

# EECS 1015 Lab 12

## Goal

- Learning about classes and objects, enabling the creation of custom data types.
- Developing crucial debugging skills to identify and fix various types of errors in code.

## Tasks

1. Guide you through the process of Creation of custom data types using classes.
2. Developing essential debugging skills by identifying and fixing errors.
3. Applying object-oriented programming concepts to real-world scenarios.

Total credit: 100 pts

**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.**

## Task 1: Follow the Steps (50 pts)

In this lab, you will learn about classes and objects in Python. Classes are used to create custom data types, and objects are instances of these classes. We will work on creating a simple class for representing geometric shapes, specifically rectangles.

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

### Q1.1 Design and Define the Rectangle Class

Your first task is to design and create the **Rectangle** class. Follow these steps:

- Define a class named **Rectangle**.

In Python, a class is defined using the **class** keyword, followed by the class name (in this case, "Rectangle"). Make sure the class name follows the PEP-8 convention, using lowercase letters and underscores to separate words.

- Inside the **Rectangle** class, create an **\_\_init\_\_** method.

The **\_\_init\_\_** method is a constructor that initializes the class's instance variables. It is called when an object of the class is created. It accepts the parameters **length** and **width**.

- Within the **\_\_init\_\_** method, initialize the instance variables **length** and **width** by assigning the values passed as arguments to them and make sure the parameters are valid. Instance variables are attributes that store data for each object created from the class. In this case, we need to store the length and width of the rectangle.

Solution:

```
class Rectangle:
    """ A class for Rectangle."""

    def __init__(self, length: float, width: float) -> None:
        """
        The constructor of the Rectangle class.

        >>> rectangle_1 = Rectangle(4.6, 1.2)
        >>> rectangle_1._length
        4.6
        >>> rectangle_1._width
        1.2
        """
        assert isinstance(length, float) and length > 0, "length is not a positive float"
        assert isinstance(width, float) and width > 0, "width is not a positive float"

        self._length = length
        self._width = width
```

Note: here we use a new built-in function to check the type of a variable. You can use the `help()` function to find the documentation of this function. In the past, we used `type(length) == float`. This is not a good coding style. However, since you were not introduced with the instance concept, we use it as a temporary solution. Now that you know what instance is, so please use the `isinstance()` method to check for the type of the variable.

### Q1.2 Implement the `calculate_area` method

- Inside the **Rectangle** class, create a method named **`calculate_area`**.

Methods in a class are functions that can perform operations on the class's data (instance variables).

- Implement the **`calculate_area`** method by writing code that calculates the area of the rectangle. The area of a rectangle is calculated by multiplying its length and width. The **`calculate_area`** method should return the computed area as its result.
- 

Solution:

```
def calculate_area(self) -> float:
    """
    A class method that calculate the area of the rectangle

    >>> rectangle_1 = Rectangle(4.6, 1.2)
    >>> rectangle_1.calculate_area()
    5.52
    """

    return self._length * self._width
```

### Q1.3 Implement the `calculate_perimeter` method

- Inside the **Rectangle** class, create a method named **`calculate_perimeter`**.
- Implement the **`calculate_perimeter`** method by writing code that calculates the perimeter of the rectangle. The perimeter of a rectangle is calculated by adding the lengths of all four sides, which is 2 times the sum of length and width.
- The **`calculate_perimeter`** method should return the computed perimeter as its result.

Solution:

```
def calculate_perimeter(self) -> float:
    """
    A class method that calculate the perimeter of the rectangle

    >>> rectangle_1 = Rectangle(4.6, 1.2)
    >>> rectangle_1.calculate_perimeter()
    11.6
    """

    return 2 * (self._length + self._width)
```

Think about what the advantage is to create class methods compared with writing functions that take in two parameters and return the area and perimeter. E.g.,

- calculate\_area(width: float, length: float) -> float
- calculate\_perimeter(width: float, length: float) -> float

#### Q1.4 Implement the setter and getter methods

Solution:

Note: Here we provide example code for getter and setter for the variable length. Please write your own getter and setter for the variable width.

```
def get_length(self) -> float:
    """
    A class method that get the value of the length

    >>> rectangle_1 = Rectangle(4.6, 1.2)
    >>> rectangle_1.get_length()
    4.6
    """

    return self._length

def set_length(self, length: float) -> float:
    """
    A class method that get the value of the length

    >>> rectangle_1 = Rectangle(4.6, 1.2)
    >>> rectangle_1.set_length(1.3)
    >>> rectangle_1._length
    1.3
    """

    assert isinstance(length, float) and length > 0, "length is not a positive float"

    self._length = length
```

Think about what the advantage is to use setter and getter methods compared with manipulating the attributes directly.

## Submission

- Go to eClass -> our course -> Week 13 -> Practice -> Lab -> Lab 12 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 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 autograder to work properly:
  - o You must NOT change the name of the methods nor the class
  - o You must NOT change the order of the arguments
- The autograder may take some time to grade

## Task 2: Debugging (30 pts)

Sam worked on a simple inventory management system for a small retail store, and wanted to create a Python program that uses classes and objects to keep track of products and their quantities. Each product has a name, and price, and quantity in stock. Sam needed to create a class to represent the products and write methods to add and remove products from the inventory due to stocking or selling, and calculate the total value of the inventory. Sam wrote the complete program, however there are a few errors in the code. Help Sam to fix the errors.

The code to debug (lab12\_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).

### Requirement

- Your submission must be based on the provided code. There are simpler ways of doing this, but you cannot delete all the provided code and write your own.

### Submission

Copy the Python code to Lab 12 -> 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 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.
- You will receive 5 points if your code passes the test for the constructor for class Product
- You will receive 5 points if your code passes the test for the reduce\_quantity method for class Product. In this test case:

```
>>> laptop = Product("laptop", 2030.5, 2)
>>> laptop.reduce_quantity(3)
>>> laptop.get_quantity()
0
```

- You will receive 5 points if your code passes the test for the \_find\_product method for class Inventory. In this test case:

```
>>> laptop = Product("laptop", 2030.5, 2)
>>> retail_store_inventory = Inventory()
>>> retail_store_inventory._products = [laptop]
>>> retail_store_inventory._find_product("laptop") is laptop
True
```

- You will receive 5 points if your code passes the test for the `_find_product` method for class `Inventory`. In this test case:
 

```
>>> laptop = Product("laptop", 2030.5, 2)
>>> retail_store_inventory = Inventory()
>>> retail_store_inventory._products = [laptop]
>>> retail_store_inventory._find_product("tablet") == None
True
```
- You will receive 5 points if your code passes the test for the `stock_product` method for class `Inventory`. In this test case:
 

```
>>> laptop_1 = Product("laptop", 2030.5, 2) >>> laptop_2 =
Product("laptop", 2030.5, 3) >>> retail_store_inventory =
Inventory() >>> retail_store_inventory.stock_product(laptop_1)
>>> retail_store_inventory.stock_product(laptop_2) >>>
retail_store_inventory._find_product("laptop").get_quantity() 5
```
- You will receive 5 points if your code passes the test for the `stock_product` method for class `Inventory`. In this test case:
 

```
>>> laptop_1 = Product("laptop", 2030.5, 2) >>>
laptop_2 = Product("laptop", 102.09, 2) >>>
retail_store_inventory = Inventory() >>>
retail_store_inventory.stock_product(laptop_1) >>>
retail_store_inventory.stock_product(laptop_2)
AssertionError
```