استفاده از پلی‌مورفیسم در پایتون

پلی‌مورفیسم در پایتون به این معناست که یک **متد واحد** می‌تواند با **انواع مختلف داده‌ها یا اشیاء** برخورد متفاوتی داشته باشد. این قابلیت به ما این امکان را می‌دهد که از یک رابط واحد برای تعامل با اشیاء مختلف استفاده کنیم و بسته به نوع شیء، رفتارهای متفاوتی اجرا شود.

پایتون از **وراثت** و **بازنویسی متدها** (Overriding) برای پیاده‌سازی پلی‌مورفیسم استفاده می‌کند. این به این معناست که یک متد می‌تواند در کلاس‌های مختلف رفتارهای متفاوتی داشته باشد، حتی اگر نام یکسانی داشته باشد.

نحوه پیاده‌سازی پلی‌مورفیسم با استفاده از متدهای مشترک در کلاس‌های مختلف

در پایتون، اگر چند کلاس مختلف دارای یک متد با نام یکسان باشند، می‌توان از این متد در شرایط مختلف استفاده کرد، و بسته به نوع شیء فراخوانی‌شده، متد مناسب اجرا می‌شود. این یکی از ویژگی‌های پلی‌مورفیسم است که به ما این امکان را می‌دهد که یک متد مشترک در کلاس‌های مختلف داشته باشیم.

مثال پیاده‌سازی پلی‌مورفیسم با متدهای مشترک در کلاس‌های مختلف

در این مثال، دو کلاس `Dog` و `Cat` داریم که هر دو متد `speak` را دارند، اما هر کدام رفتار متفاوتی را پیاده‌سازی می‌کنند:

```
class Dog:
    def speak(self):
        print("سگ پارس می‌کند.")

class Cat:
    def speak(self):
        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()  # بسته به نوع شیء، متد مناسب اجرا می‌شود
```

خروجی:

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

در اینجا:

- متد `speak` در هر دو کلاس `Dog` و `Cat` تعریف شده است.
- در هنگام اجرای حلقه، پایتون بسته به نوع شیء (سگ یا گربه) متد مناسب را فراخوانی می‌کند.
- این رفتار به دلیل پلی‌مورفیسم در زمان اجرا است.

مثال‌های کاربردی از استفاده از پلی‌مورفیسم

استفاده از پلی‌مورفیسم برای محاسبه مساحت اشکال مختلف

در این مثال، یک کلاس `Shape` داریم که متد `area` را به صورت پایه‌ای تعریف کرده‌ایم. سپس، این متد در کلاس‌های `Rectangle` و `Circle` بازنویسی می‌شود تا محاسبه مساحت خاص هر شکل را انجام دهد.

```
import math

class Shape:
    def area(self):
        pass  # متد پایه که باید در کلاس‌های فرزند بازنویسی شود

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):  # بازنویسی متد area
        return math.pi * self.radius ** 2

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):  # بازنویسی متد area
        return self.width * self.height

# ایجاد اشیاء از کلاس‌های مختلف
shapes = [Circle(5), Rectangle(4, 6)]

# استفاده از پلی‌مورفیسم برای محاسبه مساحت
for shape in shapes:
    print(f"مساحت: {shape.area()}")
```

خروجی:

```
مساحت: 78.53981633974483
مساحت: 24
```

در این مثال:

- متد `area` در کلاس‌های `Circle` و `Rectangle` به‌طور متفاوت بازنویسی شده است.
- با استفاده از پلی‌مورفیسم، می‌توانیم با استفاده از یک حلقه، به طور یکسان با هر دو نوع شیء تعامل داشته باشیم و مساحت آن‌ها را محاسبه کنیم.

ایجاد کدهایی که بسته به نوع شیء رفتارهای متفاوتی دارند

گاهی اوقات می‌خواهیم که رفتار برنامه بسته به نوع شیء متفاوت باشد. برای این منظور، می‌توان از پلی‌مورفیسم به همراه چک کردن نوع شیء استفاده کرد.

مثال: رفتار متفاوت برای اشیاء مختلف

در این مثال، از متد `speak` برای اشیاء مختلف استفاده می‌کنیم و رفتار آن‌ها را به صورت متفاوت نشان می‌دهیم.

```
class Dog:
    def speak(self):
        return "Woof!"

class Cat:
    def speak(self):
        return "Meow!"

class Bird:
    def speak(self):
        return "Chirp!"

def animal_sound(animal):
    print(f"صدای حیوان: {animal.speak()}")

# ایجاد اشیاء از کلاس‌های مختلف
dog = Dog()
cat = Cat()
bird = Bird()

# استفاده از پلی‌مورفیسم برای فراخوانی متد speak در اشیاء مختلف
animal_sound(dog)  # خروجی: صدای حیوان: Woof!
animal_sound(cat)  # خروجی: صدای حیوان: Meow!
animal_sound(bird)  # خروجی: صدای حیوان: Chirp!
```

در این مثال:

- با استفاده از یک متد عمومی به نام `animal_sound`، رفتارهای متفاوت برای اشیاء مختلف فراخوانی می‌شود.
- بسته به نوع شیء، متد مناسب `speak` اجرا می‌شود و صدای متفاوتی تولید می‌کند.
- این پیاده‌سازی از پلی‌مورفیسم برای ایجاد رفتارهای متفاوت بسته به نوع شیء استفاده کرده است.

مزایای استفاده از پلی‌مورفیسم

1. **انعطاف‌پذیری بیشتر در طراحی:**
   با استفاده از پلی‌مورفیسم، می‌توانیم برنامه‌های انعطاف‌پذیرتری بنویسیم که به راحتی می‌توان آن‌ها را گسترش داد و رفتارهای جدیدی را به کلاس‌ها اضافه کرد.
2. **کاهش تکرار کد:**
   پلی‌مورفیسم اجازه می‌دهد که کدهای مشترک را در یک متد واحد تعریف کنیم و آن را در کلاس‌های مختلف استفاده کنیم، که این باعث کاهش تکرار کد می‌شود.
3. **خوانایی و قابلیت نگهداری بهتر:**
   با استفاده از پلی‌مورفیسم، برنامه‌نویسی به شیوه‌ای ساده‌تر و خواناتر می‌شود زیرا رفتارهای متفاوت در داخل یک متد تعریف می‌شود.

📝 تمرین برای شما:

یک کلاس `Employee` ایجاد کنید که یک متد `get_salary` داشته باشد. سپس دو کلاس `Manager` و `Developer` ایجاد کنید که این متد را بازنویسی کرده و مبلغ حقوق خاص خود را محاسبه کنند. از پلی‌مورفیسم برای نمایش حقوق هر کدام استفاده کنید.