پلی‌مورفیسم (Polymorphism)

پلی‌مورفیسم یکی از مفاهیم اساسی در **برنامه‌نویسی شی‌گرا** است که به معنی "چندشکلی" است و به ما این امکان را می‌دهد که یک متد یا شیء از چندین نوع مختلف برخوردار باشد. در واقع، پلی‌مورفیسم به این معناست که می‌توانیم یک **واحد کد (مثل متد یا شیء)** را در موقعیت‌های مختلف به گونه‌ای متفاوت اجرا کنیم.

مفهوم پلی‌مورفیسم

پلی‌مورفیسم از دو واژه یونانی به معنای "چند" (پولی) و "شکل" (مورفی) تشکیل شده است. در برنامه‌نویسی، این ویژگی به ما اجازه می‌دهد که از **یک متد با چندین نوع داده** یا **یک شیء از کلاس‌های مختلف** استفاده کنیم، بدون اینکه نیازی به تغییر کد اصلی داشته باشیم.

به عبارت ساده‌تر، پلی‌مورفیسم به این معناست که یک **متد با نام یکسان** می‌تواند **رفتارهای مختلفی** بسته به نوع شیء یا کلاس اجرا کننده از خود نشان دهد.

🔷 نحوه استفاده از پلی‌مورفیسم در شی‌گرایی

در شی‌گرایی، پلی‌مورفیسم بیشتر از طریق **بازنویسی متدها (Method Overriding)** و **وراثت (Inheritance)** به دست می‌آید. به عبارت دیگر، زمانی که چندین کلاس فرزند یک متد مشابه از کلاس والد را بازنویسی می‌کنند، این متدها به طور پلی‌مورفیک عمل می‌کنند و بسته به شیء فراخوانی شده، رفتار متفاوتی خواهند داشت.

تفاوت بین پلی‌مورفیسم در زمان کامپایل و زمان اجرا

🔷 پلی‌مورفیسم در زمان کامپایل (Compile-time Polymorphism)

این نوع پلی‌مورفیسم در هنگام **ترجمه کد (کامپایل)** اتفاق می‌افتد. معمولاً در این نوع، متدها با پارامترهای مختلف (Overloading) تعریف می‌شوند. در این حالت، ترجیحاً به این دلیل که تصمیم‌گیری درباره این که کدام متد فراخوانی شود در زمان کامپایل صورت می‌گیرد، کد ترجمه می‌شود.

🔷 پلی‌مورفیسم در زمان اجرا (Runtime Polymorphism)

در این نوع پلی‌مورفیسم، تصمیم‌گیری درباره اینکه کدام متد باید اجرا شود در **زمان اجرا** (هنگامی که برنامه در حال اجرا است) انجام می‌شود. این نوع معمولاً از طریق **بازنویسی متدها (Overriding)** در کلاس‌های فرزند حاصل می‌شود. به همین دلیل است که به این نوع پلی‌مورفیسم، "پلی‌مورفیسم در زمان اجرا" گفته می‌شود.

مثال‌هایی از پلی‌مورفیسم در زمان اجرا

در اینجا مثالی از **پلی‌مورفیسم در زمان اجرا** را بررسی می‌کنیم، جایی که یک متد مشابه در کلاس‌های فرزند بازنویسی شده است:

```
class Animal:
    def speak(self):
        print("حیوان صدای خاصی تولید می‌کند.")

class Dog(Animal):
    def speak(self):  # بازنویسی متد speak در کلاس Dog
        print("سگ پارس می‌کند.")
```

```
class Cat(Animal):
    def speak(self):  # بازنویسی متد speak در کلاس Cat
        print("گربه میو میو می‌کند.")

# ایجاد اشیاء از کلاس‌های مختلف
animals = [Dog(), Cat()]

# استفاده از پلی‌مورفیسم در زمان اجرا
for animal in animals:
    animal.speak()
```

خروجی:

```
سگ پارس می‌کند.
گربه میو میو می‌کند.
```

در این مثال:

- متد `speak` در هر دو کلاس `Dog` و `Cat` بازنویسی شده است.
- در حلقه `for`، با توجه به نوع شیء، متد مناسب اجرا می‌شود.
- این عملکرد به دلیل پلی‌مورفیسم در زمان اجرا است.

مزایای پلی‌مورفیسم

1. **کاهش پیچیدگی کد:**
   با استفاده از پلی‌مورفیسم، می‌توانیم کد خود را ساده‌تر و خواناتر کنیم. به جای نوشتن کدهای مختلف برای انواع مختلف اشیاء، می‌توانیم از یک متد واحد برای همه استفاده کنیم.
2. **افزایش انعطاف‌پذیری:**
   پلی‌مورفیسم به ما این امکان را می‌دهد که برنامه خود را به‌راحتی گسترش دهیم و متدهای جدیدی اضافه کنیم، بدون اینکه نیاز به تغییر کدهای موجود باشد.
3. **کاهش تکرار کد:**
   با استفاده از پلی‌مورفیسم، می‌توانیم یک کد واحد بنویسیم که با انواع مختلف اشیاء کار کند و از تکرار کد جلوگیری کنیم.

پلی‌مورفیسم در زمان کامپایل (Overloading)

در پایتون، پلی‌مورفیسم در زمان کامپایل به صورت **Overloading** متدها به طور مستقیم پشتیبانی نمی‌شود. این ویژگی بیشتر در زبان‌هایی مانند **C++** و **Java** موجود است. در پایتون، شما نمی‌توانید متدی با نام مشابه اما با تعداد مختلف پارامترها تعریف کنید. در عوض، می‌توانید از **آرگومان‌های پیش‌فرض** یا **آرگومان‌های متغیر** استفاده کنید.

مثال پلی‌مورفیسم در زمان کامپایل (با استفاده از آرگومان‌های پیش‌فرض)

```
class Printer:
    def print_message(self, message="پیام پیش‌فرض"):
        print(message)

printer = Printer()
printer.print_message()  # استفاده از پیام پیش‌فرض
printer.print_message("پیام جدید")  # استفاده از پیام جدید
```

خروجی:

```
پیام پیش‌فرض
پیام جدید
```

در این مثال، از **آرگومان پیش‌فرض** برای پیاده‌سازی پلی‌مورفیسم در زمان کامپایل استفاده شده است.

 تمرین برای شما:

یک کلاس `Shape` با یک متد `area` بسازید که در کلاس‌های `Circle` و `Rectangle` بازنویسی شود. سپس از پلی‌مورفیسم استفاده کنید تا مساحت هر شکل را محاسبه کنید بدون اینکه نیازی به بررسی نوع شیء باشد.