COMP 1405B
Fall 2019 – Practice Problems #2

Objectives

- Practice creating and running Python programs
- Practice performing basic input and output operations
- Practice using variables and performing calculations
- Practice performing type conversion on variables and data
- Practice applying problem solving and computational thinking skills to programming problems

---

The following problems deal with introductory Python programming. Throughout the course, I will assign a difficulty rating to the posted problems. If you can understand and solve all problems of difficulty 3 or less by the time you write a midterm/exam covering that material, then you should be able to get through the course. If you can understand and solve all problems of difficulty 4 or less, you should do well in the course. If you can understand and solve all problems, then you should do very well. Remember to test your solutions to verify their correctness and discuss your solutions with both other students and TAs in the course. Verifying your solutions with TAs will be a good way of ensuring that you are on a path to success in the course.

Problem 1 (Hello World - *)

A similar process can be used to solve many programming problems. The steps below outline a process that you can apply throughout this course when working to solve programming problems:

1. Analyze and understand the problem
    a. What is the problem asking?
    b. What information is required to solve the problem?
2. Create a plan to solve the problem
3. Design an algorithm
4. Test the algorithm by tracing through execution on paper
5. Write the Python program code in a text editor
6. Execute and test your Python code/program from a command prompt. Possibly repeat any of the previous steps until the program works correctly

The first program you will complete is a simple program that prints a phrase ("Hello World!") to the console, which is the window in which the user interacts with the program. This program is simple enough that we can largely skip steps 1-4 of the above process. The algorithm for this program is one step: print "Hello World!" to the console.

The next step of the process involves writing the necessary code inside a text editor and saving it as a file on the computer. Open a text editor of your choice (Notepad++ is good on Windows, Atom is good on Windows/Mac). You may also use an IDE if you wish, though it will have to have auto-complete and other additional features turned off for the midterms and exam.

Type the following in the text editor:

```
#my first python program
print("Hello World!")
```

The first line of this code starts with #, which is used to tell Python that you are writing a comment. This means that everything to the right of the # is ignored by the Python interpreter. Comments can be used to explain what a block of code is doing, make note of your thought process, or include any other information that may be useful in understanding the code.

The second line contains a print command. This command tells the Python interpreter to print whatever value is inside the brackets, which in this case is "Hello World!".

Now that you have written some code in the text editor, you need to save it in a .py file before you can execute it and see the result. If you are using a lab computer, save the file as "helloworld.py" in your Z-drive. When you click save-as, you will find your Z-drive as one of the places to save your code. Select the Z-drive and specify the name of the file as "helloworld.py". If you are using your own computer, you can save the file wherever you prefer. A common error when saving these files is to end up with multiple file extensions (e.g., helloworld.py.py, helloworld.py.txt, etc.). Show your completed file to a TA if you want to confirm you are doing it correctly.

To execute Python programs in lecture, we will use the Windows Powershell command prompt program. If you are using a Mac computer, the equivalent program is called Terminal. Additionally, within Powershell on Windows, we will use the command 'python *filename.py*' to run a file. On Mac, this command is generally 'python3 *filename.py*'. Open the Windows Powershell program or a Terminal, type **ls** (that is, LS in lower case letters) and hit Enter to see which directory (folder) you are currently in and what files are in that directory. The output of the command should start with something like

```
    Directory: C:\Users\yourname
```
OR
```
    Directory: Z:\
```

You need to change the directory to be the same as the one where you saved your helloworld.py file. To change directories within Powershell, you use the **`cd`** command. Typing **`cd Z:`** and hitting enter will change the directory to the Z drive. Use the ls command to confirm you have changed to the Z drive and see what files are present. You should see your helloworld.py file listed – if you do not, make sure you saved it to the directory you are currently in.

Once you are in the same directory as your saved file, type **`python helloworld.py`** and hit Enter to run your program in the Python interpreter (Mac users should type **`python3 helloworld.py`**). Doing this runs a separate program, called the Python interpreter, and tells it to execute your Python file helloworld.py. The Python interpreter will then read and execute each of the instructions you have included inside the specified file. If you see “Hello World!” printed in console, your program works as expected! That’s all there is to it! You have now created and run a Python program.

Problem 2 (Basic Input - *)

In the previous problem, you used the print command to print out text. You can perform ‘user input’ by reading what the user types into the console. The Python command for reading user input from the console is:

```
    input(“Prompt you want the user to see: ”)
```

Whenever you include this command inside your code, the Python interpreter executing the code will print out the prompt included inside the brackets (in this case, ‘Prompt you want the user to see: ’) and then wait for the user to type something in and hit Enter. Whatever text the user types into the console before hitting enter is returned by this command, a concept we will discuss later. For now, we will just say that the input(…) command evaluates to the text that the user typed in. You can save what the user typed so it can be used later in the program like this:

```
    text = input(“Type some things: ”)
```

This takes the text the user typed and stores it in a variable called *text*. You can think of this as associating the name *text* with whatever the user typed in. So, any time later in your code where you use the name *text*, you will retrieve the value associated with the name *text*. To see how this works, create a new code file called basicinput.py and add the following two lines of code (note: there are no quotation marks around name in the second line):

```
    name = input(“Enter your name: ”)
    print(name)
```

Save the file and execute it from the command line the same way you did for the previous problem. The program should ask you to enter your name and then print your name once you have hit Enter.

Problem 3 (Basic Input/Output and Calculation - *)

Write a simple program that prompts a user for their name and birth year. The program will then display (output) a greeting to the user that includes their name and how old the person is. For example, when the program is executed, the output may look like the following (highlighting is used to signify user input):

```
Enter your name: Kitty
What year were you born in Kitty? 2013
Hello, Kitty. You are now 6 years old.
```

Note: The input() functions returns the input as a string. In order to convert the string into an integer use the **`int()`** function. For example, if `x = "12"`, then `int(x)` is the integer 12. Also recall that the **`str()`** function will convert a number to a string.

Problem 4 (Conversion Program - *)

Write a conversion program to convert some number of one unit into another. The unit of choice can be a distance (e.g., kilometers to miles, meters to feet, etc.), a temperature (Celsius to Fahrenheit, etc.), currency (USD to CAD, etc.) or anything else you can think of.

Problem 5 (Calculating Volume - *)

Write a program that will ask the user to enter the three dimensions of a cube: length, width, and height. The program should then calculate and output the volume of that cube to the user. For a cube, volume is equal to `length * width * height`. Try looking up the volume formula for other shapes and writing additional programs. For some, you may need to perform more advanced calculations. You can look at the documentation of Python's built-in math module online for details on some of the more advanced math operations you can easily perform, such as logarithms, square roots, trigonometric functions, and angular conversions.

Problem 6 (Grade Calculation - *)

Write a grade calculator program that will compute and output the final grade of a student based on four scores entered by the user: three midterm grades and a final exam grade. To start, assume the weighting will be 20% for each midterm and 40% for

the final. If you are unsure of the math involved for this type of problem, consider looking up how to calculate a ‘weighted average’. If you have completed this problem successfully, consider adding the ability to have the user specify the weighting to use for each item.

Once you have completed the problem above, consider adding a bonus assignment. Ask the user to enter a grade out of 10 for the bonus assignment. If the student’s grade is X/10, the new weighting should be 20% for each midterm, (40-X)% for the final exam, and a bonus X percent from the assignment. That is, the student receives X% of the mark guaranteed and the weighting of the exam is reduced by X%.

Problem 7 (Simple Guessing Game - *)

Write a program that generates a random number between 1 and 100 (inclusive) and prompts the user to guess the number. The user only gets one guess. Your program should respond to the user’s guess by telling them how far from the number they were. Your program should not include a negative sign in this number (i.e., you need to calculate an absolute value). For this question, you will need to generate a random number and calculate an absolute value. You should look through the documentation for Python’s math and random modules to figure out how to do this.

Problem 8 (Simple Graphics Drawing - *)

Create a drawing using the SimpleGraphics.py module posted on cuLearn. If you are unsure how to get started, see the lecture on introductory Python in which we use SimpleGraphics for a house drawing. You will not be tested directly on SimpleGraphics, so you do not need to worry about remembering all the commands. It is just a simple tool to practice some basic programming.