CSC108 Functions Lab

1 Tracing Function Calls

In this exercise, we will use the visualizer to understand how function calls are executed. From the Labs page of the course website, open `pizzas.py` and copy its contents. Then, open the visualizer from the Labs page and paste the `pizzas.py` code in there.

After pressing the `Visualize Execution` button, Use the button marked `Last` >> to execute the entire program. Is the output what you expected? If not, what differed?

Now, press << `First` to go back to the first line. Use `Forward` to run the program one line at a time. Before each step, think about what you expect to see; if something different occurs, pause and discuss what happened with your partner.

Switch driver and navigator.

2 Writing Functions

It's time for you to write some functions of your own! Before you begin, download all the other python starter files for week 3 from the labs page.

Open `functions.py` in IDLE. It contains three functions with steps 1-4 completed. Your job is to complete steps 5 and 6 for each one. It is good practice for the midterm questions that you will solve on paper to start by writing your solution on paper. Once you and your partner agree on a solution, type in your function bodies.

Save your file and run it. It shouldn't produce any output since none of your functions should contain any print statements. If it does produce output, go back and fix your solutions.

To test your functions, you need to call them from the Python shell. Start by calling them on the examples from the docstrings and then try some other argument values. If your functions don't work as you expect, you may want to copy them into the visualizer and trace through the execution and see where the computer's state diverges from your expectations.

1. The first function to complete is `cookies_needed`. Assume that each adult will eat 3 cookies, each child will eat 2 and each teenager will eat 5. You should not change the work that has already been completed for steps 1 through 4.

   Switch driver and navigator.

2. The second function is `is_multiple_of_3`. Try using the docstring to figure out what to do, but ask the TA if you need help. Remember to call your function from the shell to make sure it works as you expect. For now, complete this function so that it works all on its own, without any of your other functions.

   Switch driver and navigator.

3. The third function is `is_multiple`. It takes two parameters and determines if the first is a multiple of the second. Write and test this function.

4. Now that you have the function `is_multiple` available, can you see another way to write `is_multiple_of_3` by calling `is_multiple`? Go back and change your `is_multiple_of_3` so that it calls `is_multiple`. In general, it is a good idea to re-use one function inside others when you can.

   Switch driver and navigator.

Now, open `distance_converter.py` in IDLE. In this file, finish the design recipe for `miles_to_km`. Fill in the function's body and test it from the shell. For now, do not modify `km_to_miles` even though you might think it is incorrect! (We'll get there.) Note that one mile is 1.6 kilometers.

3 Using Your Own Functions in Another Program

The file `distance_converter_gui.py` that you downloaded earlier contains a very basic graphical interface (GUI) that will call your functions. Because it uses the same graphics toolkit as IDLE, you can't reliably run this new program from inside IDLE. Instead, you need to call it from the command-line by typing `python3 distance_converter_gui.py` inside a terminal window on your machine. If you are confused about what this means, please ask other nearby groups or your TA.

Once you have the converter working, try entering a few different distances and converting. Then quit.

Switch driver and navigator.

Notice that while your converter should correctly convert from miles to kilometers (using your function), whenever you try to convert from kilometers to miles, the answer is always a thousand miles no matter the number of kilometers.

Go back and look at the function `km_to_miles` that you were given in the starter code. It is incorrect! Fix it to return the correct number of miles. Save this new version and quickly test it from the IDLE shell. Go back to the terminal window and start the graphical converter again. Try converting 42.2 km now. It should convert correctly to 26.375 miles. That's because the converter is calling **your function!**

4 Writing a Calculator for Course Grades

This next task involves writing functions that will be used by a calculator to work specifically with CSC108 course grades. Students often want to know what mark they will need on the final exam to get a certain mark in the course. We are going to create a tool just for those students!

- Start by opening `mark_functions.py` in IDLE. There are seven small functions that you need to complete. In each case we have completed some parts of the design recipe but you need to do more.
- Complete the functions in the file in order. Each function initially has only one example. In order to make sure you understand what the function is supposed to do, add another example to each docstring. Make sure you and your partner agree on the correct output for your example. If you can't agree, reread the description and if you still don't agree, ask the TA for help.
- After adding another example to the docstring, complete any other missing design recipe steps. None of the function bodies are more than a few lines and some of them should call other functions in the file. Test each function as you write it. Now that the functions are a bit harder, it can get a little frustrating having to type and retype your test cases into the shell. To help you, we have put the test cases into the file `test_mark_functions.py` that you have already downloaded. You should open it in IDLE now (and keep `mark_functions.py` open, too!). Most of the tests in this file are commented out. As you finish each function, you can uncomment the tests for that function and also add your own tests.
- **Switch driver and navigator after each function.**

Finally, when all the tests in `test_mark_functions.py` have been uncommented and pass, you are ready to try another GUI interface that makes use of the functions you have written. Go to a terminal window and type `python3 mark_calculator_gui.py`. Try it out by making up some marks for now. Soon enough you'll have real grades to enter.

Once you're finished, call your TA to demonstrate your work and to receive credit for the lab. See you next week!