Polymorphism in Python – Full Detailed Notes (Advanced + Easy Explanation)

---

1. What is Polymorphism?

Definition:

Polymorphism is a core concept of object-oriented programming where **one function, operator, or method can perform different behaviors based on the object or data type it is working with**.

It allows the same interface to represent different underlying data types. In simple words:

➡️**Same name, different behaviors** depending on the object.

** **Poly = many**, **Morph = forms**

♓Polymorphism means **same function/operation behaving differently depending on the object**.

Python polymorphism types: - **1. Overloading** - Operator Overloading - Method Overloading (NOT supported like Java, but possible using default args / *args) - *Constructor Overloading (NOT supported, only last constructor works) - 2. Overriding - Method Overriding - Constructor Overriding - 3. Duck Typing**

----

2. Operator Overloading

Python allows class objects to use +, -, , >, < *etc. using magic methods** (also called dunder methods).

Operator	Magic Method
+	`__add__(self, other)`
-	`__sub__(self, other)`
*	`__mul__(self, other)`
>	`__gt__(self, other)`
<	`__lt__(self, other)`

Operator	Magic Method
==	`__eq__(self, other)`
str()	`__str__(self)`

Example 1 – + operator overloaded

```
print(5+6)   # integer addition
print("hello" + " world")  # string concatenation
```

Example 2 – Book pages addition

```
class book:
    def __init__(self, page):
        self.pages = page
    def __add__(self, other):
        return self.pages + other.pages

b1 = book(100)
b2 = book(200)
print(b1 + b2)   # 300
```

Example 3 – Compare student ages using > operator

```
class student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __gt__(self, other):
        return self.age > other.age

s1 = student("ajay", 24)
s2 = student("vijay", 22)
print(s1 > s2)  # True
```

Example 4 – Employee salary × time using * operator

```
class employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def __mul__(self, other):
        return self.salary * other.days
```

```
class time:
    def __init__(self, days):
        self.days = days

t1 = time(25)
e1 = employee("ajay", 500)
print("Total amount:", e1 * t1)   # 12500
```

Example 5 – `__str__` for readable output

```
class student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks
    def __str__(self):
        return f"student name is {self.name} object"

s1 = student("ajay", 85)
s2 = student("vijay", 90)
print(s1)
print(s2)
```

Example 6 – Chained addition (important)

```
class student:
    def __init__(self, marks):
        self.marks = marks
    def __add__(self, other):
        return student(self.marks + other.marks)
    def __str__(self):
        return f"total marks are {self.marks}"

s1 = student(85)
s2 = student(90)
s3 = student(175)
print(s1 + s2 + s3)
```

Output:

```
total marks are 350
```

Explanation:

- `s1 + s2` → student(175)
- `175 + s3` → student(350)
- Printed result → **350**

----

3. Method Overloading (Python version)

Python does **NOT** support method overloading like Java or C++.

But we can simulate using **default arguments** or `*args`.

Example using *args

```
class Test:
    def sum(self, *a):
        total = 0
        for x in a:
            total += x
        print("The sum is:", total)

t = Test()
t.sum()
t.sum(10)
t.sum(10, 20)
t.sum(10, 20, 30)
```

----

4. Constructor Overloading (Not supported)

Python keeps only **last constructor**.

But we can use `*args` to imitate.

```python
class Test:
    def __init__(self, *a):
        print("all accepted")

t = Test()
t = Test(12, 34, 45)
```

----

5. Overriding (Dynamic Polymorphism)

Overriding = **child class replacing parent class method**.

Example – Overriding method

```python
class p:
    def proverty(self):
        print("i have a bike")
    def money(self):
        print("i have 1 lakh")

class c(p):
    def money(self):
        super().money()     # call parent method
        print("i have 5 lakh")

o = c()
o.proverty()
o.money()
```

Constructor Overriding

```python
class parent:
    def __init__(self):
        print("parent class constructor")

class child(parent):
    def __init__(self):
        super().__init__()  # parent constructor
        print("child class constructor")
```

```
c = child()
```

----

6. Duck Typing

♎In Python, **if an object has a method, Python will call it** — regardless of its class.

"If it walks like a duck and quacks like a duck, it is a duck"

Example

```
class bird:
    def fly(self):
        print("bird is flying")

class aeroplane:
    def fly(self):
        print("aeroplane is flying")

def func(obj):
    obj.fly()

i = [bird(), aeroplane()]
for obj in i:
    func(obj)
```

✅Final Summary

Concept	Supported in Python?	Notes
Method Overloading	❌No	Can mimic using *args
Constructor Overloading	❌No	Can mimic using *args
Operator Overloading	✅Yes	Using magic methods
Method Overriding	✅Yes	Child overrides parent
Constructor Overriding	✅Yes	Using super()

Concept	Supported in Python?	Notes
Duck Typing	✅Yes	Based on behavior, not class

If you want, I can also create **examples, diagrams, interview questions**, or **MCQs** for this topic!