**Tutorial básico sobre Programação Orientada a Objetos (OO) em Python**

## 1. Conceitos Básicos de OO

Antes de começar com Python, é importante entender alguns conceitos fundamentais da Programação Orientada a Objetos:

- **Classe**: Um modelo ou molde que define a estrutura e o comportamento dos objetos.
- **Objeto**: Uma instância de uma classe. É um elemento concreto baseado na classe.
- **Atributo**: Características ou propriedades de uma classe. São variáveis dentro de uma classe.
- **Método**: Funções definidas dentro de uma classe que descrevem o comportamento dos objetos.
- **Herança**: Um mecanismo onde uma classe pode herdar atributos e métodos de outra classe.
- **Encapsulamento**: O ato de esconder detalhes internos de uma classe, expondo apenas o necessário.
- **Polimorfismo**: A capacidade de diferentes classes responderem a métodos da mesma maneira.

## 2. Criando uma Classe em Python

```
class Animal:
    def __init__(self, nome, especie):
        self.nome = nome  # Atributo de instância
        self.especie = especie  # Atributo de instância

    def emitir_som(self):
        print(f"{self.nome} faz um som!")
```

- **__init__**: É o método construtor da classe. Ele é chamado automaticamente quando um novo objeto é criado.
- **self**: Refere-se à instância atual da classe e é usada para acessar atributos e métodos dentro da classe.

## 3. Criando um Objeto

```
cachorro = Animal("Rex", "Cachorro")
gato = Animal("Miau", "Gato")

print(cachorro.nome)  # Saída: Rex
cachorro.emitir_som()  # Saída: Rex faz um som!
```

## 4. Herança

```
class Cachorro(Animal):
    def emitir_som(self):
        print(f"{self.nome} late!")

cachorro = Cachorro("Rex", "Cachorro")
cachorro.emitir_som()  # Saída: Rex late!
```

**5. Polimorfismo**

```
class Gato(Animal):
    def emitir_som(self):
        print(f"{self.nome} mia!")

animais = [Cachorro("Rex", "Cachorro"), Gato("Miau", "Gato")]

for animal in animais:
    animal.emitir_som()
```

- Aqui, **emitir_som** é chamado para diferentes tipos de objetos (Cachorro e Gato), mas eles respondem de maneiras diferentes.

**6. Encapsulamento**

```
class ContaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.__saldo = saldo  # Atributo privado

    def depositar(self, valor):
        self.__saldo += valor

    def sacar(self, valor):
        if valor <= self.__saldo:
            self.__saldo -= valor
        else:
            print("Saldo insuficiente!")

    def obter_saldo(self):
        return self.__saldo

conta = ContaBancaria("João", 1000)
conta.depositar(500)
print(conta.titular)
#print(conta.__saldo) # O acesso direto gera um erro
print(conta.obter_saldo())  # Saída: 1500
```

## Conceitos Fundamentais da Orientação a Objetos

**1. Classe**

- **O que é**: Uma classe é como um molde ou uma planta de um objeto. Ela define as propriedades (atributos) e comportamentos (métodos) que os objetos criados a partir dessa classe terão.
- **Exemplo**: Se você tem uma classe chamada **Carro**, ela pode ter *atributos* como **cor**, **modelo** e *métodos* como **acelerar** e **frear**.

2. **Objeto**
    - **O que é**: Um objeto é uma instância de uma classe, ou seja, é uma versão concreta da classe. Enquanto a classe é uma definição, o objeto é algo que você pode realmente usar e manipular no código.
    - **Exemplo:** Se **Carro** é uma classe, então **meu_carro = Carro()** cria um objeto **meu_carro** a partir da classe **Carro**.

3. **Atributo**
    - **O que é**: Atributos são as variáveis definidas dentro de uma classe. Eles representam as características ou propriedades dos objetos.
    - **Exemplo**: Na classe **Carro**, o atributo **cor** pode ter o valor **"vermelho"**.

4. **Método**
    - **O que é**: Métodos são as funções definidas dentro de uma classe. Eles representam as ações ou comportamentos que os objetos da classe podem realizar.
    - **Exemplo**: O método **acelerar** em um objeto **Carro** pode aumentar a velocidade do carro.

5. **Encapsulamento**
    - **O que é**: Encapsulamento é o conceito de esconder os detalhes internos de uma classe, expondo apenas o que é necessário. Ele protege os dados e a lógica dentro da classe, permitindo o acesso apenas por meio de métodos controlados.
    - **Exemplo**: Em uma classe **ContaBancaria**, você pode ter um atributo privado **__saldo** que só pode ser acessado ou modificado por meio de métodos como **depositar** ou **sacar**. Isso impede acesso direto ao saldo fora da classe, protegendo os dados.

6. **Herança**
    - **O que é**: Herança é o conceito onde uma classe (subclasse) pode herdar atributos e métodos de outra classe (superclasse). Isso promove a reutilização de código e a criação de hierarquias.
    - **Exemplo:** Se você tem uma classe **Veiculo**, você pode criar uma classe **Carro** que herda de **Veiculo**. Assim, **Carro** terá todos os *atributo*s e *métodos* de **Veiculo**, além de suas próprias características específicas.

7. **Polimorfismo**
    - **O que é**: Polimorfismo permite que objetos de diferentes classes possam ser tratados como objetos da mesma classe base, desde que compartilhem a mesma interface. Isso significa que o mesmo método pode ter diferentes comportamentos dependendo da classe do objeto que o chama.
    - **Exemplo**: Suponha que tanto a classe **Cachorro** quanto **Gat**o herdam de **Animal** e ambas implementam o método **emitir_som**. Quando você chama **emitir_som** em um objeto **Cachorro**, ele pode "latir", e em um objeto **Gato**, ele pode "miar". Apesar de o método ter o mesmo nome, o comportamento é diferente dependendo do objeto.

8. **Abstração**
    - **O que é**: Abstração é o conceito de esconder os detalhes complexos de implementação e expor apenas as funcionalidades essenciais. Ela permite que o desenvolvedor se concentre no que o objeto faz, e não em como ele faz.
    - **Exemplo**: Quando você dirige um carro, você não precisa saber como o motor funciona internamente; você apenas usa os controles (volante, pedais) para dirigir.

**Resumo**

- **Encapsulamento** protege os dados internos de uma classe.
- **Herança** permite que classes compartilhem atributos e métodos.
- **Polimorfismo** possibilita que diferentes objetos respondem de maneiras específicas ao mesmo método.
- **Abstração** foca em esconder a complexidade e mostrar apenas o necessário.