MODULE IV

Python objects and classes , **Python packages** - Flask, SQLAlchemy, REST API, Django and various other libraries Graphical user interfaces;

Event-driven programming paradigm, creating simple GUI; buttons, labels, entry fields, dialogs; widget attributes - sizes, fonts, colors layouts, nested frames.

Database Concepts - SQL, CRUD (Create, Read, Update, Delete), Roles and Permissions

RA20 APTT

Class and Object

•**Class:** The class is a user-defined data structure that **binds the data members and methods into a single unit**.

•Class is a blueprint or code template for object creation and Using a class, you can create as many objects as you want.

Object: An object is an instance of a class and we use the object of a class to perform actions.

Every object has the following property.

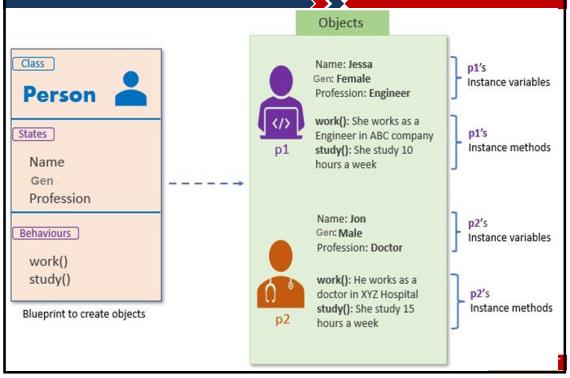
Identity: Every object must be uniquely identified.

State: An object has an attribute that represents a state or property of an object,

Behavior: An object has methods that represent its behavior.

RA20 APTT

Class and Object



Class and Object

Objects

Class: Person

- **State**: Name, Gen , Profession
- **Behavior**: Working, Study

Using the above class, we can create multiple objects that depict different states and behavior.

Object 1: Jessa

- **State**:
  - Name: Jessa
  - Gen : Female
  - Profession: Software Engineer
- **Behavior**:
  - Working: She is working as Software developer in ABC Company
  - Study: She study 2 hours a day

Object 2: Jon

- **State**:
  - Name: Jon
  - Gen : Male
  - Profession: Doctor
- **Behavior**:
  - Working: He is working as Doctor
  - Study: She study 5 hours a day

As you can see, Jessa is female, and she works as a Software engineer. On the other hand,

Creating Class

Class:is defined by using the **class keyword**.

```
class class_name:

    <statement 1>
    <statement 2>
    .
    .
    <statement N>
```

•**`class_name:`** It is the name of the class

•**`statements:`** Attributes and methods

RA20 APTT

Creating Class and Object

•An object is essential to work with the class attributes.

•The object is created using the class name.

• When we create an object of the class, it is called instantiation.

•The object is also called the instance of a class.

•Object creation is divided into two parts in Object Creation and Object initialization

•Internally, the __new__ is the method that creates the object.

•And, using the __init__() method is **a constructor** to initialize the object.

•A constructor is a special method used to create and initialize an object of a class.

•The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

RA20 APTT

Creating Class

```
class MyClass:
    x = 5

p1 = MyClass()
print(p1.x)
```

•The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

•The __new__() is **a static method of the object class**

•.

•When you create a new object by calling the class, Python calls the **__new__()** method to create the object first and then calls the **__init__()** method to initialize the object's attributes.

•Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Creating Class and Object

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

p1 = Person("Priya", 23)

print(p1.name)
print(p1.age)
```

The **__init__()** function is called automatically every time the class is being used to create a new object

Object Methods

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

  def myfunc(self):
      print("Hello my name is " + self.name)


p1 = Person("Priya", 23)

print(p1.name, p1.age)
p1.myfunc()
```


Creating Class and Object

constructor — Parameters to constructor

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

    def show(self):
        print("Name is:", self.name, "and percentage is:", self.percentage)
```

Instance variable (self.name)

Instance variable (self.percentage)

Instance method (show)

Object of class

```
stud = Student("Jessa", 80)
stud.show()
# Output: Name is: Jessa and percentage is: 80
```


SELF PARAMETER

```
class Person:
  def __init__(thisobj, name, age):
    thisobj.name = name
    thisobj.age = age

  def myfunc(thisobj):
      print("Hello my name is " + thisobj.name)


p1 = Person("Priya", 23)

print(p1.name, p1.age)
p1.myfunc()
```

•The self parameter is a reference to th current instance of the class, and is used access variables that belongs to the class.

•It does not have to be named self , you ca call it whatever you like, but it has to be th first parameter of any function in the class.

•

•I have named it thisobj in present example

Deleting Object Property

```
class Person:
  def __init__(thisobj, name, age):
    thisobj.name = name
    thisobj.age = age

p1 = Person("Priya", 23)

del p1.age

print(p1.name, p1.age)
```

ERROR : Person' object has no attribute 'age'

Deleting Object Property

```
class Person:
  def __init__(thisobj, name, age):
    thisobj.name = name
    thisobj.age = age

p1 = Person("Priya", 23)

del p1

print(p1.name, p1.age)
```

ERROR : name 'p1' is not defined '

RA20 APTT

Pass Statement

class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

```
class Person:
 pass
```

RA20 APTT

Creating Object

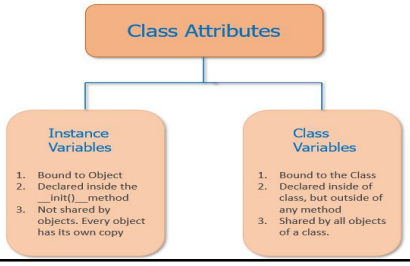
```
class Person:
    def __init__(self, name, gen , profession):
        # data members (instance variables)
        self.name = name
        self.gen = gen
        self.profession = profession

    # Behavior (instance methods)
    def show(self):
        print('Name:', self.name, 'Gen:', self.gen, 'Profession:', self.profession)

    # Behavior (instance methods)
    def work(self):
        print(self.name, 'working as a', self.profession)

# create object of a class
jessa = Person('Jessa', 'Female', 'Software Engineer')

# call methods
jessa.show()
jessa.work()
```

```
Name: Jessa Gen: Female Profession: Software Engineer
Jessa working as a Software Engineer
```

Class Attributes

•Instance Variable : The instance variables are attributes attached to an instance of a class. We define instance variables **in the constructor** __init__() method of a class.

•Class variable: A class variable is a variable that is declared inside of class, **but outside of any instance method or __init__() method.**



RA20 APTT

Class Attributes

•**Objects : do not share instance attributes**, Instead, every object has its copy of the instance attribute and is unique to each object.

• All **instances of a class share the class variables**, However, unlike instance variables, the value of a class variable is not varied from object to object.

•Only one copy of the static variable will be created and shared between all objects of the class.

•**Accessing properties and assigning values:**

•An instance attribute can be accessed or modified by using the dot notation: instance_name.attribute_name.

•A class variable is accessed or modified using the class name

RA20 APTT

Class Attributes

```
class Student:
    # class variables
    school_name = 'ABC School'

    # constructor
    def __init__(self, name, age):
        # instance variables
        self.name = name
        self.age = age

s1 = Student("Harry", 12)
# access instance variables
print('Student:', s1.name, s1.age)

# access class variable
print('School name:', Student.school_name)

# Modify instance variables
s1.name = 'Jessa'
s1.age = 14
print('Student:', s1.name, s1.age)

# Modify class variables
Student.school_name = 'XYZ School'
print('School name:', Student.school_name)
```

Output

```
Student: Harry 12
School name: ABC School

Student: Jessa 14
School name: XYZ School
```

RA20 APTT

Class Methods

Inside a Class, we can define three types of methods:

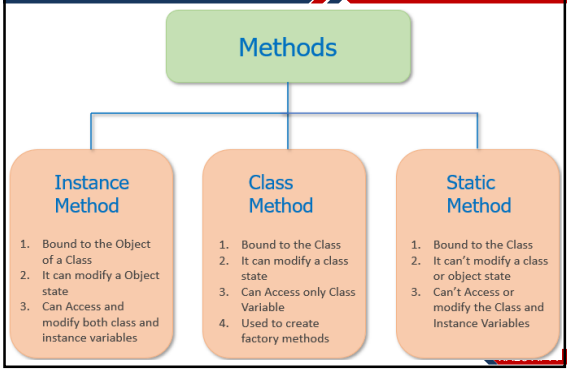
•**Instance Method:** Used to access or modify the object state. If we use **Instance Variables** inside a method, such methods are called instance methods.

•**Class Method:** Used to access or modify the class state. In method implementation, if we use only class variables , then such type of methods we should declare as a class method.

•**Static Method :** It is a general utility method that performs a task in isolation. Inside this method, **we don't use instance or class variable because** this static method doesn't have access to the class attributes.

RA20 APTT

Class Methods



Class Methods

```python
# class methods demo
class Student:
    # class variable
    school_name = 'ABC School'

    # constructor
    def __init__(self, name, age):
        # instance variables
        self.name = name
        self.age = age

    # instance method
    def show(self):
        # access instance variables and class variables
        print('Student:', self.name, self.age, Student.school_name)

    # instance method
    def change_age(self, new_age):
        # modify instance variable
        self.age = new_age

    # class method
    @classmethod
    def modify_school_name(cls, new_name):
        # modify class variable
        cls.school_name = new_name

s1 = Student("Harry", 12)

# call instance methods
s1.show()
s1.change_age(14)

# call class method
Student.modify_school_name('XYZ School')
# call instance methods
s1.show()
```

Output

```
Student: Harry 12 ABC School
Student: Harry 14 XYZ School
```

Class Naming Convention

Rule-1: Class names should follow the **UpperCaseCamelCase** convention

Rule-2: Exception classes should end in "**Error**".

Rule-3: If a class is callable (Calling the class from somewhere), in that case, we can give a class name like a **function**.

Rule-4: Python's built-in classes are typically lowercase words

RA20 APTT

Object Properties

Lets us consider **Vehicle as class** of all **Automobiles.** one of its object could be a **CAR, motor cycle etc**

RA20 APTT

Modify Object Properties

We can set or modify the object's properties after object initialization by calling the property directly using the dot operator **Obj.PROPERTY = value**

```python
class Fruit:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def show(self):
        print("Fruit is", self.name, "and Color is", self.color)

# creating object of the class
obj = Fruit("Apple", "red")

# Modifying Object Properties
obj.name = "strawberry"

# calling the instance method using the object obj
obj.show()
# Output Fruit is strawberry and Color is red
```

Delete object properties and Object

We can delete the object itself by using the del keyword. After deleting it, if we try to access it, we will get an error.

```
class Employee:
    depatment = "IT"

    def show(self):
        print("Department is ", self.depatment)

emp = Employee()
emp.show()

# delete object
del emp

# Accessing after delete object
emp.show()
# Output : NameError: name 'emp' is not defined
```

Exercise

1: Create a **Vehicle Class** with instance attributes.

2. Create a Vehicle class without any variables and methods.

```
class Vehicle:
    def __init__(self, max_speed, mileage):
        self.max_speed = max_speed
        self.mileage = mileage

modelX = Vehicle(240, 18)
print(modelX.max_speed, modelX.mileage)
```

```
class Vehicle:
    pass
```

Python Packages

•**Flask** is a web application framework or a micro web framework written in Python.

•Flask was developed byArmin Ronacher.

•Flask is classified as a microframework because it does not require particular tools or libraries.

•Flask is based on the Web Server Gateway Interface (WSGI) and has been adopted as a standard for Python web application development.

•WSGI is a specification for a universal interface between the web server and the web applications.

•It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions

Python Packages

•**SQLAlchemy :** is basically referred to as the toolkit of Python SQL that provides developers with the flexibility of using the SQL database.

•It allows Python developers to work with the language's own objects, and not write separate SQL queries.

•They can basically use **Python to access and work with databases.**

•SQLAlchemy is also an Object Relational Mapper which is a technique used **to convert data between databases or OOP languages such as Python.**

• Thus it allows the **object model and database schema** to develop in a cleanly decoupled way from the beginning.

Python Packages

•REST API, **: R**epresentational **S**tate **T**ransfer (REST) is an architectural style that defines a set of constraints to be used for creating web services.

•**REST API** is a way of accessing web services in a simple and flexible way without having any processing.

• REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth,

• Being simple and flexible making it more suitable for internet usage.

•It's used to fetch or give some information from a web service.

•All communication done via REST API uses only HTTP request.

DJANGO

•Django is a Python-based web framework that allows you to quickly create efficient web applications.

•It is also called batteries included framework because Django provides built-in features for everything including Django Admin Interface, default database – SQLlite3, etc.

• We need a similar set of components to handle user **authentication (signing up, signing in, signing out), a management panel, forms, upload files, e**tc. Django gives you ready-made components to use and that too for rapid development.

•Django is a back-end server side web framework.

•Django is free, open source and written in Python