Por: Alice Dantas

POO em Python

Classes

São um modelo para a criação de objetos.

Sintaxe:

```
class <classe>:
     <atributos>
     <metodos>
```

Observações:
- os atributos podem ser variáveis ou outros objetos
- os métodos são como funções convencionais

Referência ao próprio objeto: self

- self é uma referência ao **próprio objeto (instância)** que está chamando o método ou acessando um atributo.
- Ele deve ser o **primeiro parâmetro** dos métodos definidos dentro da classe (exceto métodos estáticos).
- Isso é útil porque uma classe pode ter **muitas instâncias diferentes**, e o self permite que cada uma mantenha **seus próprios dados**.

Situação comum:

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

Explicação:

- self.nome: cria (ou acessa) o **atributo** chamado nome **deste objeto específico**.

- nome: é o **parâmetro** passado ao método `__init__`.

- “Pegue o valor recebido no parâmetro nome e **guarde dentro do atributo nome deste objeto**.”

Por: Alice Dantas

Métodos Mágicos

Métodos mágicos são funções predefinidas que permitem que você **personalize o comportamento de objetos** em situações específicas. Eles começam e terminam com dois underscores (`__`). Exemplos:

1. `__init__(self, ...)` — Inicializador

O `__init__` é o método mágico mais básico e utilizado. Ele é chamado automaticamente quando você cria uma instância da classe, e é onde você inicializa os atributos do objeto.

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

# Criando um objeto da classe Pessoa
p1 = Pessoa("Ana", 25)
print(p1.nome)  # Saída: Ana
print(p1.idade)  # Saída: 25
```

2. `__str__(self)` — Representação em String (para `print`)

O método `__str__` define como o objeto será **representado como uma string**. Ele é chamado quando você tenta **imprimir** o objeto com `print()`.

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

    def __str__(self):
        return f'{self.nome}, {self.idade} anos'

# Criando um objeto da classe Pessoa
p1 = Pessoa("Ana", 25)
print(p1)  # Saída: Ana, 25 anos
```

Por: Alice Dantas

3. `__enter__(self)` e `__exit__(self)` — Context Manager (usado com `with`)

Permitem que sua classe seja usada como um **gerenciador de contexto** (ou seja, com o `with`). O `with` garante que alguns códigos sejam **executados antes e depois** de um bloco de código, como quando você abre e fecha arquivos. **O `__enter__`** executa antes de entrar no bloco do `with` e pode retornar algum valor. O **`__exit__`** executa ao sair do bloco do `with`, seja com sucesso ou erro.

```
class ConexaoBanco:
    def __enter__(self):
        print("Abrindo conexão com o banco...")
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        print("Fechando a conexão com o banco...")

# Usando o gerenciador de contexto
with ConexaoBanco() as banco:
    print("Conectado ao banco")

# saída: Abrindo conexão com o banco...
         Conectado ao banco
         Fechando a conexão com o banco...
```

4. `__call__(self, ...)` — Chamando o Objeto como uma Função

Permite que você "chame" um objeto como se fosse uma função. Quando você coloca parênteses após um objeto, o Python chama o `__call__` desse objeto.

```
class Somador:
    def __init__(self, valor):
        self.valor = valor

    def __call__(self, outro_valor):
        return self.valor + outro_valor

# Criando um objeto "Somador"
soma = Somador(10)

# Chamando o objeto como se fosse uma função
resultado = soma(5)
print(resultado)  # Saída: 15
```

Por: Alice Dantas

Definindo atributos

- Atributos definidos DENTRO do `__init__`:

São atributos de instância (pertencem a cada objeto criado).

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

Logo, cada vez que você faz `Pessoa("João")` ou `Pessoa("Maria")`, um novo **objeto** é criado com seu **próprio valor de `nome`**.

```
p1 = Pessoa("João")
p2 = Pessoa("Maria")

print(p1.nome)  # João
print(p2.nome)  # Maria
```

- Atributos definidos FORA do `__init__` (direto na classe):

São atributos de classe (compartilhados entre todos os objetos).

```
class Pessoa:
    especie = "Humano"  # atributo de classe

    def __init__(self, nome):
        self.nome = nome
```

Logo, `especie` é compartilhado por **todos os objetos** da classe `Pessoa`.

```
p1 = Pessoa("João")
p2 = Pessoa("Maria")

print(p1.especie)  # Humano
print(p2.especie)  # Humano
```

Por: Alice Dantas

Principais Relações entre Classes

1. Associação

- Quando uma classe **usa** ou **se relaciona** com outra.

- Exemplo: Um `Professor` dá aula em uma `Turma`.

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

class Professor:
    def __init__(self, nome, turma):
        self.nome = nome
        self.turma = turma  # Associação com a classe Turma
```

2. Agregação (uma forma especial de associação)

- Quando uma classe **possui** outra, mas elas **podem existir separadamente**. Portanto, se o objeto principal for destruído, o outro **continua existindo**.

- Exemplo: Uma `Universidade` tem `Departamentos`, mas um departamento pode existir por si só.

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

class Universidade:
    def __init__(self, nome):
        self.nome = nome
        self.departamentos = []  # Agregação

    def adicionar_departamento(self, departamento):
        self.departamentos.append(departamento)
```

Por: Alice Dantas

3. Composição (uma forma mais forte de agregação)

- Quando uma classe **compõe** outra. Logo, se o objeto principal for destruído, os objetos internos também são.

- Exemplo: Um `Carro` tem um `Motor`. Se o carro deixar de existir, o motor também.

```
class Motor:
    def __init__(self, tipo):
        self.tipo = tipo

class Carro:
    def __init__(self, modelo):
        self.modelo = modelo
        self.motor = Motor("1.6")  # Composição
```

4. Herança

- Quando uma classe **herda atributos e métodos** de outra. É uma relação **"é um"**.

- Exemplo: Um `Aluno` é uma `Pessoa`.

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

class Aluno(Pessoa):  # Herança
    def __init__(self, nome, matricula):
        super().__init__(nome)
        self.matricula = matricula
```