Aula 06

Programação Orientada a Objetos

A programação orientada a objetos (POO) é um paradigma de programação que permite organizar o código de forma modular e reutilizável, utilizando conceitos como classes, objetos, herança e polimorfismo. Python é uma linguagem de programação que suporta POO e fornece recursos poderosos para implementar esse estilo de programação de maneira eficiente e eficaz.

Abaixo temos alguns conceitos de POO:

Classes

- Uma classe é uma estrutura que define um conjunto de atributos (variáveis) e métodos (funções) que descrevem um objeto em particular. Uma classe é como um plano ou uma planta baixa que define as características e comportamentos de um objeto;

Objetos

- Um objeto é uma instância de uma classe. Ele representa um exemplo específico da classe e possui seus próprios valores para os atributos da classe. Podemos criar vários objetos de uma mesma classe, cada um com seu próprio estado e comportamento;

Atributos

- Atributos são as variáveis associadas a uma classe. Eles podem ser variáveis de instância, que são específicas de cada objeto, ou variáveis de classe, que são compartilhadas por todos os objetos da classe;

Métodos

- Métodos são as funções associadas a uma classe. Eles definem o comportamento dos objetos da classe. Existem métodos de instância, que operam em instâncias específicas da classe, e métodos de classe, que operam em atributos e objetos compartilhados pela classe;

Adiante vamos ver outros conceitos da POO. Por hora, vamos nos aprofundar nesses conceitos um a um.

Classe

Uma classe é uma estrutura fundamental na programação orientada a objetos (POO) que define um conjunto de atributos e métodos que descrevem um objeto em particular. Pode-se pensar em uma classe como um plano ou uma planta baixa que especifica as características e comportamentos de um objeto. Ela serve como um modelo que define a estrutura e o comportamento dos objetos que serão criados a partir dela.

Em POO, uma classe é um conceito abstrato, uma espécie de "molde" a partir do qual os objetos são criados. Ela encapsula dados e funcionalidades relacionadas em uma única unidade. Os atributos da classe representam as características ou propriedades que os objetos possuirão, enquanto os métodos definem as ações ou comportamentos que os objetos podem realizar.

Para criar um objeto a partir de uma classe, utiliza-se um processo chamado instanciação. A instanciação ocorre quando se cria uma instância (ou objeto) da classe. Cada objeto criado a partir da classe possui seu próprio conjunto de atributos e pode executar os métodos definidos pela classe.

Objeto

Um objeto é uma instância de uma classe na programação orientada a objetos (POO). Pode-se pensar em um objeto como uma entidade concreta ou uma entidade do mundo real que possui um estado (atributos) e comportamento (métodos) específicos, definidos pela classe da qual ele é uma instância.

Em POO, uma classe serve como um modelo ou um plano que define as características e comportamentos que um objeto específico pode ter. Uma vez que a classe é definida, podemos criar objetos a partir dela usando um processo chamado instanciação. A instanciação é o ato de criar uma instância ou um objeto específico da classe.

```python
class Pessoa:
    """Definição da classe Pessoa"""

# instanciando o objeto p1 a partir da classe Pessoa
p1 = Pessoa()
# instanciando um novo objeto e com anotações de tipo
p2:Pessoa = Pessoa()
```

Atributos

Os atributos são elementos fundamentais em programação orientada a objetos (POO) que permitem armazenar e representar dados relacionados a uma classe e aos objetos criados a partir dela. Tanto a classe quanto os objetos possuem atributos, embora existam algumas diferenças importantes entre eles.

Por hora, vamos ver apenas os atributos de instância.

Atributos de Instância

- Os atributos de instância são específicos de cada objeto individual criado a partir da classe;
- Eles são declarados dentro do método especial __init__() da classe, que é executado automaticamente quando um objeto é criado;
- Cada objeto possui uma cópia independente dos atributos de instância, com valores exclusivos para cada objeto;
- Os atributos de instância são acessíveis apenas pelos objetos, usando a notação de ponto;
- Os atributos de instância permitem armazenar informações exclusivas para cada objeto, personalizando suas características;

```python
class Pessoa:
    """Definição da classe Pessoa"""

    def __init__(self, p_nome: str, p_idade: int) -> None:
        self.nome: str = p_nome
        self.idade: int = p_idade

if __name__ == '__main__':
    # instanciando o objeto p1 passando os parâmetros
    p1 = Pessoa('Tom Cruise', 60)
    print(f'Nome: {p1.nome}, Idade: {p1.idade}')
    # instanciando um novo objeto e com anotações de tipo
    p2: Pessoa = Pessoa('Arnold Schwarzenegger', 75)
    print(f'Nome: {p2.nome}, Idade: {p2.idade}')
```

Nesse exemplo, a classe Pessoa possui o método __init__() que é chamado automaticamente durante a instanciação dos objetos. Dentro desse método, os atributos de instância nome e idade são declarados e inicializados com os valores fornecidos durante a criação dos objetos p1 e p2.

Cada objeto possui sua própria cópia desses atributos, permitindo que eles armazenem informações exclusivas. Portanto, ao acessar os atributos de instância usando a notação de ponto (p.nome), obtemos os valores específicos associados a cada objeto.

Método __init__()

O método __init__() é um método especial em Python que é chamado automaticamente quando um objeto de uma classe é criado. Ele desempenha um papel fundamental na inicialização dos atributos de instância do objeto.

A palavra-chave __init__ é reservada em Python para indicar que esse é o método de inicialização da classe. Ele é chamado imediatamente após a instanciação de um objeto e permite configurar o estado inicial do objeto definindo os valores iniciais de seus atributos.

Aqui está como o método __init__() funciona:

- A classe é definida com o método __init__() dentro dela;
- Durante a criação de um objeto a partir da classe (instanciação), o método __init__() é chamado automaticamente;
- O método __init__() recebe a referência self como seu primeiro parâmetro. Essa referência representa o objeto que está sendo criado;
- Além do parâmetro self, o método __init__() pode receber outros parâmetros que serão passados durante a criação do objeto. Esses parâmetros são usados para fornecer valores iniciais para os atributos de instância do objeto;
- Dentro do método __init__() são definidos os atributos de instância do objeto, utilizando a notação self.nome_do_atributo = valor;
- Após a execução do método __init__(), o objeto é criado com os atributos de instância devidamente inicializados;

Nesse exemplo acima, a classe Pessoa possui o método __init__() que recebe os parâmetros nome e idade. Dentro do método __init__(), esses parâmetros são utilizados para inicializar os atributos de instância nome e idade do objeto.

Ao criar o objeto p1 a partir da classe Pessoa e passar os valores "Tom Cruise" e 60 como argumentos, o método __init__() é automaticamente chamado. Ele atribui o valor " Tom Cruise " ao atributo nome e o valor 60 ao atributo idade do objeto p1.

Assim, o método __init__() permite que você configure o estado inicial dos objetos de uma classe, definindo os valores iniciais de seus atributos de instância. Ele é amplamente utilizado para realizar tarefas de inicialização, como atribuir valores padrão aos atributos ou configurar o objeto com base nos parâmetros fornecidos durante a criação do objeto.

Também podemos definir valores padrão para atributos dentro do método __init__()

```python
class Pessoa:
    """Definição da classe Pessoa"""

    def __init__(self, p_nome: str, p_idade: int) -> None:
        self.nome: str = p_nome
        self.idade: int = p_idade
        self.ja_nasceu: bool = True

if __name__ == '__main__':
    # instanciando um novo objeto e com anotações de tipo
    p1: Pessoa = Pessoa('Arnold Schwarzenegger', 75)
    print(f'Nome: {p1.nome}, Idade: {p1.idade}')
    if p1.ja_nasceu:
        print(f'{p1.nome} já nasceu!')
```

No exemplo acima, declaramos a variável self.ja_nasceu com o valor True padrão para todos os objetos instanciados a partir da classe.

Métodos de Instância

- Os métodos de instância são funções associadas a uma classe que operam em objetos específicos da classe;
- Eles são definidos dentro da classe e são acessíveis apenas pelos objetos da classe;
- Os métodos de instância também são prefixados com o prefixo self, que é uma referência ao próprio objeto no qual o método está sendo chamado;
- Os métodos de instância podem acessar e manipular os atributos de instância do objeto no qual estão sendo executados;
- Eles podem realizar operações específicas, manipular dados ou interagir com outros objetos;

Vamos adicionar um método apresentar() à classe Pessoa.

```python
class Pessoa:
    """Definição da classe Pessoa"""

    def __init__(self, p_nome: str, p_idade: int) -> None:
        self.nome: str = p_nome
        self.idade: int = p_idade
        self.ja_nasceu: bool = True

    def apresenta(self) -> None:
        """Método para mostrar a saudação da pessoa"""
        print(f'Olá, eu sou {self.nome} e tenho {self.idade} anos.')

if __name__ == '__main__':
    # instanciando um novo objeto e com anotações de tipo
    p1: Pessoa = Pessoa('Arnold Schwarzenegger', 75)
    p1.apresenta()
```

Nesse exemplo, a classe Pessoa possui o método de instância apresenta(). Esse método recebe a referência self como parâmetro, que representa o objeto no qual o método está sendo chamado.

Dentro do método apresenta(), podemos acessar os atributos de instância do objeto usando a notação de ponto (self.nome e self.idade). Esses atributos são usados para exibir uma mensagem personalizada que contém o nome e a idade do objeto.

Ao chamar o método apresenta() em diferentes objetos, o método é executado para cada objeto, e a mensagem é exibida com base nos valores específicos dos atributos de instância associados a cada objeto.

Os métodos de instância são úteis para definir comportamentos específicos dos objetos de uma classe. Eles podem realizar cálculos, modificar atributos, interagir com outros objetos ou executar qualquer tipo de operação desejada no contexto do objeto em questão.

Exercícios para Praticar

1. Crie uma classe chamada Círculo com um atributo de classe chamado pi que armazena o valor de π (pi).
2. Defina uma classe Pessoa com os atributos de instância nome e idade e o método apresentar(), que exibe o nome e a idade da pessoa.
3. Crie uma classe Retângulo com os atributos de instância largura e altura e um método calcular_area() que retorna a área do retângulo.
    a. Adicione um método calcular_perimetro() à classe Retângulo para calcular o perímetro do retângulo.
4. Crie uma classe Carro com um atributo de instância chamado modelo.
    a. Implemente o método __init__() na classe Carro para inicializar os atributos fabricante e modelo ao criar um objeto.
    b. Adicione um método informacoes() à classe Carro que exibe as informações do carro, incluindo o fabricante e o modelo.
5. Crie uma classe ContaBancaria com os atributos de instância numero_conta, titular e saldo. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método depositar() à classe ContaBancaria que recebe um valor e adiciona esse valor ao saldo da conta.
    b. Implemente o método sacar() na classe ContaBancaria para permitir que o titular da conta saque um valor do saldo.
6. Crie uma classe Aluno com os atributos de instância nome, matricula e notas. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método adicionar_nota() à classe Aluno que recebe uma nota e a adiciona à lista de notas do aluno.
7. Crie uma classe Triangulo com os atributos de instância lado1, lado2 e lado3. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método calcular_perimetro() à classe Triangulo para calcular o perímetro do triângulo.
    b. Implemente o método verificar_tipo() na classe Triangulo para verificar o tipo do triângulo com base nos comprimentos dos lados.
8. Crie uma classe Produto com os atributos de instância nome, preco e quantidade. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método calcular_total() à classe Produto para calcular o valor total do produto (preço * quantidade).
9. Crie uma classe Funcionario com os atributos de instância nome, salario e bonus. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método calcular_salario_total() à classe Funcionario para calcular o salário total, somando o salário base e o bônus.
10. Crie uma classe Livro com os atributos de instância titulo, autor e ano_publicacao. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método informacoes() à classe Livro que exibe as informações do livro, incluindo o título, autor e ano de publicação.
11. Implemente uma classe Banco com um atributo de instância chamado taxa_juros que armazena a taxa de juros para empréstimos bancários.
    a. Crie uma classe ContaCorrente com os atributos de instância numero_conta, titular e saldo. Implemente o método __init__() para inicializar esses atributos.
    b. Adicione um método calcular_juros() à classe ContaCorrente que calcula e adiciona juros ao saldo da conta com base na taxa de juros definida na classe Banco.
12. Crie uma classe Animal com os atributos de instância nome e idade. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método emitir_som() à classe Animal que exibe um som característico do animal.
13. Implemente uma classe Biblioteca com um atributo de classe chamado livros_disponiveis que armazena a quantidade de livros disponíveis na biblioteca.

14. Crie uma classe Empregado com os atributos de instância nome, cargo e salario. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método aumentar_salario() à classe Empregado que aumenta o salário do empregado em uma porcentagem específica.
15. Implemente uma classe Cachorro com os atributos de instância nome e raca. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método latir() à classe Cachorro que exibe um som característico de latido.
16. Crie uma classe CarrinhoCompras com os atributos de instância itens (uma lista vazia) e total (inicializado como 0). Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método adicionar_item() à classe CarrinhoCompras que recebe um item e seu preço, e adiciona o item à lista de itens e atualiza o total.
17. Implemente uma classe Estudante com os atributos de instância nome e curso. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método estudar() à classe Estudante que exibe uma mensagem informando que o estudante está estudando.
18. Crie uma classe Time com os atributos de instância nome e jogadores (uma lista vazia). Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método adicionar_jogador() à classe Time que recebe o nome de um jogador e o adiciona à lista de jogadores.
19. Implemente uma classe Retangulo com os atributos de instância largura e altura. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método calcular_perimetro() à classe Retangulo que retorna o perímetro do retângulo.
20. Crie uma classe Ponto com os atributos de instância x e y. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método mover() à classe Ponto que recebe valores dx e dy e move o ponto dx unidades na direção x e dy unidades na direção y.
21. Implemente uma classe Loja com o atributo de classe numero_clientes (inicializado como 0) e o atributo de instância nome. Implemente o método __init__() para inicializar o atributo de instância.
    a. Adicione um método novo_cliente() à classe Loja que incrementa o atributo de classe numero_clientes em 1.
22. Crie uma classe Fila com o atributo de instância elementos (uma lista vazia). Implemente o método __init__() para inicializar esse atributo.
    a. Adicione um método adicionar_elemento() à classe Fila que recebe um elemento e o adiciona à lista de elementos.
23. Implemente uma classe Casa com os atributos de instância endereco e numero_comodos. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método adicionar_comodo() à classe Casa que incrementa o número de cômodos em 1.
24. Crie uma classe Funcionario com os atributos de instância nome, cargo e salario. Implemente o método __init__() para inicializar esses atributos.
    a. Adicione um método aumentar_salario() à classe Funcionario que recebe um valor percentual e aumenta o salário do funcionário com base nesse percentual.
25. Implemente uma classe Pilha com o atributo de instância elementos (uma lista vazia). Implemente o método __init__() para inicializar esse atributo.
    a. Adicione um método empilhar() à classe Pilha que recebe um elemento e o adiciona à lista de elementos.