SECTION-A

a) State one difference between variable and constant.

b) Define Destructors.

d) What are used defined data types?

e) Define preprocessor directives.

f) "Abstract class may be instantiated". This statement is true or false?

g) Define scope resolution operator.

h) What are static functions?

i) Define polymorphism

j) Can a constructor be overloaded?

k) Write the use of break statement?

l) Write one difference between constructor and destructor.

SECTION-B

Note: Short answer type questions.

Q.2 i) Write short notes on

a) Encapsulation b) Data hiding

ii) What are the advantages and disadvantages of object oriented programming approach.

iii) How is static binding different from dynamic binding?

iv) Explain private inheritance using an example.

v) What are inline functions? What are the advantages?

vi) Write a program in PYTHON illustrating the concept of for loop.

vii) Explain virtual base class?

viii) Describe parametrised constructor with an example.

ix) Define union. What is the difference between a union and a structure.

x) Differentiate between constructor and destructor.

xi) Write a program in PYTHON that opens and closes a file.

SECTION-C

Q.1 Write a class named box with data member width of the box and member functionsetwidth to set the width of the box and a non member function printwidth to print thewidth of box . printwidth function is the friend of setwidth function. Use necessaryobjects and access them.

Q.2 Write a python program to find area of circle,triangle and rectangle.Use functionoverloading concept.

Q.3 What do you understand by object oriented programming? Compare procedure oriented programming and object oriented programming.

Q.4 Write a program in PYTHON to multiply two matrices.

Q.5 What are the differences between formal and actual parameters. Explain with example.

Q.6 What are the various types of control structures in PYTHON? Explain any 3 with suitable examples.

Q.7 What is abstract function? Explain the need of abstract function with an example

Section D- OUTPUT Based Questions:

Qsn1.

```
# Python program showing

# abstract base class work


from abc import ABC, abstractmethod


class Polygon(ABC):


    # abstract method

    def noofsides(self):

        pass


class Triangle(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")

# Driver code
R = Triangle()
R.noofsides()
```

```
K = Quadrilateral()

K.noofsides()


R = Pentagon()

R.noofsides()


K = Hexagon()

K.noofsides()


Qsn2. # Python program showing

# abstract base class work


from abc import ABC, abstractmethod

class Animal(ABC):


        def move(self):

                pass


class Human(Animal):


        def move(self):

                print("I can walk and run")


class Snake(Animal):
    def move(self):
        print("I can crawl")

class Dog(Animal):

    def move(self):
        print("I can bark")

class Lion(Animal):

    def move(self):
        print("I can roar")

# Driver code
R = Human()
R.move()

K = Snake()
K.move()

R = Dog()
R.move()

K = Lion()
K.move()
```

Qsn3. # Python program showing

```
# implementation of abstract
# class through subclassing
import abc
class parent:
    def geeks(self):
        pass
class child(parent):
    def geeks(self):
        print("child class")

# Driver code
print( issubclass(child, parent))
print( isinstance(child(), parent))
```

Qsn4.
```
# Python program invoking a
# method using super()
import abc
from abc import ABC

class R(ABC):
    def rk(self):
        print("Abstract Base Class")

class K(R):
    def rk(self):
        super().rk()
```

```
                print("subclass ")


# Driver code

r = K()

r.rk()
```

Qsn5. # Python program showing

```
# abstract class cannot

# be an instantiation

from abc import ABC,abstractmethod


class Animal(ABC):

        @abstractmethod

        def move(self):

                pass

class Human(Animal):

        def move(self):

                print("I can walk and run")


class Snake(Animal):

        def move(self):

                print("I can crawl")


class Dog(Animal):

        def move(self):

                print("I can bark")
```

```
class Lion(Animal):

        def move(self):

                print("I can roar")


c=Human()

c.move()

print(dir(ABC))
```

Qsn6. # Python program showing

```
# abstract class cannot

# be an instantiation

from abc import ABC,abstractmethod


class Animal(ABC):

        @abstractmethod

        def move(self):

                pass

class Human(Animal):

        def move(self):

                print("I can walk and run")


class Snake(Animal):

        def move(self):

                print("I can crawl")


class Dog(Animal):

        def move(self):
```

```
        print("I can bark")


class Lion(Animal):

    def move(self):

        print("I can roar")


c=Animal()
```