## Important

1. Due Date: **Thursday, February 2nd**
2. This homework is graded out of 100 points.
3. This is an Individual Assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 1301, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.
4. For Help:
    - TA Helpdesk (Schedule posted on class website.)
    - Email TA's or use Piazza Forums Notes:
    - How to Think Like a Computer Scientists [ http://openbookproject.net/thinkcs/python/english3e/ ]
    - CS 1301 Python Debugging Guide [ http://www.cc.gatech.edu/classes/AY2016/cs1301_spring/CS-1301-Debugging-Guide/index.html ]
5. Don't forget to include the required collaboration statement (outlined on the syllabus). Failing to include the Collaboration Statement will result in no credit.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. Comment or delete all your function calls. When your code is run, all it should do is build without any errors. Having function calls or extraneous code outside the scope of functions will result in a lower grade. (import statements and global variables are okay to be outside the scope of a function)
8. **Read the entire specifications document before starting this assignment.**
9. **IF YOUR CODE CANNOT RUN BECAUSE OF AN ERROR, IT IS A 0%.**

## Introduction

The goal of this homework is for you to implement the different types of loops you have learned. The functions you are asked to code will all use some looping technique and certain loops may be better fit for a certain purpose. Although some of the functions can be completed without the use of a loop, you are being graded on your ability to implement different looping techniques. Refer to the rubric to see how points will be rewarded for each function. You have been given HW3.py to fill out with instructions in the docstrings. However, below you will find more detailed information to complete your assignment. Read it thoroughly before you begin. You have until Thursday, February 2nd to complete this assignment.

Authors: Nikita Bawa and Cory Brooks | Report Issues: [nbawa3@gatech.edu or corybrooks@gatech.edu ]

Function name: **`countLetter`**
Parameters: **`string to be iterated through` (str), letter to count the occurrences of (str)**
Return value: **`value representing the number of times the given letter appears in the string (int)`**
Description:
Write a function that takes in two parameters, one being a string of any length, and the other being an individual letter to count the occurrences of. Uppercase and lowercase letters should both be counted. If the letter does not appear in the sentence, return 0. **Must use a for-each loop.** For example:

```
>>> countLetter("hello", "l") —› 2
>>> countLetter("Good Morning!", "o") —› 3
>>> countLetter("Computer Science Class", "c") —› 4
```

Hint: The lower() string function might help in this function.

Function name: **`encryptMessage`**
Parameters: **`string to encrypt (str)`**
Return value: **`encrypted string (str)`**
Description:
You are working on a top secret project for which you have to encrypt words that come through. Write a function that takes in the word to be encrypted as a parameter and returns the encrypted word. The rules for encryption are as follows:

1. "a" or "A" → "@"
2. "s" or "S" → "$"
3. "h" or "H" → "#"
4. Numbers should not be included in the encrypted string
5. "!" indicates to stop the encryption

Here are some test cases to show you how the function should work:

```
>>> encryptMessage("bananas45") —› "b@n@n@$"
>>> encryptMessage("ham!burger") —› "#@m"
>>> encryptMessage("hersheys64kisses") —› "#er$#ey$ki$$e$"
```

Hint: Try using break and continue to help with implementing some of the encryption rules.

Function name: **encryptSentence**
Parameters: **sentence to encrypt (str)**
Return value: **encrypted sentence (str)**
Description:
For the same top secret project, you are sometimes posed with the problem of encrypting sentences. To encrypt a sentence, only words that begin with a capital letter should be encrypted. You should call your encryptMessage() from above to encrypt words in the sentence.
Here are some test cases to show you how the function should work:

```
>>> encryptSentence("This is top Secret") —› "T#i$ is top $ecret"
>>> encryptSentence("rise up Falcons") —› "rise up Falcon$"
>>> encryptSentence("CS1301 is So fun") —› "C$ is $o fun"
```

Hint: Use the ord() function to determine whether a letter is a capital letter or not.

Function name: **stringPartition**
Parameters: **a word (str)**
Return value: **modified word (str)**
Description:
Write a function that takes in a word as a parameter and returns a modified version of the word in which all letters at an even index are in the first half of the string and all letters at an odd index are in the second half of the string. The letter at index 0 should be the first letter in the modified word.
Here are some test cases to show you how the function should work:

```
>>> stringPartition("kangaroo") —› "knaoagro"
>>> stringPartition("falcons") —› "flosacn"
```

Function name: **sumEvens**
Parameters: **an upperbound for a range (int)**
Return value: **value of all even values in the range summed together (int)**
Description:
Write a function that accepts an upper bound from a range from zero to that upper-bound. That upper bound is not included in the range. After calculating the range, your function should sum up all even values within the range.
Here are some test cases to show you how the function should work:

```
>>> sumEvens(6) —›   6
>>> sumEvens(2) —›   0
>>> sumEvens(11) —›  30
```

Function name: **vowelsAndConsonants**
Parameters: **a word (str)**
Return value: **original word ordered by vowels and consonants (str)**
Description:
Write a function that takes a string as a parameter, and separates the vowels and consonants from one another. It should return one string, consisting of the same letters of the original string, but with all of the vowels at the front of the string, and all of the consonants at the back of the string. **Must iterate through the string.**

Here are some test cases to show you how the function should work:

```
>>> vowelsAndConsonants('apple') —›  "aeppl
>>> vowelsAndConsonants('mississippi') —›  "iiiimssssspp"
>>> vowelsAndConsonants("logarithm") —›  "oailgrthm"
```

Hint: String concatenation might help for this function.

Function name: **guessNumber**
Parameters: **final answer (int)**
Return value: **number of guesses the user took (int)**
Description:
Write a function that takes an integer value in and asks the user to try to guess this answer. When the user guesses the correct value, print a congratulatory statement and tell them how many guesses it took. If the user inputs "quit" (exactly the string quit, don't worry about edge cases like 'QUIT'), your code should end, and print the correct answer. The integer returned if the user quits should be -1. **You must use a while-loop.**

Here are some test cases to show you how the function should work:

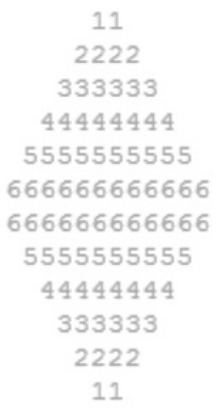
```
>>> a = guessNumber(5)
>>> "What is your guess?" —›  2
>>> "Wrong answer, try again" —›  4
>>> "Wrong answer, try again" —›  5
>>> "Correct! It took you 3 tries."
_____________________________________
>>> print(a)
>>> 3

>>> a = guessNumber(5)
>>> "What is your guess?" —›  2
>>> "Wrong answer, try again" —›  4
>>> "Wrong answer, try again" —›  quit
>>> "The correct answer was 5."
_____________________________________
>>> print(a)
>>> -1
```

Function name: **`diamond`**
Parameters: **`a number 2-9 specifying half the width of the diamond (int)`**
Return value: **`N/A`**
Description:
Write a function that takes in a number specifying half the width of the diamond as a parameter and prints a diamond of those specifications. Make sure the diamond prints in the correct format as shown in the example below. **Do not hardcode this. You must use a for-loop.**

Here is an example of how the function should work:

```
>>> diamond(6)
```



```
     11
    2222
   333333
  44444444
 5555555555
666666666666
666666666666
 5555555555
  44444444
   333333
    2222
     11
```

Hint: Use two for-loops – one for the top half and one for the bottom half

Function name: **`countDivisibility`**
Parameters: **`lower-bound value (int), upper-bound value (int)`**
Return value: **`N/A`**
Description:
Write a function that takes two parameters as the lower and upper bound of a range of intervals. The upper bound is not inclusive. Within this range, your code should count the number of values that are either only divisible by 3, only divisible by 5, or divisible by both 3 and 5. Each value can only satisfy one circumstance. For example, 15 can only be counted towards being divisible by both 3 and 5. Your code should then print the counts of divisibilities in an informative way.
Here are some test cases to show you how the function should work:

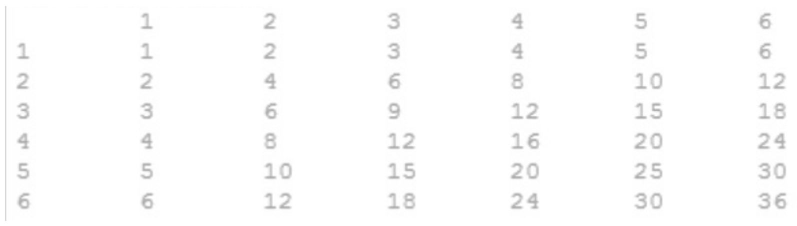
```
>>> countDivisibility(3,10)
>>> “3 number(s) divisible by three, 1 number(s) divisible by five,
     0 number(s) divisible by both”
>>> countDivisibility(12,20)
>>> “2 number(s) divisible by three, 0 number(s) divisible by five
     1 number(s) divisible by both”
```

Hint: When using range(a,b), a is included in the range, but b is not.

Function name: **`multiTable`**
Parameters: **`number to make the multiplication table for (int)`**
Return value: **`N/A`**
Description:
Write a function that takes in a number as a parameter and prints the multiplication table for that number. The table should be printed in an easy to read format. The first row and column should display the numbers from 1 – number specified. The top left corner of the table should remain blank as shown in the example below. **Do not hard code this. You must use a for-loop.**

Here are some test cases to show you how the function should work:

```
>>> multiTable(6)
```



```
	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	8	10	12
3	3	6	9	12	15	18
4	4	8	12	16	20	24
5	5	10	15	20	25	30
6	6	12	18	24	30	36
```

Hint: Try using a nested for-loop. Use tab characters to space out the grid properly.

## Grading Rubric

- **`countLetter`**: 5 points

| | |
|---|---|
| Passes all test cases | 5 points |
| Does not use a for-each loop | -3 points |

- **`encryptMessage`**: 5 points

| | |
|---|---|
| Passes all test cases | 5 points |
| Does not use a for loop | -3 points |

- **`encryptSentence`**: 10 points

| | |
|---|---|
| Passes some test cases | 5/10 points |
| Passes all test cases | 10/10 points |
| Does not call encryptMessage() | -3 points |

- **`stringPartition`**: 10 points

| | |
|---|---|
| Passes some test cases | 5/10 points |
| Passes all test cases | 10/10 points |
| Does not use a for loop | -3 points |

- **`sumEvens`**: 10 points

| | |
|---|---|
| Passes some test cases | 5/10 points |
| Passes all test cases | 10/10 points |

- **`guessNumber`**: 10 points

| | |
|---|---|
| Passes some test cases | 5/10 points |
| Passes all test cases | 10/10 points |
| Does not use a while loop | -3 points |

- **`vowelsAndConsonants`**: 10 points

| | |
|---|---|
| Passes some test cases | 5/10 points |
| Passes all test cases | 10/10 points |
| Does not call iterate string | -3 points |

- **`diamond`**: 15 points

| | |
|---|---|
| Passes some test cases | 5/15 points |
| Passes all test cases | 15/15 points |
| Does not use a loop | -10 points |

- **`countDivisibility`**: 10 points

| | |
|---|---|
| Passes some test cases | 5/10 points |
| Passes all test cases | 10/10 points |

- **`multiTable`**: 15 points

| | |
|---|---|
| Passes some test cases | 5/15 points |
| Passes all test cases | 15/15 points |
| Does not use a loop | -10 points |
| Incorrect format | -7 points |

## Provided

The following file(s) have been provided to you. There are several, but you will only edit one of them:

1. `HW3.py`
   This is the file you will edit and implement. All instructions for what the methods should do are in the docstrings.

## Deliverables

You must submit all of the following file(s). Please make sure the filename matches the filename(s) below. Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder and run them.

1. `HW3.py`