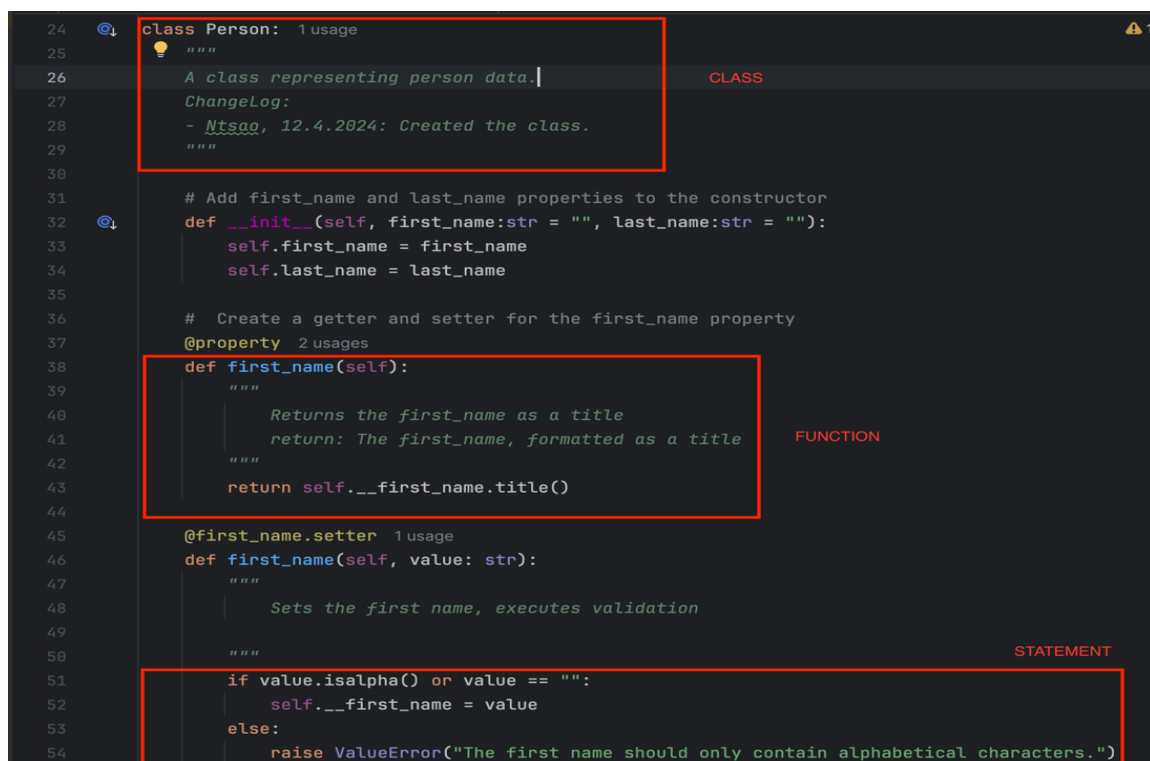


# Classes, Functions, and Beyond

## Introduction:

In module 07 we continue to dive deeper into the roles of functions and classes and how they maintain our code. We tackle new terminologies and concepts that are essential to understanding object-oriented programming. I'll walk through how Assignment 07 contributed to my understanding of the key concepts.

- 1.) Statements, Functions, Classes – We have been working with **statements** since the beginning of the class. Each statement performs a specific action or task, such as assigning a value to a variable, controlling the flow of execution, or defining a function. Statements are often found wrapped in **functions** when they are meant to be called or used multiple times. While Classes house the data for the object and the functions that operate on the data.



```
24 class Person: 1 usage
25     """
26     A class representing person data.
27     ChangeLog:
28     - Ntsao, 12.4.2024: Created the class.
29     """
30
31     # Add first_name and last_name properties to the constructor
32     def __init__(self, first_name:str = "", last_name:str = ""):
33         self.first_name = first_name
34         self.last_name = last_name
35
36     # Create a getter and setter for the first_name property
37     @property 2 usages
38     def first_name(self):
39         """
40         Returns the first_name as a title
41         return: The first_name, formatted as a title
42         """
43         return self.__first_name.title()
44
45     @first_name.setter 1 usage
46     def first_name(self, value: str):
47         """
48         Sets the first name, executes validation
49         """
50
51         if value.isalpha() or value == "":
52             self.__first_name = value
53         else:
54             raise ValueError("The first name should only contain alphabetical characters.")
```

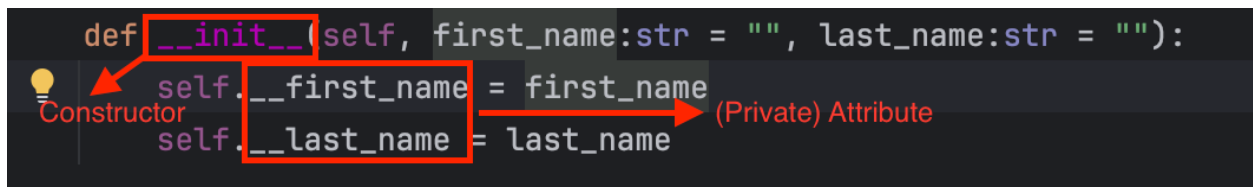
The screenshot shows a code editor with a dark theme. The code defines a Python class named `Person`. Three sections of the code are highlighted with red boxes and labeled on the right side of the editor:

- CLASS**: Points to the class definition `class Person:` and its docstring.
- FUNCTION**: Points to the `first_name` property getter method.
- STATEMENT**: Points to the validation logic inside the `first_name` setter method, specifically the `if` statement and the `raise ValueError` call.

Figure 1 – In this assignment, `class Person` is used as the **parent class or the superclass**

## 2.) Types of Classes:

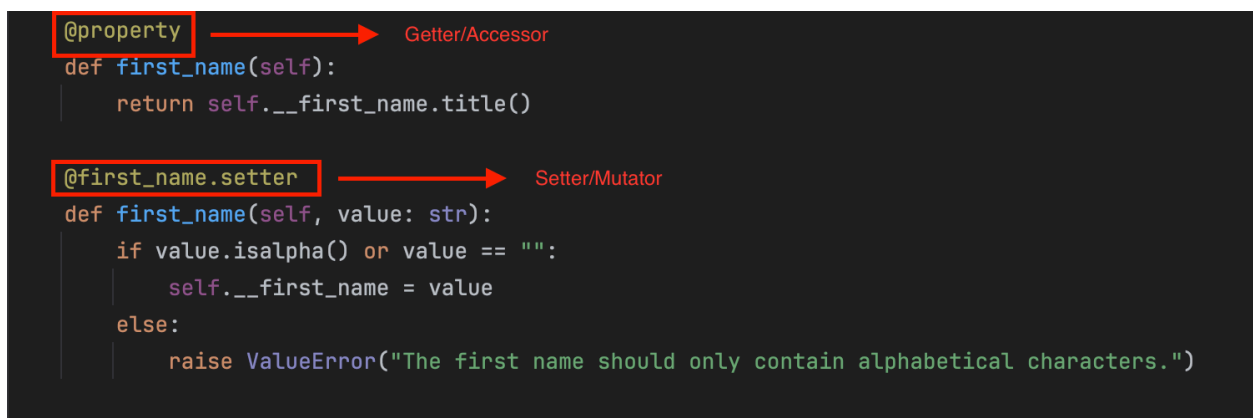
- a. Presentation Class - It manages the display of information and often translates user actions into commands that are sent to the Processing Class or Data Class.
- b. Processing Class - It is responsible for manipulating data received from the Presentation Class and working with it, often by calling methods in the Data Class.
- c. Data Class - Primarily used to store and manage data. This includes **attributes** (basically the **variables** that store the data inside the class), **constructors** (special method used to create and set up new objects) and **properties** (functions designed to manage attribute data).



```
def __init__(self, first_name:str = "", last_name:str = ""):
    self.__first_name = first_name
    self.__last_name = last_name
```

The image shows a Python code snippet for a class constructor. The `__init__` method is highlighted with a red box and labeled "Constructor" with a lightbulb icon. The assignments `self.__first_name = first_name` and `self.__last_name = last_name` are also highlighted with red boxes and labeled "(Private) Attribute" with a red arrow pointing to the right.

NOTE: Private attributes are used to prevent the code from being altered when used outside of the class.



```
@property
def first_name(self):
    return self.__first_name.title()

@first_name.setter
def first_name(self, value: str):
    if value.isalpha() or value == "":
        self.__first_name = value
    else:
        raise ValueError("The first name should only contain alphabetical characters.")
```

The image shows Python code for class properties. The `@property` decorator is highlighted with a red box and labeled "Getter/Accessor" with a red arrow. The `@first_name.setter` decorator is also highlighted with a red box and labeled "Setter/Mutator" with a red arrow. The code defines a `first_name` property with a getter and a setter that validates the input.

- 3.) Child class/subclass - can inherit data and behaviors (properties and methods) from an existing class (parent class), this is termed as "inherited code". I like to think of the parent class/superclass as the base code, then subclass is to add on other properties or methods on top of that code. This was you do not have to be redundant and enter the exact same script all over again.

```

89 # Create a Student class the inherits from the Person class
90 class Student(Person): 9 usages
91     """
92     A collection data about students
93
94     ChangeLog: (Who, When, What)
95     Ntsaq,12.4.2024, Created Class
96     Ntsaq,12.4.2024, Added properties and private attributes
97     """
98     # call to the Person constructor and pass it the first_name and last_name data
99     # add a assignment to the course_name property using the course_name parameter
100     def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
101         super().__init__(first_name=first_name, last_name=last_name)
102         self.course_name = course_name

```

The (Person) indicates that class Student will inherit its methods and properties

This also indicates inherited code

- 4.) Data Validation – Similar to error handling, data validation in Python is just making sure that the values you give to an object's attributes are correct and follow the rules you set.

```

46 @first_name.setter
47 def first_name(self, value: str):
48     if value.isalpha() or value == "":
49         self.__first_name = value
50     else:
51         raise ValueError("The first name should only contain alphabetical characters.")
52

```

Data Validation

## Summary:

We have uncovered even more uses of classes, functions and statements and how to use them to our advantage. It has been interesting to learn and practice how to convert a script using one method to another. Once you get the hang of these, you can write code that's cleaner, faster, and way easier to understand. It's like learning how to use the right tools to build something awesome!