EECS 1015 Lab 5

Goal:

- To get familiarized with global and local variables.
- To learn about function scoping

Tasks:

1. Guide you through the process of plan and write a function that takes into account of the scoping for global and local variables
2. Learn to debug a function with variables defined under different scopes
3. Learn to plan and write for a function that calls other functions
4. Learn to plan and write a more complicated function that calls other functions
5. Learn to come up with test cases in a thorough manner
6. Learn to come up with test cases in a thorough manner

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 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 in the exams and it is important to learn how to triage your tasks.

Note: For question 3 – 6, you must come up with a function description in your own words, and cannot copy the words from handout.

Task 1: Follow the Steps (30 pts)

This task is to help you familiarize yourself with scoping for global and local variables.
In this task, your goal is to write functions to calculate the area and circumference of a circle.

Q1.1 Start by completing signature and design recipe of the function circle_area().

Think about what the input should be, what is needed to calculate the area of a circle? And for the output, what are you trying to get with this function? Create some test case to check whether the function is doing what you are planning to do.

The formula to calculate the area of a circle is $A = \pi r^2$

Solution:

The formula involves radius r and pi π, pi doesn't change for different radius of circle, while radius vary for different circle. Therefore, we should have radius r as the input to the function. We want to use pi and r to calculate area.

```
def circle_area(radius: float) -> float:
    """
    This function takes radius r as input and return the area of the circle
    using the formula Area = pi * r ** 2

    radius is in the range of 0.0 to 1000.0

    >>> circle_area(2.0)
    12.56
    >>> circle_area(4.0)
    50.24
    """
```

Q1.2 Assume you can only have a radius within the range 0.0 to 1000.0 . Please Write an assertion that checks this precondition.

Solution:

You can add an assertion to check the value and the type of the radius, which should be in float and is in the range from 0.0 to 1000.0

```
def circle_area(radius: float) -> float:
    """
    This function takes radius r as input and return the area of the circle
    using the formula Area = pi * r ** 2

    radius is in the range of 0.0 to 1000.0

    >>> circle_area(2.0)
    12.56
    >>> circle_area(4.0)
    50.24
    """
    assert type(radius) == float, "radius must be a float"
    assert 0.0 <= radius <= 1000.0 , "radius must be in range of 0.0 to 1000.0"
```

Q1.3 Think about how to convert the area formula into code and implement it in your function body.

Solution:

Multiplying radius with itself and then multiplying with pi gives you the area of circle.

```
def circle_area(radius: float) -> float:
    """
    This function takes radius r as input and return the area of the circle
    using the formula Area = pi * r ** 2

    radius is in the range of 0.0 to 1000.0

    >>> circle_area(2.0)
    12.56
    >>> circle_area(4.0)
    50.24
    """
    assert type(radius) == float, "radius must be a float"
    assert 0.0 <= radius <= 1000.0 , "radius must be in range of 0.0 to 1000.0"

    pi = 3.14

    return pi * radius ** 2
```

Q1.4 Now considering writing another function circle_circumference (), repeats the above steps to implement this function. The formula to calculate the circumference of a circle is $C = 2\pi r$

Solution:

Similar to the function circle_area(radius), to calculate circumference of circle, we would also need the radius. Therefore, it should have the same set of input.

def **circle_circumference(radius)**

param: radius – the radius of the circle

output: circumference – the function should output the circumference of the circle using the formula $C = 2\pi r$

it has the same precondition. Radius should be in the range of 0 to 1000.

```python
def circle_circumference(radius: float) -> float:
    """
    This function takes radius r as input and return the circumfernece of the circle
    using the formula Circumference = 2 * pi * r

    radius is in the range of 0.0 to 1000.0

    >>> circle_circumference(2.0)
    12.56
    >>> circle_circumference(4.0)
    25.12
    """

    assert type(radius) == float, "radius must be a float"
    assert 0.0 <= radius <= 1000.0 , "radius must be in range of 0.0 to 1000.0"

    pi = 3.14

    return 2 * pi * radius
```

Q1.5 Notice how π appeared in both functions. Is there a better way to use this value in both functions?

Solution:

Think about the main point of this Task, global variable. Keep in mind the global constant should be in capital case with underscore separating words!

You can define a global constant pi and use this instead of a local variable pi in each of the functions. This way you can change pi by changing only the global constant.

```python
PI = 3.14

def circle_area(radius: float) -> float:
    """
    This function takes radius r as input and return the area of the circle
    using the formula Area = pi * r ** 2

    radius is in the range of 0.0 to 1000.0

    >>> circle_area(2.0)
    12.56
    >>> circle_area(4.0)
    50.24
    """
    assert type(radius) == float, "radius must be a float"
    assert 0.0 <= radius <= 1000.0 , "radius must be in range of 0.0 to 1000.0"

    return PI * radius ** 2
```

```python
def circle_circumference(radius: float) -> float:
    """
    This function takes radius r as input and return the circumfernece of the circle
    using the formula Circumference = 2 * pi * r

    radius is in the range of 0.0 to 1000.0

    >>> circle_circumference(2.0)
    12.56
    >>> circle_circumference(4.0)
    25.12
    """

    assert type(radius) == float, "radius must be a float"
    assert 0.0 <= radius <= 1000.0 , "radius must be in range of 0.0 to 1000.0"

    return 2 * PI * radius
```

Q1.6 Now instead of using 3.14 as pi value, change the global constant pi to 3.14159 and run the test case. Why is it no longer providing valid output? Please modify the code so that it works with new value.

To run the test case, you will need to import doctest module, then at the end, run doctest.testmod()

Solution:

This is one problem we must deal with when using global constant, once you update this global value, you must update everywhere in the code that this value is related to. In this case, it would be the test case. The test case is in the assumption that the pi value is 3.14. When we change this value to 3.14159, the test case output is no longer the same. Therefore, we also need to change the test case to account for the value update.

Submission

- Go to eClass -> our course -> Week 6 -> Practice -> Lab -> Lab 5 Submission
- Copy your code to Task 1
- 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 get full mark for this question if you submit the solution (with no typo) to the required location on Prairielearn before the deadline.

Note:

- Since we provide solutions to this question, it is ok to submit it as is.
- In order for the auto grader to work properly:
    - You must NOT change the name of the function
    - You must NOT change the order of the arguments
    - You must NOT change the output of the function
- The auto grader may take some time to grade

Task 2: Debugging (30 pts)

This task is about global variables. The objective is to troubleshoot the provided program in lab5_task2.py , which was intended to employ global variables defined in other functions. However, the program is generating errors. Your task is to fix the program so that it runs smoothly without any issues with the intended function description.

(Hint: think about why using global keyword is a bad coding practice)

Note that function1 and function2 might utilize the value differently in the autograder, but the general structure of the function remains the same.

Below are the function requirements for each function:

function1(value):

- This function initializes the global variable x and utilizes value and global variable y to update x.
- param: value, used as part of global variable x initialization.
- return: None
- Please do not change this function
- Do not submit this function

function2(value):

- This function initializes the global variable y and utilizes value to update y.
- param: value – value that is used as part of global variable y initialization.
- return: None
- Please do not change this function
- Do not submit this function

problematic_function(x_value, y_value) -> float:

- This function takes x_value and y_value, then it returns the value x + y
- param: x_value, y_value, both are used for the initialization of global variable x, y.
- return: float value x + y
- This is the function to debug
- Please submit this function ONLY

Running the function with the following parameters with the current definition of function1 and function2 should have the following output:

```
>>> problematic_function(10.0, 5.0)
20.0
```

After completing the question, think about whether the code after debugging follows good coding style. Is it truly necessary to use global variables? How to refactor (e.g., improve) the code so that it has a better coding style? Hint: Think about what we have covered in class.

Requirement

- The solution you provide must match the function description.
- you can only change the problematic_function

Submission

Copy the Python code to Lab 5 -> Task 2 on Prairielearn.

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:

- In order for the autograder to work properly:
  - You must ONLY submit the problematic_function
  - You must NOT change the name of the function.
  - You must NOT change the order of the arguments.
  - You must NOT change the output of the function.
- Do not hard code anything. When we test your code, we will use a slightly different definition of function1 and function2 (The structure will be the same, but the value of x and y will be different).

Rubrics:

- You will not receive any mark if you do not submit it to the designated location on Prairielearn before the deadline
- You will not receive any mark if the code doesn't match the function description.
- You will not receive points if there are syntax errors in the code.
- Otherwise

- You will receive 20 pts for passing test cases provided in the starter code.
  - 5 pts for each passing each of the following test case (If you use the current definition of function1 and function2, you will see the following return value. **The autograder uses different value of x and y, so the expected value will be different in the autograder**.)
    - problematic_function(10.0, 5.0): 20.0
    - problematic_function(10.0, -5.0): 0.0
    - problematic_function(5.9, 2.1): 10.1
    - problematic_function(67, 8.5): Assertion Error
- You will receive 10 pts for passing the 2 additional test cases.
  - 5 pts for each passing test case

Task 3: Implementation (10 pts)

Imagine you have been recently hired to be the regional manager of a retail business with 2 stores. Your boss has asked you to report on the total monthly revenue of all the stores. However, counting the sales of each store manually is too much work for a salary of $20 an hour. Therefore, to save some time, you have decided to write a program that keeps track of the revenue of the stores.

Each store has its own monthly sales, and the way each store calculates their revenue is also different. Store 1 can make a profit equal to 1/5 of its monthly sales, meaning that the revenue of store 1 is calculated by monthly sales / 5. Similarly, store 2 can make a profit equal to 1/8 of its monthly sales. meaning that the revenue of store 2 can be calculated by monthly sales / 8. Note that all stores can only have non-negative monthly sales. At the end you would want to report the combined revenue of both the stores.

You need to expand lab5_task3.py, and please implement the following function:
get_total_revenue (store1_monthly_sales: float, store2_monthly_sales: float) -> float.

You are recommended to define multiple functions as in the starter code, you might want to think about how these functions fit into the story:
get _store1_revenue (store1_monthly_sales: float) -> float
get _store2_revenue (store2_monthly_sales: float) -> float

Given
store1_monthly_sales = 800.0
store2_monthly_sales = 600.0
Then
get_total_revenue(store1_monthly_sales, store2_monthly_sales) should return 235.0

This function is relatively simple, but it is a good practice to de-compose your code into multiple functions and each function has its own purpose. This will be convenient for future development for several reasons (imagine this is a code that actually deployed in real-world scenarios):

- One may need to update how the revenue of store 1 should be calculated. If we have a separate function for it, it is easy to identify where the code is
- We can write test cases for each of these functions and each group of test cases only

test for one thing, rather than a combination of things. For example, if everything is in one function and it fails a test case, it is hard to predict whether one did not calculate the revenue of individual stores incorrectly or calculating the total revenue incorrectly.

- If new stores are opened or existing stores are closed, we only need to modify the function that calculates total revenue, without updating the functions that calculate the revenue of specific stores.
- It is also easier if we want to expand the set of functions. For example, we may want to add a function to find which store has the highest revenue. If we separate them into different functions, we can call the functions that calculate the revenue of individual stores, without having to calculate them again.

Requirement

- You must implement all 3 functions.
- You must NOT change the order of the arguments
- You must NOT change the output of the function
- The inputs and outputs should be in floats
- If you implement functions other than get_total_revenue() function, please also submit them. Although we will not check the function design recipe for other functions, you are highly recommended to include these for other functions.

Submission

Copy the Python code to Lab 5 -> Task 3 on Prairielearn.

You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission (You need 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 not receive any mark if your code violates the above requirements
- Otherwise
    - You will receive 0.3 pts if you provide a description to the get_total_revenue() function
    - You will receive 0.3 pts if you provide the argument annotations correctly for the get_total_revenue() function

- You will receive 0.4 pts if you provide at least 4 distinct test cases and your code passes those test cases (Note: those test cases should be different from the test cases provided below) for the get_total_revenue() function
- You will receive 3 pts if your function passes the provided test cases
    - 1 pts for each passing test case
        - get_total_revenue(800.0, 600.0) -> 235.0
        - get_total_revenue(-400.0, 600.0) -> AssertionError
        - get_total_revenue(0.0, 37.4) -> 4.675
- You will receive 6 pts if your function passes the 3 additional test cases.
    - 2 pts for each passing test case

Task 4: Implementation (10 pts)

Imagine you are in a competitive eating competition with your rival. The more food you eat, the more food credit you will gain. You decided to write a program that keeps track of your food credit and your opponent's food credit during the competition. The credits each participant receives will be based on how much food one eats during the competition and the weight of the participant.

You belonged to the Flyweight class where your food credit is calculated by how much food you ate during the competition multiply by 10. Your opponent belonged to the Straw weight class where their food credit is calculated by how much food they ate during the competition multiply by 15. For example, if you consume 20.0 kg of food, the raw credit that you will receive is 20.0 x 10 = 200.0 credits. If your opponent consumes 10.0 kg of food, the raw credit they will receive is 10.0 x 15 = 150.0 credits.

To avoid giving out too much food credit to the participants, the competition introduced a special rule that the total credit one receives is influenced by the other competitor, which results in the final credit one receives being the raw credit minus 20% of the opponent's raw credit.

For example, if your raw credit is 200.0 and your opponent's raw credit is 150.0, then your final credit is 200.0 – 0.2 x 150.0 = 170.0. Similarly, your opponent's final credit is 150.0 – 0.2 x 200.0 = 110.0. The minimum credit one will receive is 0. Note that participants can only have non-negative raw credit.

You need to expand lab5_task4.py, and please implement the following functions that calculate your final food credit and your opponent's final food credit. Again, you will need to define multiple functions and reuse them. Implementing the two functions by themselves is not challenging, but how to make the design choice is the challenging part. Try to find the common code in the two functions and how you can define help functions so that you can reuse code (Note: the process of reorganizing your code after the code works is called refactoring).

my_food_credit(my_food_ate, opp_food_ate)
opponent_food_credit(opp_food_ate, my_food_ate)

Given:

my_food_ate = 20.0

opp_food_ate= 10.0

Then

my_food_credit(my_food_ate, opp_food_ate) should return 170.0

opponent_food_credit(opp_food_ate, my_food_ate) should return 110.0

Requirement

- You must implement the 2 functions.
- You must NOT change the order of the arguments
- You must NOT change the output of the function
- Inputs and outputs should be in floats
- You can add more functions if needed (actually, you are encouraged to do so). Please submit all functions if you do so. It is highly recommended to write function design recipe for all functions.

Submission

Copy the Python code to Lab 5 -> Task 4 on Prairielearn.

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 not receive any mark if your code violates the above requirements
- Otherwise
    - You will receive 0.5 pts if you provide a description to each function:
        - my_food_credit(my_food_ate, opp_food_ate)
        - opponent_food_credit(opp_food_ate, my_food_ate)
    - You will receive 0.5 pts if you provide the argument annotations correctly:
        - my_food_credit(my_food_ate, opp_food_ate)
        - opponent_food_credit(opp_food_ate, my_food_ate)
    - You will receive 3 pts if your function passes the provided test cases
        - 0.5 pts for passing each test case

- my_food_credit(20.0, 10.0)
- opponent_food_credit(10.0, 20.0)
- my_food_credit("10", 20.0) -> AssertionError
- opponent_food_credit("10", 20.0) -> AssertionError
- my_food_credit(6.12, 36.172)
- opponent_food_credit(263.1, 126.6)

- You will receive 6 pts if your function passes the 4 additional test cases
  - 1.5 pts for each passing test case

Task 5: Implementation (10 pts)

In this task, you will implement a function and no starter code is provided. The function should be called absolute_value and accept one argument. Given a string in the format of an absolute number, the function should return a floating number. For example:

```
>>> absolute_value("|2.3|")
2.3
>>> absolute_value("|-3|")
3.0
```

Note: The precondition of this question can be tricky. Please try to come up with a comprehensive list of test cases.

This question may be challenging. Please note that you have until the end of Friday to submit your work if you need more time. There are also additional lab office hours available. This is great practice to think about the test cases and think about how to handle them.

Submission

Copy the Python code to Lab 5 -> Task 5 on Prairielearn.

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 not receive any mark if your code violates the above requirements
- Otherwise
    - You will receive 0.25 pts if you provide a description of the function.
    - You will receive 0.25 pts if you provide the argument annotations correctly
    - You will receive 0.5 pt if you include at least 4 test cases and your code pass those test cases

- You will receive 9 pts if your function passes our test cases (Note: if you did not pass a test case, you can take a look at the error message on PrairieLearn as it will show the parameter)

You do not need anything more advanced than what we have covered in class (e.g., if statement) to complete the question (be creative of using the Booleans!). However, even if you pass all the test cases, please be reminded that this may still not be a comprehensive list of test cases.

Task 6: Even number (10 pts)

In this task, you will implement a function and no starter code is provided. The function should be called is_even and accept one argument. The function will return True if the number provided is an even number and False otherwise. For example:

```
>>> is_even(4)
True
>>> is_even(-5)
False
```

Note: The precondition of this question can be tricky. Please try to come up with a comprehensive list of test cases.

Submission

Copy the Python code to Lab 5 -> Task 6 on Prairielearn.

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 not receive any mark if your code violates the above requirements
- Otherwise
    - You will receive 0.7 pts if you provide a description of the function.
    - You will receive 0.8 pts if you provide the argument annotations correctly
    - You will receive 1.5 pt if you include at least 4 test cases and your code pass those test cases
    - You will receive 7 pt if your function passes our test cases (Note: if you did not pass a test case, you can take a look at the error message on PrairieLearn as it will show the parameter)