Homework 3 - Iteration

CS 1301 - Intro to Computing - Fall 2020

Important

- Due Date: **Tuesday, September 8th, 11:59 PM**.
- This is an individual assignment. High-level collaboration is encouraged, **but your submission must be uniquely yours.**
- Resources:
  - TA Helpdesk
  - Email TA's or use class Piazza
  - How to Think Like a Computer Scientist
  - CS 1301 YouTube Channel
- Comment out or delete all function calls. Only import statements, global variables, and comments are okay to be outside of your functions.
- **Read the entire document before starting this assignment.**

The goal of this homework is for you to practice and understand how to write and use loops and iteration. The homework will consist of 5 functions for you to implement. You have been given `HW03.py` skeleton file to fill out. However, below you will find more detailed information to complete your assignment. Read it thoroughly before you begin.

Hidden Test Cases: In an effort to encourage debugging and writing robust code, we will be including hidden test cases on Gradescope for some functions. You will not be able to see the input or output to these cases. Below is an example output from a failed hidden test case:

```
Test failed: False is not true
```

Written by Assata Quinichett (aquinichett@gatech.edu), Grace McDonough (gmcdonough3@gatech.edu), and Alex King (aking1@gatech.edu)

Helpful Methods

A method is a special type of function that we'll cover in more detail later on in the course. All you need to know for this homework is that methods are called through the `.` operator on a string or a variable that contains a string.

One useful method for this homework assignment is **.isdigit()**. This method returns `True` if all of the characters in the string are numerical digits, and `False` if they are not.

```
>>> "1".isdigit()
True
>>> "A".isdigit()
False
>>> "2Cool".isdigit()
False
```

Other helpful methods include **.lower()** and **.upper()**. These methods will return a new version of the string it was used on, this time in all lowercase or all uppercase, respectively.

```
>>> string = "HanNaH MonTANA"
>>> newString = string.lower()
>>> print(string)
HanNaH MonTANA
>>> print(newString)
hannah montana
```

Movie Night

Function Name: movieNight()
Parameters: a caption (`str`)
Returns: the fixed caption (`str`)
Description: After a long day of classes, you and your friends decide to get together for a movie night. You stream the movie from a very legitimate source, but you run into a problem: the movie captions contain random numbers! Write a function that takes in a caption (`str`) and returns the fixed caption without any numbers.

```
>>> caption = "Mr. and M4rs. Dursley of nu28mber four, Privet Drive, wer903e
proud to say th6at they we6re perfectly norm3al, tha894nk you ve89ry much."
>>> movieNight(caption)
'Mr. and Mrs. Dursley of number four, Privet Drive, were proud to say that they
were perfectly normal, thank you very much.'
```

Ice Cream

Function Name: iceCream()
Parameters: flavor (`str`), number of vowels (`int`)
Returns: a sentence (`str`)
Description: After a couple of movies, you and your friends start craving ice cream. Unfortunately, you are really picky and can only have flavors that contain a certain number of vowels. Write a function that takes in a flavor (`str`) and a number of vowels (`int`), and returns a sentence that indicates whether you can eat this flavor or not. If the ice cream flavor has more vowels than the number passed in, return a string in the following format:

```
'Yes, {flavor} ice cream has more than {number} vowels!'
```

Otherwise, return a string in the following format:

```
'No, {flavor} ice cream doesn't have more than {number} vowels!'
```

Note: The flavor in the returned string should be converted to all lowercase.
Note: The vowels are any characters in "aeiou"

```
>>> iceCream("ChoCoLaTe", 3)
'Yes, chocolate ice cream has more than 3 vowels!'
```

```
>>> iceCream("strawBERRY", 2)
'No, strawberry ice cream doesn't have more than 2 vowels!'
```

Dream Car

Function Name: dreamCar()
Parameters: car price (`float`), bank balance(`float`), interest rate (`float`)
Returns: number of years (`int`)
Description: While you're eating your ice cream, you start thinking about your dream car. You want to figure out how long it will take for you to save up and buy that car. Luckily, you've got some money in the bank that's growing every year with compound interest. Given the price of your dream car (`float`), the money you currently have in the bank (`float`), and the interest rate of your bank account (`float`) **given as a percent**, return how many years it will take for you to have enough money for your dream car.

Note: All parameters will be positive numbers.

```
>>> dreamCar(50000.0, 10000.0, 5.0)
33
```

```
>>> dreamCar(100000.0, 112.15, 2.1)
327
```

Battleship

Function Name: battleship()
Parameters: board size (`int`)
Returns: None (`NoneType`)
Description: After realizing how long it will take you to buy your dream car, you decide to play a game of Battleship to pass the time. In Battleship, each space on the board is a coordinate, marked with a letter and a number. The letter starts from 'a', and the number starts from '1'. When given a board size, generate a Battleship board for you and your friends to play on. **You should only print the board.** You will not be given a board size greater than 26.

```
>>> battleship(3)
a1 a2 a3
b1 b2 b3
c1 c2 c3
```

```
>>> battleship(6)
a1 a2 a3 a4 a5 a6
b1 b2 b3 b4 b5 b6
c1 c2 c3 c4 c5 c6
d1 d2 d3 d4 d5 d6
e1 e2 e3 e4 e5 e6
f1 f2 f3 f4 f5 f6
```

Tennis Match

Function: tennisMatch()
Parameters: player 1 (`str`), player 2 (`str`), match record (`str`)
Returns: winner (`str`)
Description: After Battleship, you decide to finish the day by playing tennis with your friends. Each tennis match consists of several games. To keep track, you record the match in a string. In the string, each game is separated by a dash: `-`. Each game will contain 1s and 2s, where a 1 represents a point scored by player 1, and a 2 represents a point scored by player 2. Whoever has the most points in a game wins that game. If they have the same number of points, that game is not counted. There can be any number of games, and any number of points per game. Return a string containing the winner and the score in the following format:

```
{winner} won! The score was {winning player games}-{losing player games}.
```

If the match is a tie, return the string: `It's a tie!`

```
>>> tennisMatch("Arvin", "Arushi", "11221-222-1111-11121-22111-")
'Arvin won! The score was 4-1.'
```

```
>>> tennisMatch("Anthony", "Caitlin", "1122-22211-11122-1212-")
'It's a tie!'
```

Grading Rubric

Function	Points
movieNight()	15
iceCream()	20
dreamCar()	20
battleship()	20
tennisMatch()	25
Total	**100**

Provided

The `HW03.py` skeleton file has been provided to you. This is the file you will edit and implement. All instructions for what the functions should do are in this skeleton and this document.

Submission Process

For this homework, we will be using Gradescope for submissions and automatic grading. When you submit your `HW03.py` file to the appropriate assignment on Gradescope, the autograder will run automatically. The grade you see on Gradescope will be the grade you get, unless your grading TA sees signs of you trying to defeat the system in your code. You can re-submit this assignment an unlimited number of times until the deadline; just click the "Re-submit" button at the lower right-hand corner of Gradescope. You do not need to submit your `HW03.py` on Canvas.