Student Name: _____ **Roll No:** _____ **Section:** _____

Lab No. 03

Lab 03 – Introduction to Simple Classes, Attributes and Methods

Objectives:

- Understanding the concepts of classes and init() method in classes
- Use of Classes and subclasses by Inheritance

1. The _init_() Method

The __init__() method is profound for two reasons. Initialization is the first big step in an object's life; every object must be initialized properly to work properly. The second reason is that the argument values for __init__() can take on many forms.

Because there are so many ways to provide argument values to __init__(), there is a vast array of use cases for object creation. We take a look at several of them. We want to maximize clarity, so we need to define an initialization that properly characterizes the problem domain.

Before we can get to the __init__() method, however, we need to take a look at the implicit class hierarchy in Python, glancing, briefly, at the class named object. This will set the stage for comparing default behavior with the different kinds of behavior we want from our own classes.

In this example, we take a look at different forms of initialization for simple objects (for example, playing cards). After this, we can take a look at more complex objects, such as hands that involve collections and players that involve strategies and states.

Python is a multi-paradigm programming language. Meaning, it supports different programming approach.

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

- attributes
- behavior

Let's take an example:

Student Name: _____ **Roll No:** _____ **Section:** _____

Parrot is an object,

- name, age, color are attributes
- singing, dancing are behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

In Python, the concept of OOP follows some basic principles:

1	***Inheritance***	A process of using details from a new class without modifying existing class.
2	***Encapsulation***	Hiding the private details of a class from other objects.
3	***Polymorphism***	A concept of using common operation in different ways for different data input.

Python Object Inheritance

Inheritance is the process by which one class takes on the attributes and methods of another. Newly formed classes are called child classes, and the classes that child classes are derived from are called parent classes.

It's important to note that child classes override or extend the functionality (e.g., attributes and behaviors) of parent classes. In other words, child classes inherit all of the parent's attributes and behaviors but can also specify different behavior to follow. The most basic type of class is an object, which generally all other classes inherit as their parent.

When you define a new class, Python 3 it implicitly uses object as the parent class. So the following two definitions are equivalent:

Student Name: _____ **Roll No:** _____ **Section:** _____

Exercise 1: Write a class of Dog, each dog must be of species type mammal. Each dog has its name and age. The class can have method for description () and sound () which dog produces. Create an object and perform some operations.

```
class Dog:

    # Class Attribute
    species = 'mammal'

    # Initializer / Instance Attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def description(self):
        return "{} is {} years old".format(self.name, self.age)

    # instance method
    def speak(self, sound):
        return "{} says {}".format(self.name, sound)

# Instantiate the Dog object
razer = Dog("Razer", 6)

# call our instance methods
print(razer.description())
print(razer.speak("Woof Woof"))
```

Student Name: _____ Roll No: _____ Section: _____

Exercise 2: Write a class of Dog, each dog must be of species type mammal. Each dog has its name and age. The class can have method for description () and sound () which dog produces. Now this time you need to create two sub classes of Dogs one is Bull Dog and other is Russell Terrier Create few objects and perform some operations including the inheritance.

```
# Parent class
class Dog:

    # Class attribute
    species = 'mammal'

    # Initializer / Instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def description(self):
        return "{} is {} years old".format(self.name, self.age)

    # instance method
    def speak(self, sound):
        return "{} says {}".format(self.name, sound)


# Child class (inherits from Dog class)
class RussellTerrier(Dog):
    def run(self, speed):
        return "{} runs {}".format(self.name, speed)


# Child class (inherits from Dog class)
class Bulldog(Dog):
    def run(self, speed):
        return "{} runs {}".format(self.name, speed)


# Child classes inherit attributes and
# behaviors from the parent class
thunder = Bulldog("Thunder", 9)
print(thunder.description())

# Child classes have specific attributes
# and behaviors as well
print(thunder.run("slowly"))

spinter = Bulldog("Spinter", 12)
print(spinter.description())
print(spinter.run("fast"))
```

Student Name: _____ **Roll No:** _____ **Section:** _____

```
roger = RussellTerrier("Roger", 5)
print(roger.description())
print(roger.run("quickly"))
```

Student Name: _____ **Roll No:** _____ **Section:** _____

Exercise 3: Extending question number 2, now we need to check that either the different dog classes and their objects link with each other or not. In this case we need to create a method to find either it's an instance of each other objects or not.

```
# Parent class
class Dog:

    # Class attribute
    species = 'mammal'

    # Initializer / Instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def description(self):
        return "{} is {} years old".format(self.name, self.age)

    # instance method
    def speak(self, sound):
        return "{} says {}".format(self.name, sound)


# Child class (inherits from Dog() class)
class RussellTerrier(Dog):
    def run(self, speed):
        return "{} runs {}".format(self.name, speed)


# Child class (inherits from Dog() class)
class Bulldog(Dog):
    def run(self, speed):
        return "{} runs {}".format(self.name, speed)


# Child classes inherit attributes and
# behaviors from the parent class
thunder = Bulldog("Thunder", 9)
print(thunder.description())

# Child classes have specific attributes
# and behaviors as well
print(thunder.run("slowly"))

# Is thunder an instance of Dog()?
print(isinstance(thunder, Dog))

# Is thunder_kid an instance of Dog()?
```

Student Name: _____ **Roll No:** _____ **Section:** _____

```
thunder_kid = Dog("ThunderKid", 2)
print(isinstance(thunder, Dog))

# Is Kate an instance of Bulldog()
Kate = RussellTerrier("Kate", 4)
print(isinstance(Kate, Dog))

# Is thunder_kid and instance of kate?
print(isinstance(thunder_kid, Kate))
print("Thanks for understanding the concept of OOPs")
```

NOTE:

Make sense? Both thunder_kid and Kate are instances of the Dog() class, while Spinter is not an instance of the Bulldog() class. Then as a sanity check, we tested if kate is an instance of thunder_kid, which is impossible since thunder_kid is an instance of a class rather than a class itself—hence the reason for the TypeError.

Student Name: _____ **Roll No:** _____ **Section:** _____

Programming Exercise (Python)

Task 1: Discuss in detail what you understand by Classes, Objects and find the relation of Instances which may have in some cases and for other it will not, for all the exercises and tasks starting from task 3.

Task 2: Create UML diagrams for all the exercises and Tasks starting from task 3.

Task 3: Create class and sub classes for different types of residential houses. Each house object has different parameters such as number of location of the house, rooms, parking available or not. Price of the house.

Task 4: Create class and sub classes for differ types of vehicles for Toyota Motors. Each car object belongs to some particular model, color, price, type of car such as saloon, luxury or sports. They can be registered in different years and made in different years.

Task 5: Create class and sub classes for different types of plants. Some of the plants have fruits that we can eat and some of them are poisonous. Some are local and some are imported from other countries. Some of the fruits are sweet whereas some are sour. Create parameters according to your own visualizations.

Task 6: You are hired in a mobile company which produces multiple mobiles each year. Select any mobile company create one main class, some sub classes, add some features and methods to operate the mobile and then create some objects for your friends and check how it will work.

Task 7: You are doing programming for UIT admission office. Your task is to create some records for the students. You know that we have four types of students such as BS(CS) and BS(SE) is one type other is BE(Power/ Computer Systems /Telecommunications/Electrical), third type is master students and forth is short courses students. Now your task is to create some classes and subclasses to find how the system will work.

Task 8: You are planning to create a project for Pizza Ordering System. Your primary task is to classify different types of pizza, its flavors, sizes, crust type, price. How you analyze the system, according to your analysis perform the programming objectives.