Discrete Mathematics Spring 2021

Coding Assignment 1

Due Date: February 10, 2021

Academic Honesty Policy:

Assignments are **to be completed individually** or **in a group with one other student**. If you are completing the assignment in a group, clearly specify your teammate at the top of your code in the space provided. No other collaboration is permitted, unless otherwise specified. Please do not include any of your code snippets or algorithms in public Piazza posts. You cannot not use solutions from any external source. You are not permitted to publish code or solutions online, nor post the course questions on forums including stack overflow. We run plagiarism detection software. If you have any questions at all about this policy please ask the course staff before submitting your assignment.

Read Carefully:

- Feel free to import any standard Python libraries you need as far as output and input of functions meet the requirement. Use the provided template to implement the functions.
- You **must name your file** "coding1.py". Any submission that does not follow this naming will not be graded.
- You **must make sure your file extension is .py.** Remember that online environments like Google Colab might export a **.ipynb** file; this is **not the correct filetype.** Colab (and other environments) have the option to export as **.py.**
- A skeleton of each function has been provided to you. **You are expected to ONLY write code in the functions and blocks specified.** In any case, DO NOT modify the function signatures, return variables, or any code that is not specified to be modifiable (we will include comments in the code to distinguish these sections) for any reason. Also, though you may modify the `main` function to test other cases, **do not turn in an assignment with a modified main function, i.e. anything below the line shown below.** This will be run by our autograder, so any unexpected modifications that make it malfunction **will receive not receive points.**

```
### DO NOT TURN IN AN ASSIGNMENT WITH ANYTHING BELOW HERE MODIFIED ###
if __name__ == "__main__":
```

- Test cases will be provided in the `main` function in the code. Three of the test cases for each part will be provided to you, and the others will be hidden test cases on our side (all graded via autograder).
- **Only Python 3.x versions will be graded.** To check your Python version locally, open a terminal window and enter:

```
python --version
```

  Google Colab should use Python 3 by default, but, to check, under "Runtime," navigate to "Change runtime type."
- To receive points, make sure your code runs. We recommend using Spyder, Pycharm or Google colab. They all allow you to download .py files. Be aware that if you write your code in some platforms like Codio and copy and paste it in a text file, there may be spurious characters in your code, and your code will not compile. **Always ensure that your .py compiles. Code that does not run will not receive any points.**

In this assignment, you will be learning the basics of Python by implementing some basic principles you've likely seen in previous classes. Because the class only assumes coding knowledge in some language (not necessarily Python), we assume that you are familiar with basic data structures, syntax, and control flow for code. If not, please review those concepts first.

Like in most of programming, **Google is your friend.** Of course, DO NOT look up answers/implementations directly on Google, but looking up certain Python syntax that helps you translate from whatever language you already know to Python is perfectly acceptable and natural in the learning process. The rule of thumb: if you're not looking up something on Google that trivializes the problem, then you're welcome to learn more about Python by searching up how certain componenents/tools you might need work. Learning by doing is the best way to learn programming.

It might be helpful before attempting the assignment to either **attend our Python recitation**, **watch the Python recitation** (it is/will be recorded), or **watch some other Python basic syntax video.** Here is a good starter video on the syntax: `https://www.youtube.com/watch?v=H1elmMBnykA`.

You will learn how to use the following concepts in Python:

- Loops/Control Flow.
- Lists (Arrays in other languages).
- Dictionaries (Hash Tables/Maps in other languages).
- String Manipulation.
- Basic Recursion.

Part A: Vowel Counting

In the following, we will be writing a simple function that should familiarize you with string manipulation and basic control flow in Python.

(a) Write a function `vowel_counter` that takes a `string s` as input and counts the number of vowels in `s` returning an `int`. For this exercise, we take the vowels to be the following: `[ a, e, i, o, u]`. You may assume that the entire string is lowercase characters in a - z (no spaces, numbers, special characters, etc.)

Your Python function `vowel_counter` should take one argument `s`. It should return one integer for the number of vowels.

As with all the other problems in this homework, make sure your return types **match exactly as specified. If your return types do not match, they will not be autograded.** Here, you will be returning one object: `int`.

```
def vowel_counter(s):
    # WRITE YOUR CODE HERE
    return #number of vowels (int).
```

Hints:

- You will need to be familiar with Python **lists** (which are just arrays), **for loops**, **if statements,** and **basic string manipulation** for this exercise. The following are links to the documentation of each, for your convenience:
  - `https://docs.python.org/3/tutorial/datastructures.html`
  - `https://docs.python.org/3/tutorial/controlflow.html`
  - `https://docs.python.org/3/library/string.html`
- Strings in Python behave out of the box as lists of characters. This will be useful in iterating over a string.
- Like in all programming, printing statements to debug is a useful tool. `print()` allows you to do just that.

(b) Write a function `sometimes_y` that implements the annoying English rule that `y` is sometimes a vowel and sometimes not. We will go with a simplified rule for whether `y` is a vowel or not: **If 'y' appears at the end of the string, then it is a vowel. Otherwise, it is not.**

Instead of rewriting your previous function, however, we'll get some practice calling other functions you have already written. For this exercise, you must use `sometimes_y` to call `vowel_counter` to determine the vowel count (do not duplicate the code from before). Then, you should just write some additional logic to compensate for the 'y' condition.

Your function should take a `string s` as input and return **three outputs**: whether 'y' is in the string or not (`Boolean`), the number of original vowels from `vowel_counter` (`int`), and the number of vowels with the additional rule of 'sometimes y' (`int`). Notice that Python allows you to return multiple outputs (of varying datatypes) by simply separating them with a comma in the return line. Useful!

For example, the string `'abcdef'` would return: `False, 2, 2`. The string `'abcdefy'` would return: `True, 2, 3`. The string `'yabcdef'` would return: `True, 2, 2`.

```
def sometimes_y(s):
    # WRITE YOUR CODE HERE
    return # y is in string (Boolean), number of vowels originally (int), number of
    vowels with y rule (int)
```

Hints:

- Be careful not to return a `list`. You want to use Python's built-in functionality to return three objects.
- To call another function you've written on some input `x`, it suffices to just write: `function(x)` in your code.

(c) Write a function `sentence_counter` that takes a `string sentence` as input and outputs a single `list` containing the number of vowels in each word of the sentence.

As before, you should use your previously implemented `sometimes_y` to accomplish this. **You must adhere to the 'sometimes y' rule.**

Unlike Part (a) you **may not assume** that your entire string is just lowercase alphabetical characters. This will give you some experience in cleaning messy strings up. However, you *can* assume the following conditions:

1. Spaces will separate each word in the sentence.
2. Special characters will be limited to: '.' (period), ',' (comma), '!' (exclamation), '?' (question mark).
3. There will be no numbers in the sentence.
4. Capital letters are allowed (which still increase your count if they are a vowel).

You may ask clarification questions about special cases on Piazza.

For example, `"The boy, Sam, walked to the store."` and `"I went to office hours, and the TAs were so friendly!"` are valid sentences. `"I love that Terminal 5 is hosting Bacchanal this year"` and `"Class 3203 - Discrete Math - is about integrals and continuity."` are invalid sentences (both have numbers, and the second has numbers and dashes). You may simply assume that sentences of the second kind will not be given to your function.

On the input `"The boy, Sam, walked to the store."`, your function should return the list: `[1, 2, 1, 2, 1, 1, 2]`

```
def sentence_counter(sentence):
    # WRITE YOUR CODE HERE
    return # list containing number of vowels for each word (list)
```

Hints:

- The functions `split()`, `strip()`, and `tolower()` are all built-in `string` functions that will be useful. A quick Google search for these will teach you what you need to know.
- To adhere to the 'sometimes y' rule, you must retrieve the *third* output of your `sometimes_y` function.
- Note that the returned lists might have varying lengths depending on how long your string is.

Part B: Fibonacci and Recursion

In this section, you will be implementing a simple recursive algorithm you've likely seen in previous classes.

Recall from previous classes that we define the nth Fibonacci number F_n as the sum of the previous two Fibonacci numbers, i.e.

$$F_n = F_{n-1} + F_{n-2}$$

where $F_0 = 0$ and $F_1 = 1$. For instance, the first 8 Fibonacci numbers (not including F_0) are:

$$1, 1, 2, 3, 5, 8, 13, 21$$

(a) Write a function `recursive_fib` that takes in an integer `n` and outputs the nth Fibonacci number (also an `int`). For this, you *must* use recursion.

```python
def recursive_fib(n):
    # WRITE YOUR CODE HERE
    return # nth Fibonacci number (int)
```

Hints:

- To do recursion using Python, simply call the same function you are implementing, *assuming* that you will get the right answer.
- Recall that a recursive solution has three main components: (1) Base Case (2) Progress *towards* the base case and (3) Recursive calls to the same function.

(b) It is a known result (consequence of the Church-Turing thesis) that any recursive function can be written iteratively. In this exercise, you will implement Fibonacci iteratively instead of recursively.

Write a function `iterative_fib` that takes in an integer `n` and outputs the nth Fibonacci number (also an `int`). This function *must not* use recursion.

```python
def iterative_fib(n):
    # WRITE YOUR CODE HERE
    return # nth Fibonacci number (int)
```

Hints:

- Recall the process of *memoization* from your intro CS classes. In memoization, you may store values in an array that correspond to your function on previous calls. This allows you to perform a recursive function iteratively by effectively "memorizing" previous results of your computation instead of keeping them on the recursion stack.
- You will still need the base case to get things going, however, which are still $F_0 = 0$ and $F_1 = 1$.