COM110: Lab 1

functions, input, output, and loops

When we create a computer program, it is often for someone else to use. That person doesn’t necessarily see or read the code, but they run the program, and can interact with it and see what the program outputs. (For example, this is true of any smart phone app, or website, or software you run on your computer, etc.). So when we use the word “user,” we are referring to the person running and using our program. We are the programmers, and “they” are the “users.” When we are testing our own programs, however, we have to pretend to be the unwitting user, putting ourselves in their shoes, experiencing it as though we have no idea what the code says, how it works, or what the purpose of the program is.

[The first five checks below should take about 10 minutes each. Feel free to call us over if you get stuck or confused!]

1) **Input, output, and defining functions.**
   a. Start IDLE and try entering some commands in the interactive shell mode.
   b. Create a module by choosing **File->New File.** Begin your program with some comments that include your name and date and the purpose of the program.
   c. We will start by trying out the **input() function:**
      ```python
      <variable name> = input(prompt)
      ```
      It does the following:
      i. Prints out the text string **prompt** for the user to read, then “freezes” the program, waiting. (Usually **prompt** is a sentence we have composed that asks the user to type something, but it is optional and may be omitted from the function call.)
      ii. The moment it detects that the user has hit the Enter key, the function takes anything the user has typed, and stores it **as a string** into the variable on the left hand side of the assignment statement.
d. Using the `input()` command, prompt the user to enter his/her name and store it into a variable. E.g.,

```python
firstName = input("Please enter your name: ")
```

You will need to save this program (in a place where you will keep all lab work) and give it a name with a .py extension (for example, hello.py). Try running the program by using the `Run->Run Module` command (or hitting F5). So far, the program prompts the user for their name, stores it into the variable `firstName` when they hit enter, but does not do anything with it, so there is input from the user, but no output.

Get check 1

e. Now, print out a greeting using the name entered by the user. For example, if the user enters the name Sam, you could output something like, “Greetings, Sam! Welcome to my computer program.” (Or something more creative/humorous.)

To output more than one expression on a single line we can use the `print` function as follows:

```python
print(<expr>, <expr>, <expr>, ...)  
```

where `<expr>` can be an expression of any kind, like a string literal (anything in quotes), or a variable name, like `firstName`.

Get check 2

f. Since a program is often broken down and organized into more than one function, let’s put the code we’ve written so far into a function and then call the function. At the beginning of your program (after your comments), type

```python
def greeting():
```

and then put the rest of the code, indented and aligned under the `def greeting():` (notice the indentation that IDLE will put in for you).

g. If you now run the program from the program window, nothing will happen. Do you see why?

h. To call or invoke the function, add one more line to the end of the program: simply add an un-indented call to `greeting()`. Now, when you run the program, `greeting()` will
first be defined and then it will be called/invoked, causing the greet() function to execute.

○ Get check 3 ○

i. Now define a second function, called farewell(). You can define this below your greet() function definition and above the line that invokes greet(). Using the input() command, ask the user when they next plan to run this program again. Print out a goodbye message that includes a mention of seeing the user again when s/he specified. Don’t forget to leave a blank line after the farewell() function before the call to greet().

j. If you save and run your code, your program will execute the greet() function, but not the farewell() function. Do you see why?

k. Fix your code so that the farewell() function also executes.

○ Get check 4 ○

l. Finally, programs usually have a “main” function, which is like the “driver” or “manager” of the program: the main() function gets everything going, usually making calls to the other functions you’ve written as needed.

m. Put the calls to (i.e., the invocations of) greet() and farewell() into a main function (def main():). If you save and run your program, it should appear to do nothing. What final line of code must you add in order to get the program to actually greet the user and say farewell?

n. Notice that after you run your program once, it means it has already been “loaded up.” You can now call its functions individually from the interactive prompt. Try typing just goodbye(), farewell(), or main() at the prompt after your program has already been loaded.

○ Get check 5 ○

2) [~20 minutes] The range() function. We’ve already seen in class that the function call range(n) generates the sequence of integers [0, 1, 2, ..., n-1]. In our reading this week we learned that the range() function is actually more versatile than this!
Look at each of the `range()` function calls from Discussion Exercise 3 on page 74 and see if you can predict the outcome. Test them in the Python interactive shell. If you still can’t figure it out how `range()` works, keep playing around with calls to `range()` of your own, or refer to the reading starting on the bottom of page 64 to the middle of page 65. For this checkpoint you must show us the tests you ran and explain to us how the `range()` function works.

- Get check 6

3) [~20 minutes] For loops with the range() function. Write a program that prints out...
   a. ... the natural (counting) numbers from 1 to 10.
   b. ... then a count–down of the natural numbers starting from 10 down to 1.
   c. ... then the natural numbers from 1 to a user–specified value
   d. ... then a count–down of the natural numbers starting from a user specified value

- Get check 7

4) [~20 minutes] Pizza: small or large? At the Two Wives Pizza restaurant in downtown New London, you can get any of their wide variety of yummy pizzas in two sizes: 8 inches and 14 inches. For example, (last I checked the menu) their Four Cheese Pizza is $7 for the 8 inch size, and $11.50 for the 14 inch size. At first glance, you might think to yourself, “Okay, fine, $11.50 for 14 inches sounds like a slightly better deal than $7 for 8 inches.” But remember, these inches refer to the pizza’s diameter (length across the middle), while the area of the pizza in square inches is actually equal to \( \pi r^2 \), where \( r \) is the pizza’s radius (half of the diameter). So for each extra inch of radius, the area (in square inches) of the pizza grows by a lot more than one!

Write a program to help Two Wives customers discover the value per square inch of each pizza. Have the program ask the user how much they were asked to pay, i.e. have them input the price of each pizza size. Remember the sizes are 8 inch diameter
for the small and 14 inch diameter for the large. With the user-inputted prices, your program can then calculate the *price per square inch* of the pizza they paid for.

To complete this task, you’ll need to import the math module so that you can use the built-in Python `pi` function. That is, at the very top of your module, type the command

```python
import math
```

Thenceforth, the code `math.pi` can be used to represent the (approximate) value for $\pi$.

Get check 8

5) **Extra time? Extra Credit Bonuses. (May be completed separately in any order.)**

a. Write a program called `progFun.py` that repeats the sentence “Programming is fun!” and adds a line number in front of each repeated sentence, counting them out, e.g.:

1. Programming is fun!
2. Programming is fun!
3. Programming is fun!
4. Programming is fun!
...

Put your code inside a main function that is then called at the bottom of the program. Add code to allow the user to input the number of times the sentence is printed. Test your program with inputs of 0, 10, 100, and 1000.

b. If you completed the above by using two arguments in your call to the `range()` function, see if you can modify it so that it has the same output, but there is only one argument in your call to the `range()` function. If you did it with only one argument to the `range()` function, now do it with two.

c. Write a program that outputs all the powers of 2 up to (and including) 20. The output should be user-friendly and readable/informative (for example, for each power of 2 it might
say: $2^{10} = 1024)$. Make your output start from $2^1 = 2$. Don’t forget comments and the use of a main function.

*Get checks for bonuses*