**Module 14. Polymorphism**

**Learning Objectives**

* Understand the concept of polymorphism.
* Learn the function polymorphism, class polymorphism, and inheritance class polymorphism.
* Learn how to use isinstance() method

# **Introduction**

In programming, polymorphism refers to the ability of different objects to respond to the same message (method call) in different ways. This allows for more flexible and modular code, as methods can be written to accept objects of a superclass, and then different subclasses can provide their own implementations of those methods. This concept is fundamental to object-oriented programming (OOP) and is often associated with inheritance and dynamic dispatch. The word "polymorphism" indeed comes from the Greek roots "poly" meaning "many" and "morph" meaning "form", so it literally translates to "many forms".

# **Function Polymorphism**

def main():

    print(len('Python'))

    print(len([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))

    cars = {"manufacturer": "Kia",

            "model": "EV9",

            "price": 65000}

    print(len(cars))

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

    main()

(Output)

6

10

3

Actually, while the len() function can indeed be used to return the number of characters in a string or the number of elements in a list, its behavior isn't limited to just those data types. The len() function is a built-in Python function that can be applied to any object that supports the concept of "length" or "size".

The len() function is designed to return the length (number of items) of an object. The specific way it calculates the length depends on the type of the object it's operating on. This ability to handle different types of objects is what makes len() polymorphic.

The len() function is natively supported for several built-in types:

* **Strings**: Returns the number of characters in the string.
* **Lists**: Returns the number of elements in the list.
* **Tuples**: Returns the number of elements in the tuple.
* **Dictionaries**: Returns the number of key-value pairs.
* **Sets**: Returns the number of elements in the set.

# **Class Polymorphism**

Polymorphism in object-oriented programming allows for methods with the same name to behave differently depending on the object that calls them.

class Car:

    def \_\_init\_\_(self, manufacturer, model, price):

        self.manufacturer = manufacturer

        self.model = model

        self.price = price

    def move(self):

        print("Car moves on roads.")

class Ship:

    def \_\_init\_\_(self, manufacturer, model, price):

        self.manufacturer = manufacturer

        self.model = model

        self.price = price

    def move(self):

        print("Ship moves on water.")

class Airplane:

    def \_\_init\_\_(self, manufacturer, model, price):

        self.manufacturer = manufacturer

        self.model = model

        self.price = price

    def move(self):

        print("Airplane flies in the sky.")

def main():

    car = Car("Ford", "F-150", 50000)

    ship = Ship("Boston Whaler", "Vintage", 250000)

    airplane = Airplane("Boeing", "777", 1000000)

    for x in (car, ship, airplane):

        x.move()

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

    main()

In this example, all three classes—Car, Ship, and Airplane—have a method called move(). However, when you call move() on each object, it executes the appropriate behavior specific to that class. This is polymorphism in action: the same method name (move()) being used across different classes, but with each class providing its own implementation.

(Output)

Car moves on roads.

Ship moves on water.

Airplane flies in the sky.

# **Inheritance Class Polymorphism**

Define a Vehicle parent class and have Car, Ship, and Airplane as its child classes. Each child class inherits methods from the Vehicle class and can override methods as needed.

class Vehicle:

    def \_\_init\_\_(self, manufacturer, model, price):

        self.manufacturer = manufacturer

        self.model = model

        self.price = price

    def move(self):

        print("Move(s).")

class Car(Vehicle):

    def move(self):

        print("Car moves on roads.")

class Ship(Vehicle):

    def move(self):

        print("Ship moves on water.")

class Airplane(Vehicle):

    def move(self):

        print("Airplane flies in the sky.")

def main():

    car = Car("Ford", "F-150", 50000)

    ship = Ship("Boston Whaler", "Vintage", 250000)

    airplane = Airplane("Boeing", "777", 1000000)

    for x in (car, ship, airplane):

        print(x.manufacturer)

        print(x.model)

        print(x.price)

        x.move()

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

    main()

In this example, Car, Ship, and Airplane inherit the move() method from the Vehicle class. However, each child class provides its own implementation of the move() method, which allows for polymorphic behavior when calling move() on objects of these classes.

(Output)

Ford

F-150

50000

Car moves on roads.

Boston Whaler

Vintage

250000

Ship moves on water.

Boeing

777

1000000

Airplane flies in the sky.

# **The isinstance() Function**

The isinstance() function in Python is used to check if an object is an instance of a specified class or if it is an instance of a subclass of that class.

Here's the general syntax:

isinstance(object, classinfo)

* object: The object to be checked
* classinfo: A class or a tuple of classes to be checked against

class Vehicle:

    pass

class Car(Vehicle):

    pass

car\_instance = Car()

print(isinstance(car\_instance, Car))

print(isinstance(car\_instance, Vehicle))

In the above example, car\_instance is an instance of both the Car class and its parent class Vehicle, so isinstance() returns True for both checks.

(Output)

True

True

**Summary**

1. Polymorphism refers to the ability of different objects to respond to the same message (method call) in different ways.
2. len() is a built-in Python function that can be applied to any object that supports the concept of "length" or "size".
3. Polymorphism in object-oriented programming allows for methods with the same name to behave differently depending on the object that calls them.
4. Each subclass inherits methods from the superclass and can override methods as needed.
5. You can use isinstance() function to check if an object is an instance of a specified class or if it is an instance of a subclass of that class.

**Programming Exercises**

**Exercise 1: Shape Area Calculation**

Write a program that calculates the area of different shapes (circle, square, rectangle) using polymorphism. Define a base class Shape with a method calculate\_area(). Then create subclasses for each shape (e.g., Circle, Square, Rectangle) and override the calculate\_area() method in each subclass to calculate the area according to the shape's specific formula.

**Exercise 2: Animal Sounds**

Write a program that simulates different animals making sounds using polymorphism. Define a base class Animal with a method make\_sound(). Then create subclasses for different animals (e.g., dog, cat, bird) and override the make\_sound() method in each subclass to produce the appropriate sound.

**Exercise 3: Employee Payment Calculation**

Write a program that calculates payments for employees based on their roles using polymorphism. Define a base class Employee with a method calculate\_payment(). Then create subclasses for different roles (e.g., Manager, Developer, Intern) and override the calculate\_payment() method in each subclass to calculate payments based on their specific rules (e.g., hourly rate, monthly salary).

**Exercise 4: Vehicle Simulation**

Write a program that simulates different types of vehicles (car, bicycle, motorcycle) using polymorphism. Define a base class Vehicle with methods accelerate() and brake(). Then create subclasses for each type of vehicle and override the accelerate() and brake() methods to simulate the behavior of each vehicle type.

**Exercise 5: isinstance() Function**

Develop a function that behaves differently based on the type of input using isinstance(). Define a function calculate\_area() that calculates the area of a circle if the input is a Circle object and the area of a rectangle if the input is a Rectangle object.