COM110: Lab 1 - Aug 29 & 30

Python, functions, input, output, and loops

During COM110 labs you should always feel free at any time to consult with other students and lab TAs. Lab can and should be a team-oriented conversational activity. While you should *never* blindly copy someone else's code, discussing lab questions with each other is strongly *encouraged*.

Install Python (if needed). If you want to use your own computer/laptop, download and install Python 3.7.4. We will do most of our coding using IDLE, which will be installed when you install Python. If you don't plan to use your own laptop, you should use a lab machine (which already has Python installed). Let a TA or the instructor know if you have any issues. After Python is installed, launch IDLE (the development environment that comes with Python by default).

Experimenting with the Python Shell. Note that the below commands are sample suggestions and after typing them in you should feel free to experiment with your own versions of these expressions! The more you experiment and test things out the more you learn. Tinkering and experimentation is always highly encouraged! But do not copy and paste the code. This often won't work well.

- To start Python open IDLE (Python GUI)
- When you see the >>> prompt
  - try typing `8+56` (then hit the Enter key)
  - then try `6.5674*54`, `1-(54.7/15)`
- Now try: `print(10+65)`
- And then: `print("10+65")`

  Q1: What's happening here?

  The more you experiment the more you learn.
- `print(8/3)`
- `print(8//3)`

  Q2: How does python interpret a single slash vs a double slash?

- Some tips - try the following keyboard shortcuts on your Python Shell window:
  - On a Windows machine: try **Alt-P** and **Alt-N** to see what they do;
  - On a Mac: try **Control-P** and **Control-N**
  - Also try putting your cursor at the end of any previous line and hit **<Enter>**

Q3: What do these keyboard shortcuts do?

- Let's write a set of instructions and group them into what is called a function :

```
>>> def program1():
    print("This is my first program:")
    print(2**3)
    print("How exciting!")
```

- Follow the last line with a blank line.
- Notice the word "`def`" which is special in Python and also notice the indentation – it's important – as are the parentheses and the colon.
- Now try typing and executing the command `program1()` at the Python prompt to see what happens. Then try it again.
  Q4: What seems to be happening when you execute the command `program1()`?
  Q5: What does the ** seem to mean? (You are encouraged to do your own testing to find out!)

Discuss your answers to the above five questions with an instructor or TA.

☺ **Get check 1** ☺

Creating a Python Module.

- Create a folder (on your computer, eg. inside the Documents folder) called ***lab1_xxxx***, where xxxx is your Conn username, for example **lab1_wtarimo** for me. You will save all your Lab1 modules in this folder. **At the end of lab you will compress/zip the folder into lab1_xxx.zip and submit it to the Lab1-Submission link in Moodle.** (Ask if you need help with any of this, now or later**)**
- If we close the Shell window we will lose all the stuff we have typed – so usually we won't use the interactive Shell unless we're just testing commands or tinkering. Instead, we'll create a module or script, which is just a text file that can be saved, run/loaded, and modified at will.
- Modules in Python can be created as text files with any editor but we will create them using IDLE:
  - Under **File** select **New File**
  - Save your file as eagergreeter.py.
  - Then type in (copy and pasting will not work well)

```
#<Your name here>
#<Date here>
#COM110 Lab 1
#eagergreeter.py    (this just indicates our file/module name)
#
#This program will print out "Hi!" over and over

def main():
    for i in range(10):
        print("Hi!")

main()
```

- Save your changes
- Under the Run menu (in IDLE) choose Run Module or simply press F5 on your keyboard
- Notice strings must be in quotes or you get an error message. We will get back to this program later in the lab.

☺ Get check 2 ☺

1) **Input, output, and defining functions.**
   a. Create a new module by choosing **File->New File**. You will need to save this program in the lab1_xxx folder and give it a name with a .py extension (for example, hello.py). As you did in eagergreeter.py, begin your program with some comments that include your name and date and the purpose of the program: for now just use: `#This program prints out greetings and farewell messages to the user.`
   b. By now we've seen/used the `input()` function a few times:

      `<variable> = input(s)`

      It does the following:
      i. Prints out the text string `s` for the user to read, then "freezes" the program, waiting. (Usually `s` is the "prompt" 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 "sees" 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.
   c. Using the `input()` command, prompt the user to enter his/her name and store it into a variable.
   d. Print out a greeting using the name entered by the user. Try running the program by using the **Run->Run Module** command (or hitting F5).

☺ Get check 3 ☺

e. 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

```
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).

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

g. To call 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 your commands to execute.

☺ Get check 4 ☺

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

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

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

☺ Get check 5 ☺

k. Finally, programs usually have a "main" function, which is like the "driver" of the program: the `main()` function gets everything going, usually making calls to the other functions you've written as needed.

l. Put the *calls* to (i.e., the invocations of) `greeting()` 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?

m. 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 6 ☺

2) **Loops.** Recall that the start of a **for** loop has the format

```
for <variable> in <sequence>:
```

a. Go back and open eagergreeter.py. Modify the program to print "Programming is fun!" 20 times, instead of printing "Hi!" 10 times.

☺ Get check 7 ☺

b. Now modify the program so that it first asks for the user's name, and then prints out a greeting to the user, addressing them by name, 20 times.

☺ Get check 8 ☺

c. Now make the program also ask the user for the number of times to wish them Happy Birthday and then have the program print out that many "Happy Birthday"s. (Recall from 1c above that the `input()` function returns the user input as a *string*. To turn the input *string* into a *numerical value* that we can then operate on, we need the `eval()` function that we've been seeing in class/reading.)

☺ Get Bonus check 1 ☺

d. Modify the program again so that it adds a line number in front of each birthday wish, counting out the number of wishes, e.g.:

```
1.  Happy Birthday, Sue!
2.  Happy Birthday, Sue!
```

```
3.  Happy Birthday, Sue!
4.  Happy Birthday, Sue!
```

…

e. Put your code inside a main function that is then called at the bottom of the program.

☺ **Get Bonus check 2** ☺

3) In a new module file, 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 Bonus check 3** ☺