EECS 1015 Lab 2

Goal

- Be able to write a script with basic building blocks

Tasks

1. Guide you through the process of writing a script with basic building blocks and get familiar with submission syntax
2. Learn to debug a script with basic building blocks
3. Learn to write a script with basic building blocks
4. Learn to write a script with basic building blocks
5. Learn to write a script with basic building blocks
6. Learn to write a script with basic building blocks

Total credit: 100 pts

You are very welcome to ask clarification questions to TAs. But please read the document carefully before you ask questions.

It is an academic offense to copy code from other students, provide code to other students, let other people (including the teaching staff) debug your code, or debug other students' code.

We will make check code similarities at the end of the semester.

Questions 3 – 6 may not be arranged by the difficulty level. You are highly recommended to take a look and decide the order of completing the questions. You will experience something similar and it is important to learn how to triage your tasks.

Task 1: Follow the Steps (30 pts)

In this task, you will be provided with questions to guide you through the process of writing a script. Although answers are provided, please think about the solution and try it yourself, then compare it with the solution to best learning result.

If you want to do well in the course, please do follow the steps, rather than taking a quick look at them. Simply taking a quick look at the steps is very different from working on those yourself. Please do work out each step yourself for better learning. It is a very important skill to learn how to break down a problem, and we start with easy questions so that you are not overwhelmed.

After completing this task, you can put the materials away, and see whether you can implement the script without the step-by-step instructions, and compare your script with the solution.

You want to buy some fruit in a grocery store and want to know the total price. Here is the price of the fruits available in the store:

- Peaches: $ 3.99 / each
- Apples: $ 2.94 / each
- Watermelons: $ 7.99 / each

You are highly recommended to attempt the questions yourself before you look at the solution as a way to learn.

Q1.1 In the interactive mode of your IDE, try to calculate the price if you want to purchase 2 apples.

Solution:

```
>>> 2.94 * 2
5.88
```

Q1.2 Same as above, but in addition, use the variable `num_apples` to represent the quantity of apples.

Solution:

```
>>> num_apples = 2
>>> 2.94 * num_apples
5.88
```

Q1.3 Same as above, but in addition, use the variable `price_apples` to represent apple price.

Solution:

```
>>> num_apples = 2
>>> price_apples = 2.94
>>> price_apples * num_apples
5.88
```

Q1.4 Same as above, but in addition, assign the resulting price to the variable `total_price`.

Solution:

```
>>> num_apples = 2
>>> price_apples = 2.94
>>> total_price = price_apples * num_apples
>>> total_price
    5.88
```

Q1.5 What if you want to calculate the price of buying 3 apples?

Solution:

```
>>> num_apples = 3
>>> price_apples = 2.94
>>> total_price = price_apples * num_apples
>>> total_price
    8.82
```

Note that you need to retype everything in the interactive mode, which is not convenient. Now let's try to use the script mode. Create a new Python file by clicking on the icon:

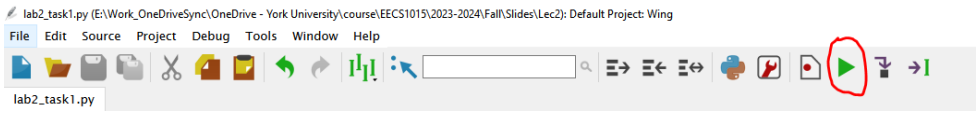
Type the following code in the script mode (i.e., in the Python file):

```
num_apples = 2
price_apples = 2.94
total_price = price_apples * num_apples
print(total_price)
```

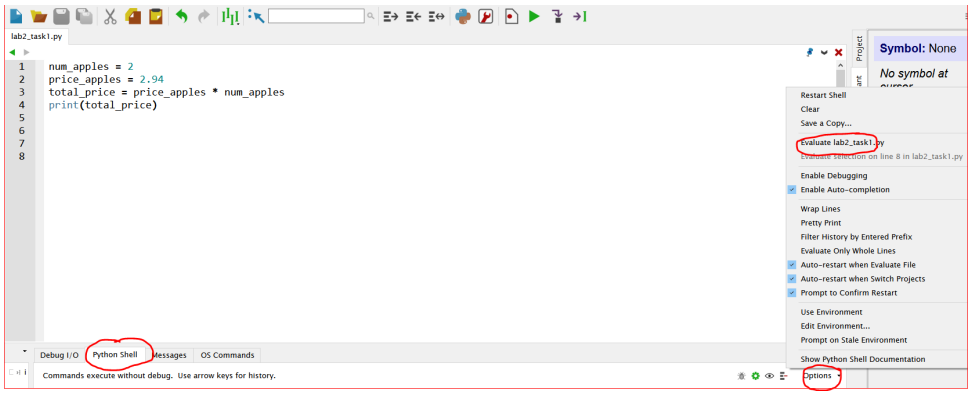
The first three lines are the same as what we typed in the interactive mode. In the fourth line, instead of `total_price`, you need to use the `print` function. In the interactive mode, when you type a variable and hit enter, it will show you the value of the variable. However, the script mode does not work that way. You need to tell the script mode explicitly that you want to see the value of the variable. You do that by using the `print` function.

Save the file (I name the script lab2_task1.py . '.py' is the python script type), and run the script. You can use one of the two ways to run the script:

- Method 1: Click the green triangle, then click ok in the pop-up window:



- Method 2: Or click Python Shell -> Options -> Evaluate lab2_task1.py (yours may look different if you name the file something else):



There are minor differences between the two methods, though doesn't matter for our purposes. Here are the differences in case you are curious:

- Method 1: it just runs the script and then exit. You will not be able to access the variables.
- Method 2: it runs the script but does not exit. You can access the variables in the interactive mode. For our particular example, you can type `num_apples` in the interactive mode, and it will show you the value.

Now let's go back to our example. With this script, it will be easier to get the price as you just need to change the value assigned to `num_apples`, and then rerun the script, without the need to retype all the codes. Furthermore, you can reopen the file after you close the IDE and keep working on the script. This is impossible in the interactive mode as once you close the IDE, you will lose all the code and variables in the memory. Therefore, you will be required to write code in the script mode for all the labs. However, if you just need to test something quickly, you are very welcome to use the interactive mode as you do not need to create a file to test it. Let's move on.

Q1.6 What if you want to buy 2 apples ($ 2.94 / each) and 4 peaches ($ 3.99 / each). (Hint: make sure you come up with meaningful variable names that follow the PEP-8 convention)

Solution:

```
num_apples = 2
num_peaches = 4
price_apples = 2.94
price_peaches = 3.99
total_price = price_apples * num_apples + price_peaches * num_peaches
print(total_price)
```

This chunk of code looks a bit messy. Usually, we group code for similar functions together and separate the groups with a blank line, for example:

```
num_apples = 2
num_peaches = 4

price_apples = 2.94
price_peaches = 3.99

total_price = price_apples * num_apples + price_peaches * num_peaches

print(total_price)
```

To make the code even more readable, we leave a comment to explain what the group of code does, for example:

```
# specify fruit quantity
num_apples = 2
num_peaches = 4

# specify fruit unit price
price_apples = 2.94
price_peaches = 3.99

# calculate the total price
total_price = price_apples * num_apples + price_peaches * num_peaches

# show the total price
print(total_price)
```

You'll notice the comments start with #. The interpreter will ignore the line as this symbol means that this line is for humans to read. You can even comment your code in the same line as your code, such as:

```
# specify fruit quantity
num_apples = 2 # apple quantity
num_peaches = 4 # peach quantity

# specify fruit unit price
price_apples = 2.94
price_peaches = 3.99

# calculate the total price
total_price = price_apples * num_apples + price_peaches * num_peaches

# show the total price
print(total_price)
```

This is unnecessary for our example as it should be obvious from the variable name. However, you are highly recommended to comment your code and form a good coding style.

Q1.7 Now in addition to the previous question, modify your code so that it includes the watermelon (suppose you want to buy 1) ($ 7.99 / each). (Hint: make sure you come up with meaningful variable names that follow the PEP-8 convention)

Solution:

```
# specify fruit quantity
num_apples = 2
num_peaches = 4
num_watermelon = 1

# specify fruit unit price
price_apples = 2.94
price_peaches = 3.99
price_watermelon = 7.99

# calculate the total price
total_price = price_apples * num_apples + price_peaches * num_peaches + price_watermelon * num_watermelon

# show the total price
print(total_price)
```

Now you have a simple script! 😊

Let's continue to improve the script. Usually, we want the script to be useful not just for yourself, but also used by other users. Currently, the fruit quantity is defined in the script. Imagine there is a user who wants to use the script, the current implementation requires the user to find where the variables are defined in the script. This may not be an easy task if the script is very long, or if the user has no prior programming experience. Recall the laptops and cell phones that you use every day, you do not need to read the code to be able to use those electronic devices. Instead, you can manipulate your device through a graphic user interface (GUI). The device takes your input and executes the code behind the scene. Similarly, in our example, let's do something similar. We will not implement a graphic user interface, though. It is too complicated at your current stage. Instead, let's just have a plain interface.

Recall the `input` function that you are required to experiment with as one of the puzzles after class. This function takes input from the user. Let's play around with this function. In the interaction mode, try this line of code `input("Question: ")` and see what happens:

```
>>> input("Question: ")
Question:
```

It prompts the user to provide some input. The string `"Question: "` is displayed for the users to tell them what information to provide. Here we can just type something randomly, like `abc`, click enter, then you will see:

```
>>> input("Question: ")
Question: abc
'abc'
```

It takes in the input from the user as a string. Now let's try to assign the input value to a variable:

```
>>> a = input("Question: ")
Question: abc
>>> a
'abc'
```

Q1.8 Now use the `input` function to modify your script so that it takes information from the user about how many apples that the users would like to purchase.

Solution:

(This is not the final solution yet, read along)

```
# specify fruit quantity
num_apples = input("How many apples? ")
num_peaches = 4
num_watermelon = 1

# specify fruit unit price
price_apples = 2.94
price_peaches = 3.99
price_watermelon = 7.99

# calculate the total price
total_price = price_apples * num_apples + price_peaches * num_peaches + price_watermelon * num_watermelon

# show the total price
print(total_price)
```

Note: here you may want to ask a question that is relevant to the information that you would like to collect so that the user knows what information to provide

Run the script, suppose you want 5 apples, then put in 5 as the answer to the question, then click enter and you will get:

```
>>> [evaluate lab2_task1.py]
    How many apples? 5
    Traceback (most recent call last):
      File "E:\Work_OneDriveSync\OneDrive - York University\course\EECS1015\2023-2024\Fall\Slides\Lec2\lab2_task1.py", li
        total_price = price_apples * num_apples + price_peaches * num_peaches + price_watermelon * num_watermelon
    builtins.TypeError: can't multiply sequence by non-int of type 'float'
```

This is because all the input are strings, and in our script we try to multiply it with float (i.e., `'5' * 2.94`). This operation is not allowed in Python. To fix let, let's convert user input to an integer with `int()`, so the solution looks like:

```
# specify fruit quantity
num_apples = input("How many apples? ")
num_apples = int(num_apples)
num_peaches = 4
num_watermelon = 1

# specify fruit unit price
price_apples = 2.94
price_peaches = 3.99
price_watermelon = 7.99

# calculate the total price
total_price = price_apples * num_apples + price_peaches * num_peaches + price_watermelon * num_watermelon

# show the total price
print(total_price)
```

You may rewrite the first two lines of code as:

```
# specify fruit quantity
num_apples = int(input("How many apples? "))
num_peaches = 4
num_watermelon = 1

# specify fruit unit price
price_apples = 2.94
price_peaches = 3.99
price_watermelon = 7.99

# calculate the total price
total_price = price_apples * num_apples + price_peaches * num_peaches + price_watermelon * num_watermelon

# show the total price
print(total_price)
```

Execute your code, give the input 5, you will get:

```
>>> [evaluate lab2_task1.py]
    How many apples? 5
    38.65
```

You can provide any positive integers, and it will give you the price.

Q1.9 Now modify your code so that it can also take information from the user about how many peaches and watermelons that the users would like to purchase.

Solution:

```python
# specify fruit quantity
num_apples = int(input("How many apples? "))
num_peaches = int(input("How many peaches? "))
num_watermelon = int(input("How many watermelons? "))

# specify fruit unit price
price_apples = 2.94
price_peaches = 3.99
price_watermelon = 7.99

# calculate the total price
total_price = price_apples * num_apples + price_peaches * num_peaches + price_watermelon * num_watermelon

# show the total price
print(total_price)
```

Run your code, and provide the quantities:

```
>>> [evaluate lab2_task1.py]
    How many apples? 7
    How many peaches? 10
    How many watermelons? 2
    76.46000000000001
```

Again, you can provide any positive integers as the quantity.

When you submit the code, in order to accommodate the auto-grader, please only print the final result we want. In this case, it is the `total_price`. Please also do not make it fancy, just print this variable as we will compare what you print with the desired result.

Submission

Steps:

- Go to PrairieLearn to submit your work.
- In Lab 2 in PrairieLearn, click Task 1. Copy and the paste the code to the text editor. The text editor is NOT an appropriate IDE. You are NOT recommended to write code there directly. You should work on an IDE and test your code there.
- Click Save & Grade. You will see the grading result shortly.
- You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need to spend some time debugging!).

Note:

- Your grade will not appear on eClass. We will get the grades from PrairieLearn at the end of the semester
- Since we provide solutions to this question, it is ok to submit it as is.
- In order for the autograder to work properly:
    - You must NOT change the order of the first few lines of taking input
    - You must NOT add more print statement when submit the code
    - You must NOT change the line with the print statement
    - The print statement should be the last line of your code
- If you have trouble with the submission, here is a video with instructions: https://yorku.zoom.us/rec/share/9V8PPMNTghQWnCFzETCm8ZbOYr7MuG8-mdDyuntcyBUJsVAv4589cr2LvZ1c4qPg.KWXqw-xS8AXwF5Xz?startTime=1693258246000

Rubrics:

- You will get full mark for this question if you submit the solution (with no typo) to the required location.

Task 2: Debugging (30 pts)

In this task, you will be provided with a potential script for a problem. However, this script does not work. You need to debug the code based on what you learned in task 1 and in the lecture to make it work.

The problem is to count the total number of wheels given different numbers of vehicles:

- A bicycle: 2 wheels
- A tricycle: 3 wheels
- A car: 4 wheels
- A huge truck: 18 wheels

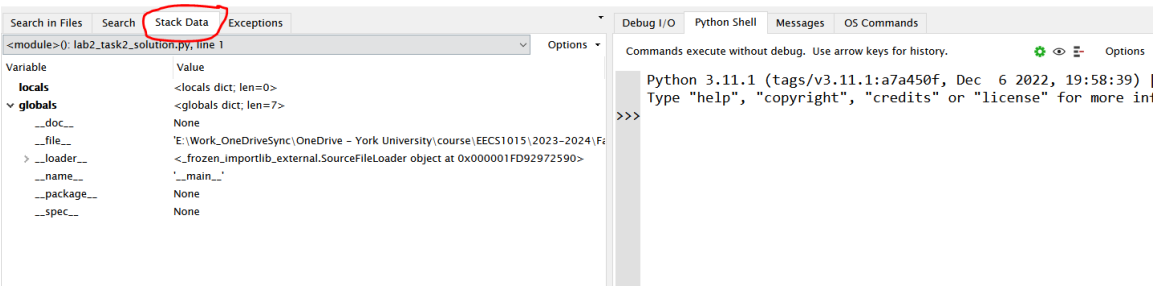
For example, given 1 bicycle, 0 tricycle, 2 cars and 5 trucks, there are in total:

$$2 \times 1 + 3 \times 0 + 4 \times 2 + 18 \times 5 = 100 \text{ wheels}$$

The code to debug (lab2_task2.py) may have syntax errors, run-time errors, and semantic errors. To help with debugging, you can use the debugger to execute the code line by line (you should fix the syntax errors before doing this). To use the debugger, click:

Click OK in the pop-up window.

Then the line that is currently executing will be highlighted in red. Click on the Stack Data tab to see the current value of local variables to help debugging:



To execute the next line, click on this icon:

If you still have trouble with debugging, here is a demonstration video that uses the debugger with **task 1**: https://yorku.zoom.us/rec/share/9V8PPMNTghQWnCFzETCm8ZbOYr7MuG8-mdDyuntcyBUJsVAv4589cr2LvZ1c4qPg.KWXqw-xS8AXwF5Xz?startTime=1693258404000

Submission

In order for the autograder to work properly:

- You must NOT change the order of the first few lines of taking input

- You must NOT add more print statement when submit the code
- You must NOT change the line with the print statement
- The print statement should be the last line of your code

Rubrics:

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will receive 15 pts if your script correctly provide solution to our example (i.e., given 1 bicycle, 0 tricycle, 2 cars and 5 trucks, there are 100 wheels in total)
- You will receive 8 pts if your script correctly provide solution to another example
- You will receive 7 pts if your script correctly provide solution to an example where the number of vehicles is randomly generated

Task 3: Implementation (10 pts)

In this task, you will implement a simple script yourself by expanding lab2_task3.py. It calculates the largest number of donuts one can buy after asking the user how much money the user has. Each donut costs $2.3

For example, with 5 dollars, you can only buy 2 donuts.

You should complete line 2 (asking how much money the user has) and line 5 (how many donuts can be bought) in lab2_task3.py. You can add more lines if you need.

Note:

- You cannot buy half a donut or ¾ donut.
- You can convert from string to float with the `float()`. For example `float('3.4')` will give you the number 3.4
- In order for the autograder to work properly
    - You must NOT add more print statement when submit the code
    - You must NOT change the line with the print statement
    - The print statement should be the last line of your code

Hint:

- Try to figure out how you should make the calculation (which operation you should use) by using the interactive mode.
- If the code does not work as you planned, you can use the debugger to figure out the problem.
- If you still cannot figure out the problem with a debugger, try to identify the problem and isolate/abstract the problem in a different script and ask for help from the TA.
- If you have trouble identifying the problem and/or isolating the problem, you can ask the TA how to do so. (Note: Your TA cannot debug the code for you)

Submission

In order for the autograder to work properly:

- You must NOT change the order of the first few lines of taking input
- You must NOT add more print statement when submit the code
- You must NOT change the line with the print statement
- The print statement should be the last line of your code

Rubrics:

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will receive 5 pts if your script correctly provide solution to our example (i.e., with 5 dollars, you can buy 2 donuts.)
- You will receive 3 pts if your script correctly provide solution to another example
- You will receive 2 pts if your script correctly provide solution to an example with random input

Task 4: Implementation (10 pts)

In this task, you will implement a more complicated script yourself by expanding lab2_task4.py. It calculates the score of the course (not our course). The marking scheme includes the following items with corresponding weight:

- Labs: 5%
- Assignments: 20%
- Midterm: 25%
- Final: 50%

The course has two labs, the one with the higher score will count. For example, a student got the following scores (you can assume that the scores for each submission are integers):

- Lab 1: 95
- Lab 2: 25
- Assignment:60
- Midterm: 70
- Final: 80

Since Lab 1 has a higher score, the final lab score is 95. To calculate the score of the course:

$$95 \times 0.05 + 60 \times 0.2 + 70 \times 0.25 + 80 \times 0.5 = 74.25$$

Hint:

- You must not change the order of line 2 to line 6. The autograder assumes this order.
- You can add any lines of code after line 9
- You may want to use one of the built-in functions in the after-class puzzles to calculate the lab grades

Submission

In order for the autograder to work properly:

- You must NOT change the order of the first few lines of taking input
- You must NOT add more print statement when submit the code
- You must NOT change the line with the print statement
- The print statement should be the last line of your code

Rubrics:

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will receive 5 pts if your script correctly provide solution to our example (i.e., with the scores listed above, the course score is 74.25)
- You will receive 3 pts if your script correctly provide solution to another example
- You will receive 2 pts if your script correctly provide solution to an example with random input

Task 5: Implementation (10 pts)

In this task, you will write a script from scratch and no starter code is provided. The script is used to calculate one's BMI and print out the results. The formula of BMI is: $BMI = \frac{kg}{m^2}$, where kg is the user's weight, and m is height in meters (e.g., 177.8 cm, 75.0 kg is BMI = 23.72). Moreover, your solution should be rounded to two decimal points (Hint: you should use what you have learned in class to find the document and learn to use the function **round**). You script should take the following input from the user:

- Weight
- Height

Requirement

- The input must be in the order of first considering weight (the user will provide the weight in kg), and then height (the user will provide the height in cm).
- Your code should not take any other input
- The print statement should be the last line of your code
- Make sure you just print the value, and no other texts
- You should remove other print statements
- You can assume one's height is positive.
- You should pay attention to the calculation order (e.g., adding parathesis when necessary)

Submission

You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need to spend some time debugging!).

Rubrics

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will receive 5 pts if your script correctly provide solution to our example (i.e., with the scores listed above, the bmi is 23.72)
- You will receive 3 pts if your script correctly provide solution to another example
- You will receive 2 pts if your script correctly provide solution to an example with random input

Task 6: Implementation (10 pts)

In this task, you will write a script from scratch and no starter code is provided. It takes four numbers from the user, print the average of the largest number and smallest number.

For example, the numbers are 3.5, 6.7, -3.5, 0.24. The largest number is 6.7, and the smallest number is -3.5, so the script will print the average of these two numbers (e.g., (6.7 + (-3.5)) / 2):

1.6

Requirement

- Your script should take four numbers (e.g., four lines of input()).
- Your code should not take any other input
- The print statement should be the last line of your code
- Make sure you just print the value, and no other texts
- You should remove other print statements

Submission

You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need to spend some time debugging!).

Rubrics

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will receive 5 pts if your script correctly provide solution to our example (i.e., with the example inputs above, your script should output 1.6)
- You will receive 3 pts if your script correctly provide solution to another example
- You will receive 2 pts if your script correctly provide solution to an example with random input