Polymorphism

Polymorphism means “many forms”
Polymorphism is an OOP concept that allows a single interface (method or function) to represent different types of behavior.

In simpler words:
`The same operation can behave differently on different objects.`

1. Same method name, different behavior depending on the object.
2. Polymorphism supports code simplicity and maintainability.

Two main types in Python:

- Compile-time Polymorphism (Method Overloading – not supported in Python)
- Run-time Polymorphism (Method Overriding – widely used in Python)
  1. Function Polymorphism (len(), max())
  2. Operator Polymorphism (+, *)
  3. Method Polymorphism.
  4. Inheritance Polymorphism.

```
# 1. Polymorphism with function.
s = "Python"
l = ["Python"]

print(len(s),len(l),sep=",")
print(max(s),max(l),sep=",")

# O/P:--
6,1                    # Same method name, different behavior depending on the
object
y,Python         # Same method name, different behavior depending on the object
```

```
# 2. Polymorphism with operator.
x = 10
y = 20
print(x+y)
x = '10'
y = '20'
print(x+y)

# O/P:--
```

```
30   # Same method name, different behavior depending on the object
1020 # Same method name, different behavior depending on the object
```

```
# 3. Polymorphism with method.
class Employee:
    def speeks(self):
        return "This is employee"

class Student:
    def speeks(self):
        return "This is student"

for speek in (Employee(), Student()):
    print(speek.sound())

# Output:--
This is employee
This is student
```

```
# 4. Polymorphism with Inheritence and overriding.
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def speak(self):
        print("Dog barks")

class Cat(Animal):
    def speak(self):
        print("Cat meows")

animals = [Dog(), Cat(), Animal()]

for animal in animals:
    animal.speak()  # Same method, different behavior

# Output
Dog barks
Cat meows
Animal speaks
```

```
# Overloading + Operator
class Point:
```

```
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # Overloading +
    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

    def display(self):
        print(f"({self.x}, {self.y})")

p1 = Point(2, 3)
p2 = Point(4, 5)

p3 = p1 + p2   # Calls __add__
p3.display()   # Output: (6, 8)

# Overloading == Operator
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # Overloading ==
    def __eq__(self, other):
        return self.age == other.age

p1 = Person("Alice", 25)
p2 = Person("Bob", 25)
p3 = Person("Charlie", 30)

print(p1 == p2)  # True, because ages are same
print(p1 == p3)  # False
```

Polymorphism Type	Supported in Python?	Runtime or Compile-Time?	Reason
Function Polymorphism (`len()`, `max()`)	✓ Yes	**Runtime**	Behavior depends on the object type, which is decided at runtime
Operator Polymorphism (`+`, `*`)	✓ Yes	**Runtime**	Python dynamically checks operand types during execution
Method Polymorphism (same method name in	✓ Yes	**Runtime**	Method binding and selection happen at

Polymorphism Type	Supported in Python?	Runtime or Compile-Time?	Reason
different classes)			runtime
Inheritance Polymorphism (method overriding)	✓ Yes	**Runtime**	The overridden method in the child class is resolved at runtime
Method Overloading	❌ No	—	Python has no native support for method signature overloading
Compile-Time Polymorphism	❌ No	—	Python does not perform compile-time type checking