<?php // connect to mongodb $m = new MongoClient(); echo "Connection to database successfully"; // select a database $db = $m->examplesdb; echo "Database examplesdb select

Polymorphism

Module 4

What is Polymorphism?

- Polymorphism refers to having several different forms
- It is one of the key features of OOP
- It enables the programmers to assign a different meaning or usage to a variable, function, or an object in different contexts
- While inheritance is related to classes and their hierarchy, polymorphism, on the other hand, is related to methods
- When polymorphism is applied to a function or method depending on the given parameters, a particular form of the function can be selected for execution
- In Python, method overriding is one way of implementing polymorphism

How to use Polymorphism?

Operator Overloading

Operator overloading is the type of overloading in which an operator can be used in multiple ways beyond its predefined meaning

Method Overloading

Method overloading means a class containing multiple methods with the same name but may have different arguments(In python method overloading is not practically possible we can just create a illusion of method overloading)

Method Overriding

Like in other programming languages, the child classes in Python also inherit methods and attributes from the parent class. We can redefine certain methods and attributes specifically to fit the child class, which is known as **Method Overriding.**

- **Example 1: Polymorphism in addition operator**

We know that the + operator is used extensively in Python programs. But, it does not have a single usage.

- **For integer data types, + operator is used to perform arithmetic addition operation**.

```
num1 = 1
num2 = 2
print(num1+num2)
```

- **Similarly, for string data types, + operator is used to perform concatenation**.

```
str1 = "Python"
str2 = "Programming"
print(str1+" "+str2)
```

Polymorphism with Methods:Method Overloading

- **Method Overloading:**
- Two or more methods have the same name but different numbers of parameters or different types of parameters, or both. These methods are called overloaded methods and this is called method **overloading**.
- Like other languages such as java, c++, python does not support method overloading by default. But there are different ways to achieve method overloading in Python.

Example

```
class MyClass:
    def my_method(self, a, b=None, c=None):
        if b is None and c is None:
            print(a)
        elif c is None:
            print(a + b)
        else:
            print(a + b + c)
obj=MyClass()
obj.my_method(10)
obj.my_method(10,20)
obj.my_method(10,10,20)
```

- **Polymorphism with Inheritance:**
- Like in other programming languages, the child classes in Python also inherit methods and attributes from the parent class. We can redefine certain methods and attributes specifically to fit the child class, which is known as **Method Overriding**.
- Polymorphism allows us to access these overridden methods and attributes that have the same name as the parent class.

Method Overriding: Example

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

    def fact(self):
        return "I am a two-dimensional shape."

class Square(Shape):
    def __init__(self, length):
        self.length = length

    def area(self):
        return self.length**2

    def fact(self):
        return "Squares have each angle equal to 90 degrees."


class Circle(Shape):
    def __init__(self, radius):
```

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

    def area(self):
        return 3.14*self.radius**2


a = Square(4)
b = Circle(7)
print(b.fact())
print(a.fact())
print(b.area())
```

Polymorphism with Class Methods

- Python allows different classes to have methods with the same name. Python can use different types of classes in the same way.
- For loops that iterate through multiple objects are created. Next, call the methods without caring about what class each object belongs to. These methods are assumed to exist in every class.

```python
class Cat:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a cat. My name is {self.name}. I am {self.age}
            years old.")

    def make_sound(self):
        print("Meow")


class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a dog. My name is {self.name}. I am {self.age}
            years old.")

    def make_sound(self):
        print("Bark")


cat1 = Cat("Kitty", 2.5)
dog1 = Dog("Fluffy", 4)

for animal in (cat1, dog1):
    animal.make_sound()
    animal.info()
    animal.make_sound()
```