MIT World Peace University

Python Programming
Second Year B. Tech, Semester 4

Learning the Basics of *Obeject Oriented Programming with Python*

Assignment No. 6

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

March 18, 2023

Contents

1 Aim

Write a program to read 3 subject marks and display pass or failed using class and object.

2 Objectives

1. To learn and implement concepts of Object Oriented Programming in Python.

3 Problem Statement

Object oriented programming concepts to display Students grade as Pass or Fail.

4 Theory

4.1 Object Oriented Programming in Python

4.1.1 Definition

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another. There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.

4.1.2 Example

```python
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles
```

4.1.3 Explanation

In the above example, we have defined a class Car with attributes *make, model, year, odometer_reading*. The *__init__* method is used to initialize the attributes of the class. The *get_descriptive_name* method is used to return the *long_name* variable which is a string. The *read_odometer* method is used to print the *odometer_reading* attribute. The *update_odometer* method is used to update the *odometer_reading* attribute. The *increment_odometer* method is used to increment the *odometer_reading* attribute.

4.2 Inheritance

4.2.1 Definition

Inheritance is a way to form new classes using classes that have already been defined. The newly formed classes are called derived classes, the classes that we derive from are called base classes. Important benefits of inheritance are code reuse and reduction of complexity of a program. The derived classes (descendants) override or extend the functionality of base classes (ancestors).

4.2.2 Example

```python
class ElectricCar(Car):
    def __init__(self, make, model, year):
        super().__init__(make, model, year)
        self.battery = Battery()
```

4.2.3 Explanation

In the above example, we have defined a class ElectricCar which inherits the Car class. The *__init__* method is used to initialize the attributes of the class. The *super* method is used to inherit the attributes of the parent class. The *self.battery* attribute is used to store the battery instance.

4.3 Polymorphism

4.3.1 Definition

Polymorphism is the ability to present the same interface for differing underlying forms (data types).

4.3.2 Example

```python
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def get_descriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def make_sound(self):
        print("Vroom!")

class ElectricCar(Car):
    def __init__(self, make, model, year, battery_size):
        super().__init__(make, model, year)
        self.battery_size = battery_size

    def describe_battery(self):
        print(f"This car has a {self.battery_size}-kWh battery.")

    def make_sound(self):
        print("Silent, but deadly.")

class HybridCar(Car):
    def __init__(self, make, model, year, gas_mileage, battery_size):
        super().__init__(make, model, year)
        self.gas_mileage = gas_mileage
        self.battery_size = battery_size

    def describe_battery(self):
        print(f"This car has a {self.battery_size}-kWh battery.")

    def describe_gas_mileage(self):
        print(f"This car gets {self.gas_mileage} miles per gallon.")

    def make_sound(self):
        print("Vroom! But also, silent sometimes.")

my_car = Car('audi', 'a4', 2020)
my_electric_car = ElectricCar('tesla', 'model s', 2020, 100)
my_hybrid_car = HybridCar('toyota', 'prius', 2020, 50, 1.5)

cars = [my_car, my_electric_car, my_hybrid_car]

for car in cars:
    print(car.get_descriptive_name())
    car.make_sound()
    if isinstance(car, ElectricCar) or isinstance(car, HybridCar):
        car.describe_battery()
    if isinstance(car, HybridCar):
        car.describe_gas_mileage()
```

5 Input and Output

5.1 Input

Reading marks of students from keyboard.

5.2 Output

Display Students Grade as Pass/Fail.

6 Code

6.1 Assignment 6 - Classes and Objects

6.1.1 Write a program to read 3 subject marks and display pass or failed using class and object.

[15]:
```
class student(object):
    """Class to manage student stuff. """
    def __init__(self, name, subjects, marks):
        self.name = name
        self.subjects = subjects
        self.marks = marks
        self.results = {}
        self.results = self.calculate_results()

    def calculate_results(self):
        for sub in self.subjects:
            self.results[sub] = 'Fail'
        for sub, mark in zip(self.subjects, self.marks):
            self.results[sub] = 'Pass' if mark >= 50 else 'Fail'
        return self.results

    def get_results(self):
        return self.results
```

[17]:
```
Ramesh = student("Ramesh", ["Maths", "Physics", "Chemistry"], [50, 60, 70])
Suresh = student("Suresh", ["Maths", "Physics", "Chemistry"], [50, 40, 30])
Tony_Stark = student("Tony Stark", ["Maths", "Physics", "Chemistry"], [100, 100,␣
 ↪100])

for sub, res in zip(Ramesh.subjects, Ramesh.results.values()):
    print(f'The Result of {sub} for {Ramesh.name} is {res}')
for sub, res in zip(Suresh.subjects, Suresh.results.values()):
    print(f'The Result of {sub} for {Suresh.name} is {res}')
for sub, res in zip(Tony_Stark.subjects, Tony_Stark.results.values()):
    print(f'The Result of {sub} for {Tony_Stark.name} is {res}')
```

```
The Result of Maths for Ramesh is Pass
The Result of Physics for Ramesh is Pass
The Result of Chemistry for Ramesh is Pass
The Result of Maths for Suresh is Pass
The Result of Physics for Suresh is Fail
The Result of Chemistry for Suresh is Fail
The Result of Maths for Tony Stark is Pass
The Result of Physics for Tony Stark is Pass
The Result of Chemistry for Tony Stark is Pass
```

7 Conclusion

Studied Object oriented programming in Python and classes and objects are used to display Students Grade as Pass/Fail.

8 FAQ

1. **What is difference between procedural proxgramming and Object oriented programming.**

   Procedural Programming:

   - Procedural programming is a programming paradigm, based on the concept of the procedure call.
   - A procedure, also known as a subroutine, is a routine that contains a series of programming
   - statements to perform a specific task.
   - A procedure is called by another part of the program using a procedure call.
   - The main advantage of procedural programming is that it breaks a large, complex task into smaller and simpler sub-tasks.
   - The disadvantage of procedural programming is that it does not model the real world very well.
   - In procedural programming, the data and the functions that operate on the data are not associated with any particular class or object.
   - In procedural programming, the data is passed to the functions as parameters.
   - It is different from Object Oriented Programming (OOP) in the sense that in OOP, the data and the functions that operate on the data are associated with a class and an object.

   Object Oriented Programming:

   - Object-oriented programming is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.
   - A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self").
   - In OOP, computer programs are designed by making them out of objects that interact with one another.
   - An object can be defined as a data field that has unique attributes and behavior.
   - Objects are basically the things you think about first in designing a program and they are also the units of code that are eventually derived from the process.
   - A Class is a blueprint for the object.
   - The Main advantage of OOP is that it provides a clear modular structure for programs. It is good for defining abstract data types, and also allows programmers to create full reusable applications with less code and shorter development times. It also makes software easier to maintain and modify.

- The disadvantage of OOP is that it makes programs slower at runtime because of the overhead involved in the creation of the classes and objects.
- In OOP, the data is not passed to the functions, but the functions are associated with the data.

2. **What is superclass?**

   Superclass: The Superclass is the class from which a subclass inherits. The subclass inherits all the attributes and methods of the superclass.

   Example

```
class Parent:
        def __init__(self):
                self.value = "Inside Parent"

        def get_value(self):
                return self.value

class Child(Parent):

        def __init__(self):
                super().__init__()
                self.value = "Inside Child"

        def get_value(self):
                return "Inside Child"

        def get_parent_value(self):
                return super().get_value()
```

```
Inside Child
Inside Parent
```

3. **Explain instance variables and Class variables?**

   Instance Variables: Instance variables are variables whose value is assigned inside a constructor or method with self. Instance variables are not shared between objects. Each object contains its own copy of the instance variable.

   Example

```
class Student:
        def __init__(self, name):
                self.name = name

        def get_name(self):
                return self.name
```

```
>>> student1 = Student("Ramesh")
>>> student2 = Student("Suresh")
>>> student1.get_name()
'Ramesh'
>>> student2.get_name()
'Suresh'
```

Class Variables: Class variables are variables whose value is assigned in the class. Class variables are shared between all objects. For example, the company name of all the employees is the same. So it is assigned in the class. It saves memory.

Example

```
class Student:
company = "MegaCorp"

        def __init__(self, name):
                self.name = name

        def get_name(self):
                return self.name

        def get_company(self):
                return self.company
```

```
>>> student1 = Student("Ramesh")
>>> student2 = Student("Suresh")
>>> student1.get_name()
'Ramesh'
>>> student2.get_name()
'Suresh'
>>> student1.get_company()
'MegaCorp'
>>> student2.get_company()
'MegaCorp'
```

4. **What is the purpose of the init method?**

init: The init method is similar to constructors in C++ and Java. Constructors are used to initialize the object's state. The task of constructors is to initialize(assign values) to the data members of the class when an object of class is created. Like methods, a constructor also contains collection of statements (i.e. instructions) that are executed at time of Object creation. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

Example

```
class Student:
def __init__(self, name):
self.name = name
```

```
def get_name(self):
return self.name


>>> student1 = Student("Ramesh")
>>> student2 = Student("Suresh")
>>> student1.get_name()
'Ramesh'
>>> student2.get_name()
'Suresh'
```

5. **What is 'self' in Python?**

   self: The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class. It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class.