.append(num)
    if num%2 == 1: #check if odd or even
        break

print(newList)
```

```
Note: Input an odd number to end the process
Please enter an integer: 2
Please enter an integer: 4
Please enter an integer: 2
Please enter an integer: 6
Please enter an integer: 10
Please enter an integer: 3
[2, 4, 2, 6, 10, 3]
```

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 [82]:
```
                                                                    # 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.

Solution

In [9]:
```
triNum = [(n*(n+1))/2 for n in range(1,11)]
print(triNum)
```

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

(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 [31]:
```
s = input()
vowels = ['a','e','i','o','u']
L = [n for n in s if n not in vowels and n.isalnum()]
print(L)
```

```
"make it so, number one"
['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 [53]:
```
X = input("Please input a list of numbers: ")
X = X.split()
X = [int(n) for n in X]
k = int(input("Please input an exponent: "))

L = [[base**exp for exp in range(k+1)] for base in X]
print(L)
```

```
Please input a list of numbers: 5 6 7
Please input an exponent: 2
[[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 [54]:
```
X = input("Please input a list of numbers: ")
X = X.split()
X = [int(n) for n in X]
k = int(input("Please input an exponent: "))

L = [[base**exp for exp in range(k+1) if exp%2 == 0] for base in X]
print(L)
```

```
Please input a list of numbers: 5 6 7
Please input an exponent: 8
[[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.

Example

```
import random

upper_bound = 3
lower_bound = -3

# your code here
# ...

# message
Upper bound at 3 reached

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

Finally, you might be interested in visualizing the walk. Optionally, 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.

In [102]:
```
from matplotlib import pyplot as plt
plt.plot(positions)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-102-2544af150976> in <module>
      4 lower_bound = -3
      5 from matplotlib import pyplot as plt
----> 6 plt.plot(positions)

NameError: name 'positions' is not defined
```

Solution

In [6]:
```
import random

upper_bound = int(input('Upper Bound: '))
lower_bound = int(input('Lower Bound: '))

step = 0 #by default
stepList = [] #initialize list

while True:
    #generate random choice
    x = random.choice(['0','1'])

    #check if moving up or down
    if x == '1':
        step += 1
    elif x == '0':
        step -= 1

    stepList.append(step)

    if step == upper_bound:
        print('\nUpper bound at ' + str(upper_bound) + ' reached!')
        break
    elif step == lower_bound:
        print('\nLower bound at ' + str(lower_bound) + ' reached!')
        break


print('Positions:')
print(stepList)
```

```
Upper Bound: 3
Lower Bound: -3

Lower bound at -3 reached!
Positions:
[1, 0, -1, -2, -1, -2, -3]
```

In [ ]:

In [ ]: