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.

Class:

Class is a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.

A class in Python is a category or set of different elements grouped together that share one or more similarities with one another, but yet distinct from other classes via type, quality and kind. In technical terminology, we can define a class in Python as being a blueprint for individual objects with same or exact behavior.

Object:

Object is one of instances of the class. Which can perform the functionalities which are defined in the class

Self:

Self represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class in python.

__init__:

"__init__" is a reserved method in python classes. It is known as a constructor in object oriented concepts. This method called when an object is created from the class and it allow the class to

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

initialize the attributes of a class.

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.

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:

```
class Car(object):
  """
    blueprint for car
  """

  def __init__(self, model, color, company, speed_limit):
    self.color = color
    self.company = company
    self.speed_limit = speed_limit
    self.model = model


  def Details(self):
      print("Car Details ",self.model, self.color,
self.company,self.speed_limit)


  #1st Object
maruthi_suzuki = Car("ertiga", "black", "suzuki", 60)
maruthi_suzuki.Details()
 #2nd Object
audi = Car("A6", "red", "audi", 80)
audi.Details()
```

Programming Exercise (Python)

Task 1 Create UML diagrams for all the exercises and Tasks

Task 2: Create class residential houses. Each house object has different parameters such as number of location of the house, rooms, parking available ornot. Price of the house.

Task 3: You are hired in a mobile company which produces multiple mobiles each year. Select any mobile company create class, add some features and methods to operate the mobile and then create some objects for your friends and check how it will wok