CSCA20 - Lab 5

Functions & Testing

Learning Objectives

This lab focuses on using functions to break problems down into smaller, more easily managed and tested chunks, and learning to separate algorithmic planning from detailed implementation.

Prelab

The starter code has a few functions that you will need to implement. You don't need to show up to the lab with them completed, but you should have at least sketched out the algorithm with comments so that you can spend your time in lab focusing on implementing that code.

You will also want to have added at least a few basic tests that will fail if the functions are not correctly implemented (hint: you may need to comment them out while you're developing your code, but remember to un-comment them once you think a function is done, and don't move onto the next function until your first function passes all tests)

Demonstration & Evaluation

Successfully completing this lab will demonstrate the skills of: User I/O, loops, selection, loops + selection, functions, and testing.

Your TAs will also be checking how well you've outlined your algorithm using comments, and will have the opportunity to award you with demonstration of documentation (internal).

The Scenario

The Umbilical Territorial Stem-cell Collective (UTSC) needs help with their booking system. They have someone to handle the actual database part, but what they really need is a menu that will ask the users for their information, and make sure that it's in the format that they can use in their records. Lots of donors are mis-formatting their info, and it's causing problems with the system.

Functions

The starter code has some function definitions already started for you. In order to complete this task you will need to implement (at least) 2 functions, and then use them in your code.

Matching a Template Character

In this scenario, we want to check whether a string matches a specific template. So for example, we know that phone numbers should be entered as a series of 9 numbers with brackets and a dash in the correct place, but we don't care about what the numbers actually are. So we use a template to say that `#` can represent any number, and then the template for a phone number would be `(###)###-####`. In this system, we use `#` to represent any number and `&` to represent any letter.

Matching a Template String

Once we have the function to match single characters, matching entire strings of those characters should be relatively easy. In fact, if you wanted, you could write your entire `matches_string` function without having written your `matches_character` function, but that would be a bad idea. The goal here is to think about breaking the ideas down and only thinking about one thing at a time. So having all the details of matching characters and templates dealt with in `matches_character` should mean that in `matches_string`, you only have to worry about looping over the string(s).

Menu

You'll need to build a menu that prompts the user for their phone number, donor ID, and the booking time, and then makes sure the format is correct (and informs them if it isn't).

Hints

Here are a few hints that might help you with this assignment:

- Work on one function at a time. Don't waste mental resources thinking about how you're going to use the character matching function while you're implementing it, and likewise, when you're implementing it, just trust that the function itself will work
- Test each function before you move on. It can get really complicated if you have a bug in your function that you don't realize until it's being called by another function.
- Document your functions thoroughly, and read the provided documentation. This goes back to separating algorithm from implementation: focus on the algorithm with the documentation, then you can worry about implementation separately.
- There are a few assumptions you can make that will simplify your life. For example: Since the documentation for `matches_string` says the strings must be of the same length, inside the function you can just take that for granted. Then worrying about different length strings becomes a separate (and easily solved) problem.

Extra Practice

Here are some ideas of things you can do if you're finished early or just want more to do:

- Adding extra templates is fairly easy, but what about adding extra template types? For example, what about adding a a symbol that works for only upper case letters? Or what about one that works for anything **except** spaces?
- For a real challenge, try making `#` match *any string* of numbers and `&` match *any string* of letters. So phone numbers would be `(#)#-#`, which would match (123)456-7890 (but would also match (12345)6-123456789. (As a hint for this one, make a function that doesn't return `True` or `False` if a string matches a template, but the number of characters that it matches before failing)
- If you like the fun of the templates part, you can learn more about these sort of tools: we call them **regular expressions**, and they can actually be very powerful. We will see these more later in the term, but you can play around with them now: `https://www.w3schools.com/python/python_regex.asp`