Programação Orientada a Objetos

(Python)

Feito por: Yann Ricardo Teixeira Xavier
Análise De Sistemas
Uninassau Paulista
Data: 02/04/2024

Introdução à Programação Orientada a Objetos (POO)

A Programação Orientada a Objetos (POO) é um paradigma de programação amplamente utilizado em que os programas são estruturados em torno de objetos. Python é uma linguagem que suporta completamente esse paradigma e fornece recursos poderosos para criar e manipular objetos. Nesta apostila, exploraremos os conceitos fundamentais da POO em Python, desde a definição de classes e objetos até herança, polimorfismo e encapsulamento.

1. Classes e Objetos

Classes e objetos são conceitos fundamentais em POO. Uma classe é uma estrutura que define o comportamento e as propriedades de um objeto, enquanto um objeto é uma instância específica de uma classe.

Exemplo:

python Copy code

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

    def apresentar(self):
        print(f"Olá, meu nome é {self.nome} e tenho {self.idade} anos.")

# Criando um objeto Pessoa
pessoa1 = Pessoa("João", 30)
pessoa1.apresentar()
```

2. Encapsulamento

Encapsulamento é o princípio de esconder os detalhes de implementação de um objeto e expor apenas a interface necessária para interagir com ele. Em Python, podemos controlar o acesso aos atributos de uma classe usando métodos especiais e atributos privados.

Exemplo:

python Copy code

```python
class ContaBancaria:
    def __init__(self, saldo):
        self.__saldo = saldo

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

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

    def get_saldo(self):
        return self.__saldo

# Criando um objeto ContaBancaria
conta = ContaBancaria(1000)
conta.depositar(500)
conta.sacar(200)
print("Saldo atual:", conta.get_saldo())
```

3. Herança

Herança é um conceito que permite criar uma nova classe com base em uma classe existente. A classe nova, chamada de classe filha ou subclasse, herda os atributos e métodos da classe pai ou superclasse.

Exemplo:

python Copy code

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

    def fazer_som(self):
        pass

class Cachorro(Animal):
    def fazer_som(self):
        return "Au au!"

class Gato(Animal):
    def fazer_som(self):
        return "Miau!"

# Criando objetos Cachorro e Gato
cachorro = Cachorro("Rex")
gato = Gato("Garfield")

print(cachorro.nome, "faz", cachorro.fazer_som())
print(gato.nome, "faz", gato.fazer_som())
```

4. Polimorfismo

Polimorfismo é a capacidade de objetos de diferentes classes responderem ao mesmo método de maneiras diferentes. Isso nos permite tratar objetos de diferentes classes de maneira uniforme.

Exemplo:

python Copy code

```python
class Animal:
    def fazer_som(self):
        pass

class Cachorro(Animal):
    def fazer_som(self):
        return "Au au!"

class Gato(Animal):
    def fazer_som(self):
        return "Miau!"

# Função polimórfica
def emitir_som(animal):
    print(animal.fazer_som())

# Criando objetos Cachorro e Gato
cachorro = Cachorro()
gato = Gato()

# Chamando a função polimórfica
emitir_som(cachorro)
emitir_som(gato)
```

5. Abstração

Abstração é o processo de simplificar complexidades, representando os recursos mais importantes sem incluir detalhes desnecessários. Em Python, podemos usar classes abstratas para definir métodos que devem ser implementados por suas subclasses.

python Copy code

```python
from abc import ABC, abstractmethod

class Forma(ABC):
    @abstractmethod
    def calcular_area(self):
        pass

class Retangulo(Forma):
    def __init__(self, altura, largura):
        self.altura = altura
        self.largura = largura

    def calcular_area(self):
        return self.altura * self.largura

# Criando um objeto Retangulo
retangulo = Retangulo(5, 10)
print("Área do retângulo:", retangulo.calcular_area())
```

6. Associação, Agregação e Composição

Associação, agregação e composição são tipos de relacionamentos entre objetos. Associação refere-se a um relacionamento simples entre objetos. Agregação é uma forma mais fraca de associação, onde um objeto pode existir independentemente do outro. Composição é uma forma forte de agregação, onde um objeto é composto por outros objetos.

Exemplo:

python Copy code

```python
class Carrinho:
    def __init__(self):
        self.produtos = []

    def adicionar_produto(self, produto):
        self.produtos.append(produto)

class Produto:
    def __init__(self, nome, preco):
        self.nome = nome
        self.preco = preco

# Criando objetos Carrinho e Produto
carrinho = Carrinho()
produto1 = Produto("Camisa", 30)
produto2 = Produto("Calça", 50)

carrinho.adicionar_produto(produto1)
carrinho.adicionar_produto(produto2)

for produto in carrinho.produtos:
    print(produto.nome, "-", produto.preco)
```

7. Exceções em POO

Tratamento de exceções em POO envolve lidar com erros que ocorrem durante a execução de métodos e operações de objetos. Em Python, podemos usar blocos `try`, `except` e `finally` para capturar e lidar com exceções.

Exemplo:

python Copy code

```python
class DivisaoPorZeroError(Exception):
    pass

class Calculadora:
    def dividir(self, a, b):
        try:
            if b == 0:
                raise DivisaoPorZeroError("Divisão por zero não é permitida.")
            resultado = a / b
            return resultado
        except DivisaoPorZeroError as e:
            print(e)
        except Exception as e:
            print("Ocorreu um erro:", e)

# Criando um objeto Calculadora
calculadora = Calculadora()

# Testando o método dividir
print(calculadora.dividir(10, 2))
print(calculadora.dividir(10, 0))
```

8. Princípios SOLID em POO

Os princípios SOLID são um conjunto de cinco princípios de design de software que visam tornar o código mais fácil de entender, manter e estender. Eles incluem Single Responsibility Principle (SRP), Open/Closed Principle (OCP), Liskov Substitution Principle (LSP), Interface Segregation Principle (ISP) e Dependency Inversion Principle (DIP).

Exemplo:

pitão Copy code

```
# Exemplo de aplicação dos princípios SOLID em Python
# (implementação de cada princípio pode ser feita em exemplos anteriores)
```

9. Design Patterns em POO

Design patterns são soluções típicas para problemas recorrentes em design de software. Existem diversos padrões de projeto, cada um com uma finalidade específica. Alguns exemplos incluem Singleton, Factory, Observer, Strategy, entre outros.

Exemplo:

pitão Copy code

```
# Implementação de um padrão de projeto (ex: Singleton, Factory, etc.)
```

10. Práticas Recomendadas e Considerações Finais

Práticas recomendadas incluem seguir convenções de nomenclatura, escrever código claro e legível, documentar o código adequadamente e realizar testes de unidade. Com uma compreensão sólida dos conceitos de POO em Python, você estará pronto para desenvolver programas mais robustos, escaláveis e modularizados.

Aqui estão 10 exercícios de Programação Orientada a Objetos (POO) em Python:

01º) Crie uma classe Pessoa com os atributos nome e idade. Em seguida, crie objetos dessa classe e imprima suas informações.

02º) Crie uma classe Retângulo com os atributos comprimento e largura. Implemente métodos para calcular a área e o perímetro do retângulo.

03º) Crie uma classe Conta Bancária com os atributos saldo e titular. Implemente métodos para depositar, sacar e verificar saldo.

04º) Crie uma classe Carro com os atributos marca, modelo e ano. Implemente métodos para acelerar, frear e imprimir as informações do carro.

05º) Crie uma classe Produto com os atributos nome, preço e quantidade em estoque. Implemente métodos para aumentar e diminuir a quantidade em estoque.

06º) Crie uma classe Triângulo com os atributos lado1, lado2 e lado3. Implemente um método para verificar se é um triângulo válido e outro método para calcular a área.

07º) Crie uma classe Conta Poupança que herda da classe Conta Bancária. Adicione um método para calcular juros sobre o saldo.

08º) Crie uma classe Livro com os atributos título, autor e número de páginas. Implemente um método para verificar se o livro é longo (mais de 300 páginas).

09º) Crie uma classe Animal com os atributos nome e som. Em seguida, crie subclasses dessa classe para diferentes tipos de animais (por exemplo, Cachorro, Gato, Pássaro) e implemente métodos para fazer o animal emitir seu som.

10º) Crie uma classe Estudante com os atributos nome, idade e notas. Implemente um método para calcular a média das notas do estudante.