Polymorphism in Python means **“many forms”** — the same function or method behaves differently depending on the context.

**What is Polymorphism in Python?**

Polymorphism allows you to use a **single interface** (like a method or operator) to work with **different types of data** or classes.

Think of + — it **adds numbers**, **joins strings**, and even **combines lists**. That’s **polymorphism**!

**Types of Polymorphism in Python**

**1. Duck Typing**

“If it looks like a duck and quacks like a duck, it’s a duck.”

**Example:**

class Dog:

def speak(self):

print("Woof!")

class Cat:

def speak(self):

print("Meow!")

def animal\_sound(animal):

animal.speak()

dog = Dog()

cat = Cat()

animal\_sound(dog) # Woof!

animal\_sound(cat) # Meow!

The function doesn’t care what type of object it is, as long as it has a speak() method.

**2. Operator Overloading (Built-in Polymorphism)**

**Example:**

print(10 + 5) # Adds numbers

print("Hi " + "There") # Concatenates strings

print([1, 2] + [3]) # Merges lists

The same + operator behaves differently depending on the operands.

**3. Method Overriding (Runtime Polymorphism)**

When a **child class** provides its own version of a method that is already present in the **parent class**.

**Example:**

class Shape:

def draw(self):

print("Drawing a shape")

class Circle(Shape):

def draw(self):

print("Drawing a circle")

obj = Circle()

obj.draw() # Drawing a circle

**4. Function Overloading (default arguments, varargs, @Dispatch)**

**Example:**

def greet(name=None):

if name:

print(f"Hello, {name}!")

else:

print("Hello!")

greet() # Hello!

greet("Nitin") # Hello, Nitin!

**Summary:**

* **Polymorphism = Same action, different behavior**
* **Used in:** functions, classes, operators
* **Main types:** Duck typing, operator overloading, method overriding, simulated overloading