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: ____________

Exercise1:

```
#class name
class Person:

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

     self.name = name
     self.age = age

# Instantiate the Person object
person1= Person("ali", 6)

print(person1.name,person1.age)

person2 = Person("ahmed", 9)
print(person2.name,person2.age)
```

Student Name: ______________ Roll No: __________ Section: __________

Exercise2:

```
class Person:

     # Initializer / Instance Attributes
  def __init__(self, name, age):
    self.name = name
    self.age = age
# instance method
  def description(self):
    print("Hello my name is " + self.name)

p1 = Person("Ali", 36)

p1.description()
```

Student Name: ______________ Roll No: __________ Section: ________

Exercise 3:

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

  def printdetails(self):
    print(self.name, self.age)
# child class

class Employee(Person):
  def __init__(self, name, age, post):
    # invoking the __init__ of the parent class
   Person.__init__(self, name, age)
   self.post = post

  def Details(self):
    print("Employee Data  Name age and post is ",self.name,
self.age, self.post)

#creation of an object  or an instance
ob = Employee("ali", 35,"Clerk")
ob.Details()
ob.printdetails()
```

Programming Exercise (Python)

Task 1 Create UML diagrams for all the exercises and Tasks

Task 2: 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 3: 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 4: 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.

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