CRIANDO E INSTANCIANDO CLASSES

IFBA – Campus Ilhéus

Curso: Back End com Python

Objetivo da Aula

- Implementar e utilizar classes.

Revisão rápida (ligando à aula anterior)

- **Classe** = molde / tipo (ex.: Pessoa, Carro).
- **Objeto** = instância concreta da classe (ex.: p1 = Pessoa("Ana", 20)).
- **Atributo** = característica (nome, idade, cor).
- **Método** = função que o objeto executa (apresentar, acelerar).

Sintaxe básica e __init__ (explicação + exemplo)

```
class Pessoa:
    def __init__(self, nome, idade):
        # self é a referência para a instância atual
        self.nome = nome      # atributo de instância
        self.idade = idade

    def apresentar(self):
        # método que utiliza atributos da instância
        return f"Olá, sou {self.nome} e tenho {self.idade} anos."

    def fazer_aniversario(self):
        self.idade += 1
```

Uso / instanciação:

```
p = Pessoa("Ana", 30) # cria um objeto Pessoa
print(p.apresentar())    # "Olá, sou Ana e tenho 30 anos."
p.fazer_aniversario()
print(p.idade)           # 31
```

- **Pontos importantes:**
  - __init__ é chamado automaticamente quando Pessoa(...) é executado.
  - Sempre passe self como primeiro parâmetro dos métodos de instância.
  - self.nome e self.idade pertencem àquela instância específica.

Valores padrão, argumentos nomeados

```
class Livro:
    def __init__(self, titulo, autor, ano=None):
        self.titulo = titulo
        self.autor = autor
        self.ano = ano     # pode ser None se desconhecido

# instanciar com argumento nomeado:
l = Livro(titulo="1984", autor="George Orwell", ano=1949)
```

Atributos de classe vs de instância

```
class Carro:
    contador = 0        # atributo de classe (compartilhado por todas as instâncias)

    def __init__(self, marca, modelo):
        self.marca = marca      # atributo de instância
        self.modelo = modelo
        Carro.contador += 1  # incrementa o contador para cada criação
```

Atributos de classe vs de instância

```
# Uso:
c1 = Carro("Fiat", "Uno")
c2 = Carro("Ford", "Ka")
print(Carro.contador)  # 2
```

Método especial __str__ e representações

- O método especial __str__ em Python é um **método dunder** (double underscore), ou seja, um método especial definido dentro de classes para modificar o comportamento padrão de objetos.
- O método __str__ serve para **definir a representação em forma de string** (texto) de um objeto — ou seja, o que será mostrado quando usamos print(objeto) ou str(objeto).

Método especial __str__ e representações

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

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

p = Pessoa("João", 25)
print(p)  # chama p.__str__() -> "João (25 anos)"
```

Propriedades (property) — acesso controlado (opcional, útil em 2ª parte)

- A funcionalidade **@property** em Python é um **recurso que transforma um método em um atributo de leitura**, permitindo acessar o resultado de uma função **sem precisar chamá-la com parênteses**.
- Ela é muito usada para **controlar o acesso a atributos** (encapsulamento), mantendo uma sintaxe simples e elegante.

Propriedades (property) — acesso controlado (opcional, útil em 2ª parte)

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

    @property
    def area(self):
        return self.largura * self.altura

r = Retangulo(3, 4)
print(r.area)   # 12 — acessa como atributo, mas é cálculo
```

Propriedades (property) — acesso controlado (opcional, útil em 2ª parte)

- **Por que usar @property?**
  - **Encapsulamento** — você pode mudar a forma como um valor é calculado sem alterar o modo como ele é acessado.
  - **Validação** — você pode controlar o que entra e sai de um atributo.
  - **Leitura mais natural** — o código fica mais limpo e parecido com linguagem natural.

Erros comuns a ensinar (pontos para reforçar)parte)

- Esquecer self como primeiro parâmetro.
- Usar valores padrão mutáveis no __init__, ex.: def __init__(..., lista=[]) — evitar. Use None e inicialize dentro do construtor.
- Confundir atributo de classe com de instância (ex.: mudar Carro.contador vs self.contador).
- Escrever __init em vez de __init__ (typo comum).

Exercício

- Crie um programa em Python que simule um **jardim virtual**. Nesse jardim, é possível **plantar flores** e **regar** para que elas cresçam. **Crie a classe Flor, contendo:**
- **Atributos:**
  - nome: o nome da flor
  - altura: altura atual da flor em centímetros (inicia em 0)
  - rega: quantidade de vezes que a flor foi regada (inicia em 0)
- **Métodos:**
  - regar(): aumenta a altura da flor em **2 cm** e soma 1 à quantidade de regas. Deve exibir uma mensagem dizendo que a flor foi regada.
  - mostrar_status(): exibe o nome da flor, sua altura e quantas vezes foi regada.

Exercício

- Em seguida no programa principal, crie:
- Dois objetos a partir da classe Flor (flor1 e flor2), passando para o construtor Girassol e Tulipa, respectivamente.
- Regue 2x a flor1 e 1x a flor2.
- Mostre o status da flor1 e da flor2.