Introdução à Programação Orientada a Objetos (POO) - Parte 2 (Herança e Polimorfismo) 🐍

Na aula de hoje, vamos continuar nossa jornada na **Programação Orientada a Objetos (POO)**, explorando dois pilares fundamentais: **Herança** e **Polimorfismo**. Esses conceitos são ferramentas poderosas para criar sistemas mais organizados, reutilizáveis e flexíveis. Vamos lá! 🚀

1. O que é Herança? 👨‍👧

Herança é um mecanismo que permite que uma classe (chamada de **classe filha**) herde atributos e métodos de outra classe (chamada de **classe pai**). Isso promove **reutilização de código** e organiza melhor as funcionalidades.

Imagine uma classe geral chamada `Animal`, e classes específicas como `Cachorro` e `Gato`. Ambas podem herdar comportamentos comuns de `Animal`.

```python
# Classe pai: Animal
class Animal:
    def __init__(self, nome):
        self.nome = nome

    def comer(self):
        print(f"{self.nome} está comendo.")

# Classe filha Cachorro herdando de Animal
class Cachorro(Animal):
    def latir(self):
        print(f"{self.nome} está latindo!")

rex = Cachorro("Rex")
rex.comer()  # Saída: Rex está comendo.
rex.latir()  # Saída: Rex está latindo!
```

2. Construtores em Classes Filhas 🏗️

Quando uma classe filha herda de uma classe pai, o método `__init__()` da classe pai não é automaticamente chamado. Se quiser usá-lo, você precisa chamá-lo explicitamente com `super()`.

```
class Animal:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

class Cachorro(Animal):
    def __init__(self, nome, idade, raca):
        super().__init__(nome, idade)  # Chama o construtor da classe pai
        self.raca = raca

# Agora, a classe filha Cachorro tem atributos próprios e herdados.
rex = Cachorro("Rex", 5, "Labrador")
print(rex.nome)  # Saída: Rex
print(rex.raca)  # Saída: Labrador
```

3. O que é Polimorfismo? 🎭

Polimorfismo é a habilidade de diferentes classes terem métodos com o mesmo nome, mas comportamentos diferentes. Isso permite que o mesmo código funcione com objetos de diferentes tipos.

```
class Cachorro(Animal):
    def fazer_som(self):
        print(f"{self.nome} faz: Au Au!")

class Gato(Animal):
    def fazer_som(self):
        print(f"{self.nome} faz: Miau!")

rex = Cachorro("Rex")
mingau = Gato("Mingau")

rex.fazer_som()     # Rex faz: Au Au!
mingau.fazer_som()  # Mingau faz: Miau!
```

4. Sobrescrita de Métodos (Overriding) 🔄

Sobrescrita permite que uma classe filha reimplemente um método da classe pai, alterando o comportamento para atender a necessidades específicas.

```
class Animal:
    def mover(self):
        print("O animal está se movendo.")

class Passaro(Animal):
    def mover(self):
        print("O pássaro está voando.")

beija_flor = Passaro()
beija_flor.mover()  # Saída: O pássaro está voando.
```

Conclusão 🏁

Hoje aprendemos dois pilares fundamentais da POO: **Herança** e **Polimorfismo**. Esses conceitos permitem criar códigos reutilizáveis, organizados e adaptáveis a diferentes contextos.

- **Herança**: Permite que uma classe filha reutilize atributos e métodos de uma classe pai.
- **Construtores** em Classes Filhas: Utilizamos `super()` para chamar o construtor da classe pai.
- **Polimorfismo**: Permite que diferentes classes tenham métodos com o mesmo nome e comportamentos distintos.
- **Sobrescrita de Métodos**: Adapta métodos herdados para necessidades específicas.