ENGR 13300 Fall 2021- HW 5 PY 1
Python 1: Introduction to Python

Individual Tasks

Guidelines for Tasks 6–7:

1. These two tasks are individual assignments. You may seek help from classmates, the instructional team or others but the work you submit should be your own. If you collaborate with others and use information developed together or by someone else, ALWAYS document and reference that material.
2. Each individual is responsible for submitting their own individual assignment to Gradescope.

Task 6 (of 7) [Individual]

Learning Objectives: Perform arithmetic operations in Python (i.e., addition, subtraction, multiplication, division, and exponentiation), while keeping in mind order of operations; Create valid identifiers, accounting for relevant Python rules (i.e. keywords) and code standard; Implement the use of expressions in Python to assign values to variables; Recognize and explain the differences between various data types in Python; Convert between data types (e.g. strings to integers, or floats to integers) in Python based on program needs; Identify which resources are available to you to aid in learning Python; Import modules in Python (i.e. math module); Employ the Spyder IDE to write, edit, and save Python code; Output data from a script to the screen in Python.

Task:

As you further explore programming, you will find that there are several concepts and features hidden within each language that lend greatly to the usability of the language in a niche field or when looking to solve a specific problem in a specific way.

Python contains a number of hidden elements within its built-in modules that apply to unique situations or tasks. In this activity, you will be asked to delve deeper into the world of Python modules. In order to learn about and apply these modules, you will need to do some research on your own. Being able to use your resources to help you find answers to difficult problems will be an important skill for you to master, not only for this class, but also going forward as an engineer.

Suggested modules to research: `random`, `fractions`

Suggested built-in functions to research: `round()`, `int()`, `float()`, `input()`, `limit_denominator()`

Hint: When you want to find out what a function does, type `help(function_name)`, e.g. `help(int)` or `help(round)` into the Python Console (or search the internet).

Suggested: Investigate the difference between the following and how it affects the calling of functions:

```
from (module) import (function)
import module
```

You are spending your next summer consulting at a start-up company that creates on-line educational tools to assist elementary school students in practicing math. You are working on a tool that demonstrates to students the equivalence between doing calculations with decimal numbers and doing the same calculations with fractions. The tool should randomly generate four numbers between zero and 100. Add the numbers together, and then convert these numbers to fractions and repeat the addition using the fractions. The random numbers should be printed with three decimal's precision to the screen. The fractions should be printed to the screen with a denominator smaller or equal to 100. An example output is shown below.

Write a Python script (filename: `Py1_Task6_username.py`) that will follow the procedure outlined above. Utilizing `random.seed(int(input("Enter the seed: ")))` before generating the random numbers initially is required in your program for it to be graded properly. When generating a random number in Python, a pseudo-random number generator is used which generates random numbers based on a seed. If a seed is set, the sequence of random numbers it generates is determined and the sequence of random numbers will be the same again if you set the seed to the same value.

Example Output:

```
Enter the seed: 133
First Random Number : 49.383
Second Random Number : 48.235
Third Random Number : 76.895
Fourth Random Number : 80.829
Sum from decimals: 49.383 + 48.235 + 76.895 + 80.829 = 255.342
Sum from fractions: 2321/47 + 820/17 + 1461/19 + 6143/76 = 9703/38
```

Before you start programming in Python, create a flow chart representing the algorithm and save it in a PDF file that will be named `Py1_Ind_username.pdf`.

Note that your program will be graded based upon the output and the process that you used to reach this output. Your program should not produce any errors when running or points will be deducted. Do not use the `input()` function except to set the seed for the pseudo-random number generator.

Task 6 Files:

1) `Py1_Task6_username.py`
2) `Py1_Ind_username.pdf`

Task 7 (of 7) [Individual]

This task will be autograded by Gradescope: You can submit the file and see how it is evaluated and make as many changes as you need to maximize your points.

Learning Objectives: Create and execute simple scripts comprised of basic Python concepts; Output data from a script to the screen in Python; Apply course code standard in development of Python scripts; Modularize and comment code in Python for readability and reusability.

Task:
You may recall from Physics class that a capacitor is an electrical component comprised of an insulator sandwiched between conductive plates. Capacitors store energy in an electric field when the plates hold a charge. There are two types of capacitive networks: series and parallel. For each of the network types one can calculate an equivalent capacitance for the entire network but the two types are calculated differently. The following equations for capacitors in series and parallel:

$$C_{equiv} = C_1 + C_2 + \ldots + C_n \qquad \text{(Parallel Capacitance)}$$

$$\frac{1}{C_{equiv}} = \frac{1}{C_1} + \frac{1}{C_2} + \ldots + \frac{1}{C_n} \qquad \text{(Series Capacitance)}$$

For this assignment, you will need to write a Python script (filename: `Py1_Task7_username.py`) to calculate the equivalent capacitance of both the series and parallel configurations of two capacitors. The program should output the results for both series AND parallel calculations. The output should be formatted in a table modeled similar to the one seen below, with all values rounded to the nearest tenth. You have been given a test case by your supervisor to illustrate the computation which you can use to test if your program outputs correct values.

The following table is using test cases only and should not be included in your program.

Type of Capacitance	First Capacitance	Second Capacitance	Equivalent Capacitance
Parallel	10.0 µF	20.0 µF	30.0 µF
Series	10.0 µF	20.0 µF	6.6 µF

Next, you must demonstrate the correctness of your program by applying it to the test case where the first capacitance is an input from the user and the second capacitance is $\sqrt{2.72}e$ µF. (Hint: When finding the value of $\sqrt{2.72}e$, be as exact as possible.). Do not use the `input()` function except to get the first capacitance from the user. To better display your logic, create a flowchart and save it in your previously created PDF file (`Py1_Ind_username.pdf`)

Example: Assuming the following values: $C_1 = 6$ µF and $C_2 = \sqrt{2.72}e$ µF
The outputs of your Python program should be formatted like the following (inputs in bold):

```
Input the first capacitor value [µH]: 6
Type             First         Second         Equivalent Capacitance
Parallel          6.0 µH        4.5 µH         10.5 µH
Series            6.0 µH        4.5 µH          2.6 µH
```

Hints: Use f-strings to format your output. Entering `\u00b5` in a string will produce the µ symbol.

Task 7 Files:

1) Py1_Task7_*username*.py
2) Py1_Ind_*username*.pdf

List of all files to submit for Ind HW5-PY1:

1. Py1_Ind_*username*.pdf
2. Py1_Task6_*username*.py
3. Py1_Task7_*username*.py