EXERCÍCIOS DE PYTHON - PROGRAMAÇÃO ORIENTADA A OBJETOS

Exercício 1 – Criando sua primeira classe

Crie uma classe chamada Pessoa que deve armazenar:

O nome da pessoa.

A idade da pessoa.

1. No método especial __init__, inicialize esses atributos.

→ O ***__init__*** é o construtor da classe: chamado sempre que criamos um objeto.

→ O ***self*** representa o próprio objeto criado.

2. Crie um método chamado apresentar que imprima:

'Olá, meu nome é [***nome***] e tenho [***idade***] anos.'

3. Crie dois objetos e chame o método ***apresentar()*** para cada um.

```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.')


# Programa principal com interação do usuário
if __name__ == "__main__":
    # Solicitando dados do usuário
    nome = input("Digite seu nome: ")
    idade = int(input("Digite sua idade: "))

    # Criando objeto com os dados fornecidos
    pessoa_usuario = Pessoa(nome, idade)

    # Apresentando a pessoa
    pessoa_usuario.apresentar()
```

Exercício 2 – Classe com comportamento

Crie uma classe chamada ***ContaBancaria*** que possua:

- Atributos: ***titular***, ***saldo (inicialmente 0).***

- Métodos:
  - ***depositar(valor)*** aumenta o saldo.
  - ***sacar(valor)*** diminui o saldo, se houver saldo suficiente.
  - ***mostrar_saldo()*** mostra o saldo atual.

→ **def** define métodos em classes e sempre recebe **self**.

Crie um objeto e faça depósitos e saques.

```python
class ContaBancaria:
    def __init__(self, titular): self.titular, self.saldo = titular, 0
    def depositar(self, valor): self.saldo += valor
    def sacar(self, valor):
        if valor <= self.saldo: self.saldo -= valor
        else: print("Saldo insuficiente!")
    def mostrar_saldo(self): print(f"Saldo: R$ {self.saldo:.2f}")

# Uso
conta = ContaBancaria("João")
conta.depositar(100)
conta.sacar(30)
conta.mostrar_saldo()
conta.sacar(80)  # Saldo insuficiente
```

Operadores Básicos

= (Igual)

- Atribui um valor
- Exemplo: x = 10 (x recebe 10)

+= (Mais Igual)

- Soma e guarda o resultado
- Exemplo: saldo += 50 (saldo = saldo + 50)

-= (Menos Igual)

- Subtrai e guarda o resultado
- Exemplo: saldo -= 30 (saldo = saldo - 30)

Exercício 3 – Relacionando objetos

Crie duas classes: **Aluno** e **Curso**.

1. **Aluno**: atributos nome, id_aluno, e um método apresentar().
2. **Curso**: atributos nome, codigo, alunos (lista).
   - ***adicionar_aluno(aluno)***: adiciona aluno à lista.
   - ***listar_alunos()***: mostra todos os alunos do curso.

→ Mostra como um objeto pode conter outros objetos.

Exercício 4 – Trabalhando com métodos especiais

Crie uma classe Produto com atributos: nome, preco.

Implemente __**str**__ para retornar:

'Produto: [nome] - Preço: R$ [preco]'.

→ O __***str***__ é chamado automaticamente quando usamos print().

Crie três produtos e mostre-os com print().

Exercício 5 – Criando uma classe mais completa

Crie uma classe **Biblioteca**:

- Atributos: nome, livros (lista).
- Métodos:
  - adicionar_livro(titulo)
  - listar_livros()
  - buscar_livro(titulo)

→ Mostra como usar listas em classes e manipular dados de um objeto.

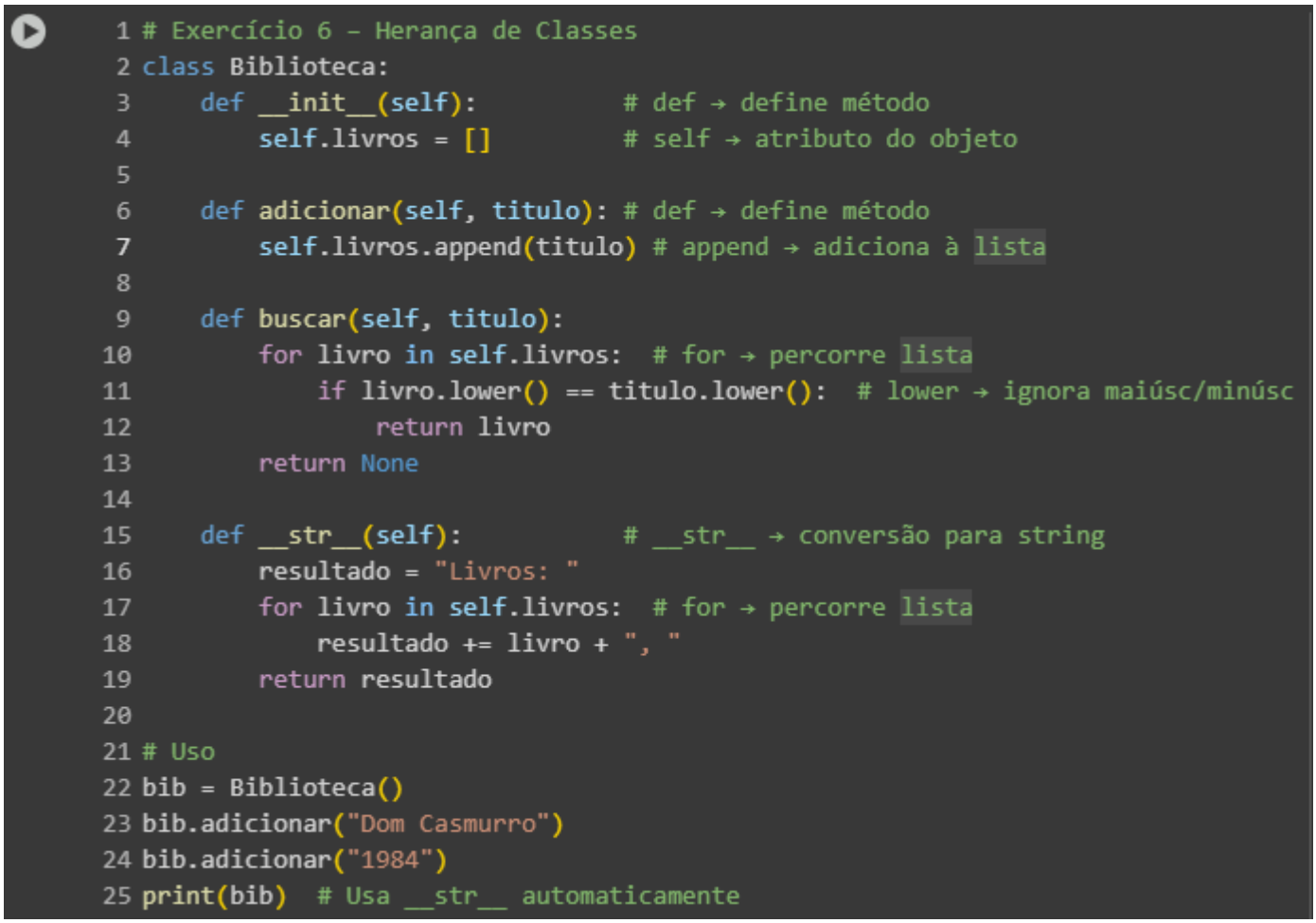
```python
# Exercício 4 - Classe Produto com __str__
class Produto:
    def __init__(self, nome, preco):
        self.nome = nome      # Atributo nome
        self.preco = preco    # Atributo preço

    def __str__(self):
        # Método especial que define como o objeto vira string
        return f'Produto: {self.nome} - Preço: R$ {self.preco:.2f}'

# Criando 3 produtos
p1 = Produto("Notebook", 2500)
p2 = Produto("Mouse", 89.90)
p3 = Produto("Teclado", 150)

# Print chama __str__ automaticamente
print(p1)
print(p2)
print(p3)

print("\n" + "="*50 + "\n")

```

Exercício 6 – Herança de Classes

Crie um sistema simples de biblioteca.

- Tenha uma classe **Livro** com atributos como título, autor e se está disponível ou não.
- Crie também a classe **Biblioteca**, que deve armazenar vários livros e permitir que o usuário "empreste" ou "devolva" livros.
- Faça um pequeno menu de testes que permita adicionar livros, listar os disponíveis e realizar empréstimos.

O **objetivo dos do exercício** é mostrar: Como **objetos se relacionam** dentro de um sistema (biblioteca tem livros).



```python
# Exercício 6 – Herança de Classes
class Biblioteca:
    def __init__(self):          # def → define método
        self.livros = []         # self → atributo do objeto

    def adicionar(self, titulo): # def → define método
        self.livros.append(titulo) # append → adiciona à lista

    def buscar(self, titulo):
        for livro in self.livros:  # for → percorre lista
            if livro.lower() == titulo.lower():  # lower → ignora maiúsc/minúsc
                return livro
        return None

    def __str__(self):           # __str__ → conversão para string
        resultado = "Livros: "
        for livro in self.livros:  # for → percorre lista
            resultado += livro + ", "
        return resultado

# Uso
bib = Biblioteca()
bib.adicionar("Dom Casmurro")
bib.adicionar("1984")
print(bib)  # Usa __str__ automaticamente
```

.

Exercício 7

Crie um sistema que represente pessoas de uma escola.

- Tenha uma classe **Pessoa** (atributos: nome, idade).
- Crie duas classes que herdam dela: **Aluno** (atributo: matrícula) e **Professor** (atributo: disciplina).
- Cada um deve ter um método ***apresentar()*** que mostra suas informações, mas o **Aluno** deve acrescentar a matrícula e o **Professor** a disciplina.

```python
#Exercício 7 Crie um sistema que represente pessoas de uma escola
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def apresentar(self):
        return f"Nome: {self.nome}, Idade: {self.idade}"

class Aluno(Pessoa):
    def __init__(self, nome, idade, matricula):
        super().__init__(nome, idade)
        self.matricula = matricula

    def apresentar(self):
        return f"{super().apresentar()}, Matrícula: {self.matricula}"

class Professor(Pessoa):
    def __init__(self, nome, idade, disciplina):
        super().__init__(nome, idade)
        self.disciplina = disciplina

    def apresentar(self):
        return f"{super().apresentar()}, Disciplina: {self.disciplina}"
# Exemplo de uso
aluno1 = Aluno("João", 16, "A123")
prof1 = Professor("Maria", 40, "Matemática")

print(aluno1.apresentar())
print(prof1.apresentar())

```

O **objetivo dos do exercício** é mostrar: Como **herança e polimorfismo** funcionam em Python.

Resumo dos conceitos:

class → define uma nova classe (um molde para criar objetos).

__init__ → método construtor, executado ao criar um novo objeto.

self → referência ao próprio objeto. Sempre vem como primeiro parâmetro nos métodos.

def → define métodos dentro da classe.

__str__ → método especial que define como o objeto será exibido quando usado com print().

Objeto → instância da classe (exemplo real criado a partir do molde).

self → "eu mesmo" (o objeto)

def → "criar função"

for → "para cada" percorre a lista.

append → "adicionar na lista" no final da lista.

lower → "deixar minúsculo"

__**str**__ → "como me transformar em texto"