Aula 1 — Introdução à Programação Orientada a Objetos (POO) com Python

---

Slide 1 — Capa

Disciplina: Programação Orientada a Objetos

Aula 1: Introdução à POO com Python

Professor: Antonio Luis **Duração:** 4 horas

Slide 2 — Objetivos da aula (o que você vai aprender)

- Entender a diferença entre programação estruturada e orientada a objetos.
- Conhecer e identificar: **classe**, **objeto**, **atributo** e **método**.
- Criar a primeira classe em Python e instanciar objetos.
- Aplicar abstração modelando objetos do cotidiano.

Slide 3 — Agenda da aula (fluxo)

1. Paradigmas: estruturada vs POO (20 min)
2. Conceitos fundamentais (classe, objeto, atributos, métodos) (20 min)
3. Exemplo guiado em Python: `Pessoa` (40 min)
4. Atividade em grupo: modelagem de objetos (30 min)
5. Prática supervisionada: criar `Carro` (50 min)
6. Desafio individual: `ContaBancaria` (30 min)
7. Revisão e dúvidas (30 min)

Slide 4 — Paradigmas de programação (breve)

Programação Estruturada

- Foca em procedimentos/funções.
- Dados e funções são separados.
- Boa para scripts e problemas lineares.

Programação Orientada a Objetos (POO)

- Foca em objetos que reúnem dados + comportamento.
- Facilita modelagem do mundo real.
- Melhora organização, reaproveitamento e manutenção.

Comentário para escrever no quadro: “Função vs Objeto: dados como variáveis soltas × dados encapsulados em objetos”.

Slide 5 — Exemplo comparativo (código)

Estruturada (função):

```
def calcular_area_retangulo(base, altura):
    return base * altura

print(calcular_area_retangulo(10, 5))
```

Orientada a Objetos:

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

    def calcular_area(self):
        return self.base * self.altura

r = Retangulo(10, 5)
print(r.calcular_area())
```

Slide 6 — Conceitos fundamentais (definições claras)

- **Classe:** molde/projeto (ex.: Carro).
- **Objeto (instância):** exemplar concreto do molde (ex.: meu carro, um Fusca 1978).
- **Atributo:** característica/estado (ex.: cor, ano).
- **Método:** comportamento/ação (ex.: ligar, frear).

Exemplo simples: Classe `Carro` → atributos: `marca`, `modelo`, `ano`; métodos: `ligar()`, `desligar()`.

Nota para o quadro: desenhe um retângulo “Carro” com atributos e métodos listados (UML simplificado).

Slide 7 — Anatomia de uma classe em Python (passo a passo)

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

    def apresentar(self):
        return f"Olá, eu sou {self.nome} e tenho {self.idade} anos."
```

Explique cada linha:

- `class Pessoa:` → declaração da classe.
- `def __init__(self, nome, idade):` → construtor: inicializa atributos.
- `self.nome = nome` → atribui valor ao atributo do objeto.
- `def apresentar(self):` → método da classe.
- `return f"..."` → retorno formatado.

Slide 8 — Instanciando objetos e chamando métodos

```python
p1 = Pessoa("Ana", 20)
p2 = Pessoa("Carlos", 25)

print(p1.apresentar())
print(p2.apresentar())
```

Resultado esperado:

```
Olá, eu sou Ana e tenho 20 anos.
Olá, eu sou Carlos e tenho 25 anos.
```

Slide 9 — Exemplo detalhado: Classe `Carro` (com estado)

```python
class Carro:
    def __init__(self, marca, modelo, ano):
        self.marca = marca
        self.modelo = modelo
        self.ano = ano
        self.ligado = False

    def ligar(self):
        if not self.ligado:
            self.ligado = True
```

```
            return "Carro ligado"
        return "Já está ligado"

    def desligar(self):
        if self.ligado:
            self.ligado = False
            return "Carro desligado"
        return "Já está desligado"

    def exibir_info(self):
        return f"{self.marca} {self.modelo} ({self.ano}) - {'ligado' if
self.ligado else 'desligado'}"
```

Uso no REPL:

```
c = Carro('Fiat', 'Uno', 2010)
print(c.exibir_info())
print(c.ligar())
print(c.exibir_info())
```

Slide 10 — Atividade em grupo (modelagem)

Objetivo: praticar abstração e identificação de atributos/métodos.

Tarefa (30 min):

1. Formem grupos de 3–4 alunos.
2. Escolham um objeto (ex.: Celular, Escola, Livro, Controle Remoto).
3. Em 15 minutos, escrevam: nome da classe, 6 atributos e 4 métodos (mínimo).
4. Em 10 minutos, desenhem um diagrama simples (retângulo com atributos e métodos) e um exemplo de instância.
5. Cada grupo apresenta 2 minutos.

Gabarito orientador (ex.: Celular):

- Classe: `Celular`
- Atributos: `marca`, `modelo`, `capacidade_bateria`, `armazenamento`, `tela_tamanho`, `sistema_operacional`.
- Métodos: `ligar()`, `desligar()`, `fazer_chamada(n)`, `instalar_app(nome)`.

Slide 11 — Prática supervisionada: implementar `Carro` (código pronto)

Objetivo: produzir código funcional com ajuda do professor.

Passos (50 min):

1. Escrever a classe `Carro` (base no slide anterior).
2. Instanciar 2 carros com marcas diferentes.
3. Simular: exibir info, ligar, tentar ligar de novo, desligar.
4. Comentar o código e rodar no terminal/REPL.

Dica de execução: `python3 nome_arquivo.py` ou usar REPL interativo no VS Code/IDLE.

Slide 12 — Desafio individual: `ContaBancaria` (30 min)

Requisitos:

- Atributos: `titular` (str), `saldo` (float).
- Métodos:
  - `depositar(valor)` → soma ao saldo e retorna novo saldo.
  - `sacar(valor)` → se saldo suficiente, subtrai e retorna True; caso contrário retorna False.
  - `exibir_saldo()` → imprime/retorna o saldo formatado.
- Deve tratar entradas inválidas (ex.: valores negativos) com mensagens amigáveis.

Critérios de aceitação: o método `sacar` não pode permitir saldo negativo.

Gabarito (exemplo):

```
class ContaBancaria:
    def __init__(self, titular, saldo=0.0):
        self.titular = titular
        self.saldo = float(saldo)

    def depositar(self, valor):
        if valor <= 0:
            return "Valor de depósito inválido"
        self.saldo += valor
        return self.saldo

    def sacar(self, valor):
        if valor <= 0:
            return "Valor de saque inválido"
        if valor > self.saldo:
            return False
        self.saldo -= valor
        return True
```

```
    def exibir_saldo(self):
        return f"Saldo de {self.titular}: R$ {self.saldo:.2f}"
```

Slide 13 — Erros comuns e como depurar (lista prática)

- **Esquecer `self`** em métodos** → `NameError` ou comportamento inesperado.
- **Indentação errada** → `IndentationError`.
- **Chamar método sem parênteses** → retorna objeto função.
- **Atribuir atributo fora do `__init__`** sem intenção** → inconsistências de estado.
- **Tipos inesperados** (somar str + float) → `TypeError`.

Dicas de depuração: print() para inspecionar valores, usar REPL, ler tracebacks e apontar a linha do erro.

Slide 14 — Semântica de referência (explicação e exemplo)

Conceito: em Python, variáveis armazenam referências a objetos, não cópias do conteúdo.

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

p1 = Pessoa('Ana')
p2 = p1
p2.nome = 'Maria'
print(p1.nome)  # 'Maria'
```

Explicação: `p1` e `p2` referenciam o mesmo objeto. Alterações via `p2` aparecem em `p1`.

Slide 15 — Cópia rasa x cópia profunda (shallow x deep copy)

Exemplo com listas:

```
import copy

lista1 = [[1,2],[3,4]]
lista2 = lista1          # referência
lista3 = copy.copy(lista1)   # shallow copy
lista4 = copy.deepcopy(lista1)  # deep copy

lista1[0][0] = 99
print(lista2)  # muda
```

```
print(lista3)  # muda (sublistas referenciadas)
print(lista4)  # não muda
```

Slide 16 — Encapsulamento em Python (convenções)

Princípio: proteger dados do acesso direto e impor regras.

Convenções em Python:

- `atributo` (público)
- `_atributo` (protegido por convenção)
- `__atributo` (name mangling — pseudo-privado)

Exemplo com property:

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

    @property
    def nome(self):
        return self._nome

    @nome.setter
    def nome(self, valor):
        if not valor:
            raise ValueError('Nome inválido')
        self._nome = valor
```

Nota: mostre no quadro o fluxo de leitura/escrita via `property`.

Slide 17 — UML simples (diagrama de classes) e mapeamento para Python

Exemplo UML (texto):

```
+----------------+
|     Carro      |
+----------------+
| - marca: str   |
| - modelo: str  |
| - ano: int     |
+----------------+
| + ligar(): str  |
| + desligar(): str|
+----------------+
```

Mapeamento para Python: a UML mostra atributos (com sinais +/-) e métodos; implemente em Python na classe.

Atividade rápida: peça que convertam o diagrama UML do “Celular” para código Python em 8 minutos.

Slide 18 — Associação, Agregação e Composição (definições e exemplo)

- **Associação:** relacionamento fraco entre objetos (ex.: Aluno - Curso).
- **Agregação:** relação “tem” mas objeto parte pode existir sem o todo (ex.: Turma tem Aluno).
- **Composição:** relação forte, a parte não existe sem o todo (ex.: Casa tem Cômodo — se a casa é destruída, os cômodos também).

Exemplo de composição (código):

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

class Carro:
    def __init__(self, marca, motor):
        self.marca = marca
        self.motor = motor  # composição/associação dependendo do contexto
```