COSC 101 Homework 4: Spring 2025

Introduction

This assignment is designed to give you practice with the following new topics:

- The accumulator pattern
- Boolean expressions
- Conditionals

Note: Homeworks build on each other. You should also draw on the skills you used in prior homeworks and labs, including functions and for loops.

You are not allowed to use constructs/methods/statements, that we have not yet covered in this course–which includes while loops, break and continue statements as well as string methods.

Important Tips

- Remember, computer science is a science. Always write code with a prediction in mind. While you can sparingly code to try and see output, you should focus on thinking through what you want your code to do, what you expect as a result, and compare what your code actually does to your result.
- Before you type, trace the execution of the starter code already provided
- Sometimes sub-tasks go together, like computation followed by printing. You should consider them simultaneously if that helps you follow the correctness of your work.
- Use extra steps if needed for your thought process, printing results to test your code. Just remember to remove these extra prints once you know your code is correct.
- Match your output to the output given. Precision is important in computer science.
- Don't change file names. You're generally given `.py` files to work in. Don't change the filenames of those files.

Your assignment

Your task is to complete following steps:

1. Download the `hw4.zip` file and open it. You will see two python files, `hw4_cheers.py` and`hw4_count.py`, in the unzipped folder. You are expected to write your programs in these files.
2. Complete `hw4_cheers.py`. This file is used in Part 1.
3. Complete `hw4_count.py`. This file is used in Part 2.
4. Submit your completed programs, with each header filled in.

Grading

When we are assessing your code, higher levels of achievement are demonstrated, in part, by - Starter code left unmodified - All lines of output are present - Lines of output have proper formatting, including spaces and blank lines - Use **spaces** and **blank lines** in code for readability - Variables are appropriately named and used - Functions are appropriately named and definitions include type annotations, and `docstring`s - Functions are used to capture patterns and minimize repeated code - Functions are used to break down problems and employ abstraction - Loops are appropriately used to abstract repeated patterns

Part 1: Cheers

In `hw4_cheers.py` implement a program that prints a pep band chant based on two inputs:

1. A word that forms the basis of the chant
2. A special character that determines which parts of the chant to elongate.

The special character functions in the following way:

- For all characters other than `!`, wherever the character occurs in the word, it gets elongated. Specifically, it is repeated three times.
- If the special character is `!` then all vowels in the word are elongated. Specifically, they are repeated as many times as the position they occur in the word.

This behavior is demonstrated in the following example outputs.

Example 1

```
Give me the team to cheer for: gate
Give me a special character (or ! to make all vowels special): t

Leader: Give me a g!
Band: g!
Leader: Give me a a!
Band: ga!
Leader: Give me a t!
Band: gattt!
Leader: Give me a e!
Band: gattte!
```

Example 2

```
Give me the team to cheer for: Gate
Give me a special character (or ! to make all vowels special): a

Leader: Give me a G!
Band: G!
Leader: Give me a a!
Band: Gaaa!
Leader: Give me a t!
Band: Gaaat!
Leader: Give me a e!
Band: Gaaate!
```

Example 3

```
Give me the team to cheer for: Gatte
Give me a special character (or ! to make all vowels special): t

Leader: Give me a G!
Band: G!
Leader: Give me a a!
Band: Ga!
Leader: Give me a t!
Band: Gattt!
Leader: Give me a t!
Band: Gattttttt!
Leader: Give me a e!
Band: Gatttttte!
```

Example 4

```
Give me the team to cheer for: colgate
Give me a special character (or ! to make all vowels special): !

Leader: Give me a c!
Band: c!
Leader: Give me a o!
Band: coo!
Leader: Give me a l!
Band: cool!
Leader: Give me a g!
Band: coolg!
Leader: Give me a a!
```

```
Band: coolgaaaaa!
Leader: Give me a t!
Band: coolgaaaaat!
Leader: Give me a e!
Band: coolgaaaaateeeeeee!
```

Example 5

```
Give me the team to cheer for: goalie
Give me a special character (or ! to make all vowels special): !

Leader: Give me a g!
Band: g!
Leader: Give me a o!
Band: goo!
Leader: Give me a a!
Band: gooaaa!
Leader: Give me a l!
Band: gooaaal!
Leader: Give me a i!
Band: gooaaaliiiii!
Leader: Give me a e!
Band: gooaaaliiiiieeeeee!
```

Task

Your task is to implement this in `hw4_cheers.py`. For full credit on this part, you must ensure:

- The output matches the expected output exactly! Pay close attention to punctuation and spaces.
- You must use at least one function other than `main()` and call that function in `main().` Your `main()` function should have at least two additional lines of code apart from this function call.

Part 2: Counting

In `hw4_count.py` implement a program that asks the user for the following inputs:

- A `phrase`
- The `number` of times to repeat the program

Then, for each iteration of the program, it further prompts the user for two things:

- A starting character `start`
- A character to count `letter`

Then, it prints the number of times `letter` occurs before `start` in the phrase.

Additionally the program has some special behavior in the following scenarios: * If the `phrase` is empty * The `number of times` to repeat the program is zero

This behavior is demonstrated in the following example outputs.

Example 1

```
Enter the phrase: the quick brown fox jumps over the lazy dog
Enter the number of times you want to repeat this program: 3

Enter the character to count: o
Enter the starting character: b
The letter o occurs 4 after b

Enter the character to count: o
Enter the starting character: d
```

```
The letter o occurs 1 after d

Enter the character to count: z
Enter the starting character: y
The letter z never occurs after y
```

Example 2

```
Enter the phrase:
Enter the number of times to repeat this program: 0
You don't want to do much... See you later
```

Example 3

```
Enter the phrase: to be or not to be
Enter the number of times to repeat this program: 0
We are not gonna count much... Bye bye
```

Example 4

```
Enter the phrase:
Enter the number of times to repeat this program: 100
You entered the empty phrase "" not much to do... Goodbye
```

Building on your skills from previous homeworks, you will use an incremental development process for this task. Instead of implementing functions in the order that the program will run, you will build the core components first and then composing them together.

Task A

A core component of this program is to count the occurrences of a character. Therefore, we start from here. For this part, implement the `occurrences` function in `hw4_count.py`. You will notice that this function comes with some doctests. These tests will run automatically each time the `main` function is run, because of the line `doctest.testmod`. This can be a convenient way of ensuring that the function has the expected behavior. **Do not delete the doctests provided**. Instead, as your work on your implementation, add additional doctests and leave them in there as evidence of your practice. You will know that this part is complete when you are able to successfully pass all of the provided doctests, and the new ones that you've added.

Task B

Use the `occurrences` function that you wrote to generate the behavior described above (and illustrated in the examples).

Task C

Write a reflection in the header comments why using an incremental development process approach is effective.

Submission Instructions

Submit three Python files to the platform indicated in your class section:

- `hw4_cheers.py`
- `hw4_count.py`

Remember to complete the questions at the top of each file before submitting.