Chapter 10

A.

1. D
2. C
3. C
4. D
5. C
6. B
7. B
8. B
9. A
10. A
11. C
12. B
13. C
14. A
15. C
16. C
17. B
18. C
19. C
20. C

B.

1. False
2. False
3. True
4. True
5. False
6. True
7. True
8. True
9. True
10. True
11. False
12. False
13. True

14. True
15. True

C.

1. A class is a template for creating objects. Objects have member variables and behavior associated with them. In python a class is created by the keyword class.

2.

Syntax for defining a class in Python is as follows:

```
Class Class_Name:
    Initializer
    attributes
    methods()
    Statement(s)
```

3. Before adding attributes to a class, consider following simple example of class where there are no attributes in it

```
class Demo:
    pass
D1=Demo()     #Instance or Object of the class Demo
print(D1)
Output:
<__main__.Demo object at 0x029B3150>
```

The class **Demo** doesn't contain any data.It doesn't do anything. Thus to make class useful we have to add attributes to it. The syntax to add attribute to a class is as follows

```
Name_of_attribute  = Some_Default_Value
```

Following Program illustrates about adding attributes to a class Demo.

```
class Demo:
    a = 10     #attributes a and b
    b = 20
Ob1 = Demo()
```

4. The class usually consists of two things **instance variable** and **instance methods**. The following is the syntax to add methods to a class.

```
class Class_Name:
```

```
        instance variable;  #instance variable with initialization
         def method_name(Self,parameter_list):#Paramter List is Optional
                        block_of_statements
```

5. In python as we are aware of to add methods to the existing class, the first parameter for each method should be **self**. This is only one specific difference between class methods and ordinary functions. The **self** parameter is used in the implementation of the method, but it is not used when the method is called. The **self** parameter references the object itself. The following program demonstrates the use of self as parameter for a method defined within a class.

```
class Demo:
    a = 10
    b = 20
    def print_data(self):
        print(' The value of a is ',self.a)
        print(' The value of b is ',self.b)
Ob1 = Demo()
Ob1.print_data()
Output
The value of a is  10
The value of b is  20
```

Thus in the above program, self references the contents of current object.

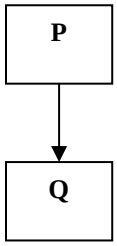
6. The special class attributes or methods are associated with each built in function and operator. Corresponding to special method python internally convert the expression into a call to perform certain operation. For example if programmer wants to perform sum of two operands then they write as x + Y. But when python observes the + operator then it convert that expression i.e. x + y to call a special method __add__. Thus to overload + operator just include the implementation of the special method __add__.

7. The existing classes are main components of inheritance. The new classes are created from existing one. The properties of existing classes are simply extended to the new classes. The new classes created using such method are known as ***derived classes or sub class*** and the existing classes are known as ***base class or super class.*** Therefore, the procedure of creating a new class from one or more existing class is termed as **"inheritance".**

8. Different types of inheritance supported by python are as follows
   - Single Inheritance
   - Multilevel Inheritance
   - Multiple Inheritance

Single Inheritance

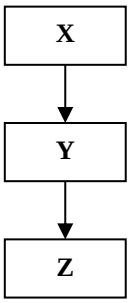
Only one base class is used for derivation of a new class.



P is a base class. Q is a derived class. This type involves one base and derived class. Further, no class is derived from Q.

Multilevel Inheritance

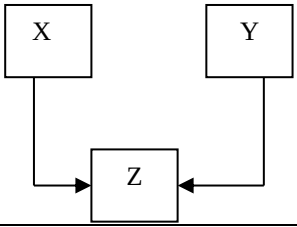
When a class is derived from another derived class i.e. derived acts as base class. Such type of inheritance is known as Multilevel Inheritance.



X is a base class. Y is derived from X. Further; Z is derived from Y. Here, Y is not only a derived class but also a base class for Z. Further, Z can be used as a base class.

Multiple Inheritance

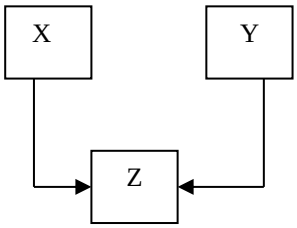
When two or more base classes are used for derivation of a new class is called multiple Inheritance.



X and Y are base classes. Z is a derived class. Class Z inherits properties of both X and Y. Further, Z is not used as a base class.

9. Multiple Inheritance can be defined as follows

When two or more base classes are used for derivation of a new class is called multiple Inheritance.



X and Y are base classes. Z is a derived class. Class Z inherits properties of both X and Y. Further, Z is not used as a base class.

10. **Method overriding**, in object-oriented programming , is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes. The implementation in the subclass overrides the implementation in the superclass by providing a method that has same name, same parameters or signature, and same return type as the method in the parent class.

11. Example to override a method is as follows

```
class A:                #Base Class
    i = 0
    def display(self):
        print(' I am in Super  Class')

class B(A):           #Derived Class
    i = 0
    def display(self):   #Overridden Method
        print(' I am in Sub Class')
```

Thus in case of overriding, the method in derived class should have same name as that of name given to a method in a base class.

12.

```
class Coordinate(object):
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def getX(self):
        return self.x
```

```
    def getY(self):
        return self.y

    def __eq__(self, other):
        assert type(other) == type(self)
        return self.getX() == other.getX() and self.getY() ==
     other.getY()


P1 = Coordinate(10,20)
print(' P1(X, Y) = ',P1.getX(), P1.getY())
P2 = Coordinate(10,20)
print(' P2(X, Y) = ',P2.getX(), P2.getY())
print( P1 == P2)
```

13.

a. A is the parent class

b. B,C and D are the child class

c. <class '__main__.A'>

John

Programming Assignments

1.

```
class Demo():
    def __init__(self):
        self.str1 = ""

    def get_String(self):
        self.str1 = input('Please Enter the String')

    def print_String(self):
        print(' Entered String is: ',self.str1)
        print(' String in Upper Case:')
        print(self.str1.upper())

str1 = Demo()
str1.get_String()
print_String()
```

2.

```
import math
class Circle:
    radius = 0
```

```
    def __init__(self, R):
        self.radius = R


    def get_radius(self):
        return self.radius

    def Calc_area(self):
        return self.radius*self.radius*math.pi

C1 = Circle(10)
print(' Radius = ',C1.get_radius())
print(' Area  of Cirlce = ',C1.Calc_area())
```

3.

```
class Point:
    # A class that represents a 2D Point

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def  Display(self):
        print(' X = ',self.x)
        print(' Y = ',self.y)


    def Translate(self, delta_x, delta_y):
        # Moves this Point delta_x units in the x-direction
        #  and delta_y units in the y direction.
        # delta_x and delta_y must be numbers.
        # Returns the new coordinates.
        self.x += delta_x
        self.y += delta_y
        return (self.x, self.y)

P1 = Point(10,20)
P1.Display()
print(P1.Translate(2,2))
```

4.

```
import math
class Circle:
```

```
    radius = 0
    def __init__(self, R):
        self.radius = R


    def get_radius(self):
        return self.radius

    def Calc_area(self):
        return self.radius*self.radius*math.pi

class Cylinder(Circle):
    height = 0
    def __init__(self,R,H):
        self.height = H
        super().__init__(R)

    def Calc_area(self):
        return 2*math.pi*super().get_radius()*self.height

C1 = Circle(10)
C2 = Cylinder(20,10)
print(' Radius = ',C1.get_radius())
print(' Radius of Cylinder =',C2.radius)
print(' Area of Cylinder = ',C2.Calc_area())
print(' Area  of Cirlce = ',C1.Calc_area())
```

5.

```
import math
class Shape:
    color = 0

    def __init__(self, C):
        self.color = C

    def get_Color(self):
        return self.color

class Triangle(Shape):
    base = 0
    height = 0
    def __init__(self,B,H,C):
        self.base = B
        self.height = H
```

```
        super().__init__(C)

    def Tring_Details(self):
        print(' Base =',self.base)
        print(' Height =',self.height)
        print(' Color = ',self.color)

    def Calc_area(self):
        return 0.5*self.base * self.height

class Rectangle(Shape):
    length = 0
    breadth = 0
    def __init__(self,L,B,C):
        self.length = L
        self.breadth = B
        super().__init__(C)

    def Rect_Details(self):
        print(' Length =',self.length)
        print(' Length =',self.breadth)
        print(' Color = ',self.color)

    def Calc_area(self):
        return self.length * self.breadth


C1 = Rectangle(10,20,'Red')
C2 = Triangle(10,15,'Pink')
print('------Rectangle Details -------------')
C1.Rect_Details()
print(' Area = ',C1.Calc_area())

print('------Triangle Details -------------')
C2.Tring_Details()
print(' Area = ',C2.Calc_area())
```