Por: Alice Dantas

# Tipos de Métodos em uma Classe

## 1. Método de Instância (self)

- Acessa atributos do objeto.
- Precisa de um objeto para ser chamado.

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

    def saudacao(self):
        print(f"Olá, eu sou {self.nome}")
```

## 2. Método de Classe (@classmethod)

- Usa cls como primeiro parâmetro.
- Atua sobre a **classe**, não sobre um objeto.
  Pode ser chamado sem criar um objeto.

```
class Pessoa:
    especie = "Humano"

    @classmethod
    def mostrar_especie(cls):
        print(f"Somos da espécie {cls.especie}")
```

## 3. Método Estático (@staticmethod)

- Não recebe self nem cls.
- Não acessa atributos da instância nem da classe.
- Serve como uma **função utilitária** dentro da classe.

```
class Pessoa:
    @staticmethod
    def maioridade(idade):
        return idade >= 18
```

Por: Alice Dantas

# O que são os arrobas (@) nos métodos de classe?

O **@** (chamado de decorador) é um atalho elegante para aplicar uma **função que recebe outra função como argumento** e retorna uma versão "modificada" dela. No caso de métodos em classes, os decoradores mais comuns são:

## @classmethod

- Define um **método de classe**.
- Recebe cls como primeiro parâmetro (referência à **classe** e não ao objeto).
- Pode ser chamado **sem instanciar** a classe.

```
class Pessoa:
    especie = "Humano"

    @classmethod
    def mostrar_especie(cls):
        print(f"Espécie: {cls.especie}")

Pessoa.mostrar_especie()  # Funciona sem criar nenhum objeto
```

## @staticmethod

- Define um método **que não precisa de acesso nem à instância (self), nem à classe (cls).**

```
class Pessoa:
    @staticmethod
    def maioridade(idade):
        return idade >= 18

print(Pessoa.maioridade(20))  # True
```

### - Por que usar @staticmethod se o método não depende de nada?

Em Python, todo método definido dentro de uma classe, por padrão, é considerado um método de instância. Isso significa que, ao chamá-lo, o Python automaticamente **passa o objeto (self) como o primeiro argumento**. Por isso, se você **não usar o decorador @staticmethod**, o Python vai presumir que o método espera receber self. E se o método não estiver preparado para receber esse argumento, um erro será gerado.