Page No.
Date :

Assignment No. 1

Q.1) Justify the polymorphism with programme in python.

→ (1) polymorphism is one of the fundamental concepts in OOPS and it allows object of different classes to be treated as object of common superclass.

(2) In python, polymorphism is implemented through method overriding and method overloading.

(3) Code :

```
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "woof!"

class Cat(Animal):
    def speak(self):
        return "meow!"

class Cow(Animal):
    def speak(self):
        return "moo!"
```

```
def animal_sound(animal):
    return animal.speak()

# instances
dog = Dog()
cat = Cat()
cow = Cow()

print(animal_sound(dog))
print(animal_sound(cat))
print(animal_sound(cow))
```

In this code, we define a common base class 'Animal' with a method 'speak()'. This method overridden in subclasses 'Dog', 'Cat' and 'Cow'.

The 'Animal_sound()' function takes an 'Animal' object as an argument and calls its 'speak()' method.

When we call 'animal_sound()' with different animal instances, the appropriate 'speak()' method of each subclasses is invoked based on the actual object type. This is dynamic polymorphism, where the method to be called is determined at runtime.

Page No.
Date :

Q.2 Describe the following term in python:

① Function :→

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

Example:

```
def my_function():
    print("Hello!!")

my_function()
```

② Variable :→

Variable in python is containers that store values. The type of variable is determined based on the value it is assigned.

Example:

```
name = "Sahil"
age = 21
height = 5.8
is_student = True

print("Name:", name)
print("Age:", age)
print("Height:", height)
print("Is student:", is_student)
```

Page No.
Date :

③ Data Type: →

Datatypes are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Some built-in data types in python are: Numeric (int, float, complex no.),
Sequence (String, list, Tuple),
Boolean (True, False),
Set, dictionary.

④ module: →

A python module is a file containing python definitions and statements. A module can define functions, classes and variables.

Example:

```
import calc
print (calc.add(10,2))
```

Page No.
Date :

Q.3 Enlist different operators in python and explain with programme.

→ python divides the operators in the following groups:

① Arithmetic operators: →

They are used with numeric value to perform common mathematical operations:

operator	Name	Example
+	Addition	x+y
-	Substraction	x-y
*	multiplication	x*y
/	Division	x/y
%	modulus	x%y
**	Exponentiation	x**y
//	Floor division	x//y

program:

```
val1 = 2
val2 = 3

add = val1 + val2
print(add)                    #5

Sub = val1 - val2
print(sub)                    #-1

mul = val1 * val2
print(mul)                    #6
```

Page No.
Date :

```
div = val1 / val2
print(div)                    # 1.5

mod = val1 % val2
print(mod)                    # 1

expo = val1 ** val2
print(expo)                   # 8

fix = val1 // val2
print(fix)                    # 1
```

(2) Assignment operator:

They are used to assign values to variables

Operator	Example	Same as
=	x=5	x=5
+=	x+= 3	x= x+3
-=	x-= 3	x= x-3
*=	x*= 3	x= x*3
/=	x/= 3	x= x/3
%=	x%= 3	x= x%3
//=	x//= 3	x= x//3
**=	x**= 3	x= x**3
&=	x&= 3	x= x&3
|=	x|= 3	x= x|3
^=	x^= 3	x= x^3
>>=	x>>= 3	x= x>>3
<<=	x<<= 3	x= x<<3

Page No.
Date :

program :

```
a = 10
b = 5
a += b          # a = a+b
print(a)        # 15
```

③ Comparison operators : →

They are used to compare two values.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	less than	x < y
>=	Greater than or equal to	x >= y
<=	less than or equal to	x <= y

program :

```
a = 5
a = 2
print(a>b)      # True
```

④ Logical operator : →

They are used to combine conditional statements.

Operator	Description	Example
and	Returns True if both statement are True	x<5 and x<10
or	Return True if one of statement is True	x<5 or x<4
not	Reverse the result returns false if the result is true	not (x<5 and x<10)

Page No.
Date :

program:

```
a=5
b=6
print ((a>2) and (b>=6))      # True
```

⑤ Identity operator :→

They are used to compare the object, not if they are equal, but if they are actually the same objects, with the same memory location.

Operator	Description	Example
is	Return True if both variable are the same object	x is y
is not	Returns true if both variables are not the same objects	x is not y

program:

```
x2 = 'Hello'
y2 = 'Hello'
print (x2 is y2)      # print True.
```

⑥ Membership Operators :→

They are used to test if a sequence is presented in any object.

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object.	x in y

Page No.
Date :

Not in	Returns True if a sequence with the specified value is not present in object	x not in y.

program:

```
y = {1:'a', 2:'b'}
print('1' in y)      # True.
```

(7) Bitwise operators :→

They act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

Operator	Description	Example
&	Bitwise AND	x & y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x << 2 = 40 (0010 1000)

program:

```
a = 10
b = 14
print(a&b)      # 0
print(a|b)      # 14
print(a~b)      # -11
print(a^b)      # 14
print(a>>b)     # 2
print(a<<b)     # 40
```

Page No.
Date :

Q.4 Construct the python programme to check leap year.

→

```
def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or
    (year % 400 == 0):
        return True
    else:
        return False

year = int(input("Enter a year: "))

if is_leap_year(year):
    print(f"{year} is leap year.")
else:
    print(f"{year} is not leap year.")
```

Q.5 Interpret oop concept in python.

→ Object-Oriented Programming (OOP) is a programming paradigm that is widely used in python and many other programming languages.

* Classes and objects :→

A class is a blueprint or template for creating objects. It defines the structure & behavior of objects of that class.

An object is an instance of a class.

Page No.
Date :

program:

```
class Dog :
    def __init__(self, name) :
        self.name = name
    def bark(self) :
        print(f"{self.name} barks!")
my_dog = Dog("Buddy")
my_dog.bark()          # Buddy barks!
```

* Attributes :→

Attributes are variables that holds data within a class. They defines the characteristics or properties of objects created from the class.

program:

```
class Circle:
    def __init__(self, radius) :
        self.radius = radius
my_circle = Circle(5)
print(my_circle.radius)    # output : 5
```

* Methods :→

Methods are functions defined inside a class that operate on the attributes or perform actions related to the objects behaviour.

program:

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

my_rect = Rectangle(4,6)
print(my_rect.area())          # 24
```

* Inheritance :→

It allows you to create a new class that inherit attributes and methods from an existing class.

program:

```
class Animal:
    def speak(self):
        pass
class Dog(Animal):
    def speak(self):
        return "woof!"

my_dog = Dog()
print(my_dog.speak())          # woof!
```

Page No.
Date :

* Encapsulation: →

It is the concept of hiding the internal details of an object and restricting access to certain parts of it. You can achieve this by using private attribute and defining getter and setter method.

program:

```
class BankAccount:
    def __init__(self, balance):
        self._balance = balance
    def get_balance(self):
        return self._balance
    def deposit(self, amount):
        if amount > 0:
            self._balance += amount

my_account = BankAccount(1000)
print(my_account.get_balance())    # 1000
```

* polymorphism: →

It allows objects of different classes to be treated as objects of a common superclass. It simplifies code and allow for flexibility in handling different types of objects.

program:

```
class shape:
    def area(self):
        pass
```

Page No.
Date :

```
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius * self.radius

Class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

shapes = [Circle(5), Rectangle(4,6)]
for shape in shapes:
    print(f"Area: {shape.area()}")
```

Q.6 Construct the python programme to demonstrate multiple inheritance.

→ multiple inheritance occurs when a class inherits from more than one parent class.

program:

```
class A:
    def method_A(self):
        print("method from class A")

class B:
```

```
    def method_B (self):
        print ("method from class B")

class ((A,B):
    def method_C (self):
        print ("method from class (")

obj_C = C()
obj.C.method_A()
obj_C.method_B()
obj_C.method_C()
```

. Here, we have two parent classes, 'A' & 'B', each with their own methods. The child class 'C' is created by inheriting from both 'A' and 'B'. This is multiple inheritance. 'C' can access and use methods from both 'A' and 'B' in addition to its own methods.