Experiment No.10

AIM: Write python program to implement polymorphism with method overloading and method overriding.

Theory:

1. Method Overloading:

- Method overloading refers to defining multiple methods in a class with the same name but different parameter lists.

- Python does not support method overloading by default like some other languages (e.g., Java or C++), where methods can have different signatures.

- In Python, method overloading can be achieved by using default parameter values or variable length argument lists (\*args or \*\*kwargs).

- When a method is called, Python checks the number and types of arguments passed and executes the corresponding method with matching parameters.

- If no method matches the parameters provided, Python raises a TypeError or calls the method with default parameters if defined.

2. Method Overriding:

- Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.

- The overridden method in the subclass has the same name and signature as the method in the superclass.

- When an object of the subclass calls the overridden method, the subclass's implementation is executed instead of the superclass's implementation.

- Method overriding allows for polymorphism, where objects of different classes can be treated uniformly based on their common superclass.

- It enables the customization of behavior in subclasses while maintaining a consistent interface across related classes.

Here's a summary of the differences between method overloading and method overriding:

- Method Overloading:

- Multiple methods with the same name but different parameters in the same class. - Parameter signature determines which method is called.

- Supported in some languages like Java and C++ with compile-time polymorphism. - Achieved in Python through default parameter values or variable-length argument lists.

- Method Overriding:

- Subclass provides a specific implementation of a method that is already defined in its superclass. - Involves redefining a method with the same name and signature in the subclass. - Subclass's method is called instead of the superclass's method when invoked on objects of the subclass.

- Enables runtime polymorphism and customization of behavior in subclasses.

Program:

class Animal:

def speak(self):

pass

class Dog(Animal):

def speak(self):

return "Woof"

class Cat(Animal):

def speak(self):

return "Meow"

# Method Overloading

class Calculator:

def add(self, a, b, c=None): if c is not None:

return a + b + c

else:

return a + b

# Method Overriding

class Shape:

def area(self):

pass

class Rectangle(Shape):

def \_init\_(self, length, breadth): self.length = length

self.breadth = breadth

def area(self):

return self.length \* self.breadth

class Circle(Shape):

def \_init\_(self, radius):

self.radius = radius

def area(self):

return 3.14 \* self.radius \* self.radius

if \_name\_ == "\_main\_":

print("")

print("Polymorphism with method overriding")

dog = Dog()

cat = Cat()

print(dog.speak()) # Output: Woof

print(cat.speak()) # Output: Meow

print("")

print("Polymorphism with method overloading")

calc = Calculator()

print(calc.add(1, 2)) # Output: 3

print(calc.add(1, 2, 3)) # Output: 6

print("")

print("Polymorphism with method overriding")

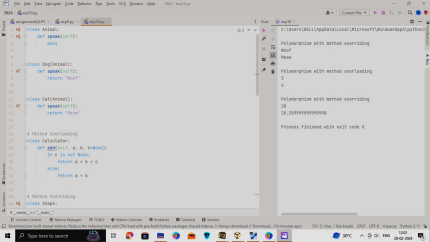
rectangle = Rectangle(5, 4)

circle = Circle(3)

print(rectangle.area())

print(circle.area())

Output:



Experiment No.10

Name:Swatantra Kasliwal

ID:VU4F2223035

Batch: B SE/IT/A

AIM: Write python program to implement polymorphism with method overloading and method overriding.

Theory:

1. Method Overloading:

- Method overloading refers to defining multiple methods in a class with the same name but different parameter lists.

- Python does not support method overloading by default like some other languages (e.g., Java or C++), where methods can have different signatures.

- In Python, method overloading can be achieved by using default parameter values or variable length argument lists (\*args or \*\*kwargs).

- When a method is called, Python checks the number and types of arguments passed and executes the corresponding method with matching parameters.

- If no method matches the parameters provided, Python raises a TypeError or calls the method with default parameters if defined.

2. Method Overriding:

- Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.

- The overridden method in the subclass has the same name and signature as the method in the superclass.

- When an object of the subclass calls the overridden method, the subclass's implementation is executed instead of the superclass's implementation.

- Method overriding allows for polymorphism, where objects of different classes can be treated uniformly based on their common superclass.

- It enables the customization of behavior in subclasses while maintaining a consistent interface across related classes.

Here's a summary of the differences between method overloading and method overriding:

- Method Overloading:

- Multiple methods with the same name but different parameters in the same class. - Parameter signature determines which method is called.

- Supported in some languages like Java and C++ with compile-time polymorphism. - Achieved in Python through default parameter values or variable-length argument lists.

- Method Overriding:

- Subclass provides a specific implementation of a method that is already defined in its superclass. - Involves redefining a method with the same name and signature in the subclass. - Subclass's method is called instead of the superclass's method when invoked on objects of the subclass.

- Enables runtime polymorphism and customization of behavior in subclasses.

Program:

class Animal:

def speak(self):

pass

class Dog(Animal):

def speak(self):

return "Woof"

class Cat(Animal):

def speak(self):

return "Meow"

# Method Overloading

class Calculator:

def add(self, a, b, c=None): if c is not None:

return a + b + c

else:

return a + b

# Method Overriding

class Shape:

def area(self):

pass

class Rectangle(Shape):

def \_init\_(self, length, breadth): self.length = length

self.breadth = breadth

def area(self):

return self.length \* self.breadth

class Circle(Shape):

def \_init\_(self, radius):

self.radius = radius

def area(self):

return 3.14 \* self.radius \* self.radius

if \_name\_ == "\_main\_":

print("")

print("Polymorphism with method overriding")

dog = Dog()

cat = Cat()

print(dog.speak()) # Output: Woof

print(cat.speak()) # Output: Meow

print("")

print("Polymorphism with method overloading")

calc = Calculator()

print(calc.add(1, 2)) # Output: 3

print(calc.add(1, 2, 3)) # Output: 6

print("")

print("Polymorphism with method overriding")

rectangle = Rectangle(5, 4)

circle = Circle(3)

print(rectangle.area())

print(circle.area())

Output:

