ENCAPSULAMENTO E MÉTODOS DE INSTÂNCIA

IFBA – Campus Ilhéus

Curso: Back End com Python

Objetivo da Aula

- Ensinar o conceito de encapsulamento e a criação de métodos de instância para manipulação de dados dentro de objetos.

Encapsulamento

- É a prática de esconder os detalhes internos (estado/implementação) de um objeto, expondo apenas uma interface pública (métodos) para uso externo.
- Importância:
  - Protege invariantes (por exemplo, saldo não pode ficar negativo).
  - Permite mudar a implementação interna sem quebrar quem usa a classe.
  - Facilita depuração e manutenção.

“Modificadores” de acesso

- Python **não** tem public/private/protected rígidos como Java/C++; usa convenções e *name mangling*:
  - atributo (sem underscore) → público (convenção: acessível diretamente).
  - _atributo (um underscore) → protegido por convenção (sinal de “não use externamente; pode ser usado por subclasses”).
  - __atributo (dois underscores) → private via name mangling (é transformado internamente em _Classe__atributo, dificultando acesso acidental).

Métodos de instância: self

- Método de instância: função definida na classe que opera sobre uma instância; sempre recebe self como primeiro parâmetro.
- self é a referência ao próprio objeto — permite ler/alterar atributos daquele objeto.

Exemplo simples

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

    def cumprimenta(self):
        return f"Olá, eu sou {self.nome}“

p = Pessoa("Ana")
print(p.cumprimenta())  # Olá, eu sou Ana
```

Exemplo completo

```
class ContaBancaria:
    def __init__(self, titular, saldo_inicial=0):
        self.titular = titular          # público
        self.__saldo = saldo_inicial      # "privado" (name mangling)

    # método de instância para depositar
    def depositar(self, valor):
        if valor <= 0:
            raise ValueError("Valor de depósito deve ser positivo.")
        self.__saldo += valor
```

Exemplo completo - continuação

```
# método de instância para sacar
def sacar(self, valor):
    if valor <= 0:
        raise ValueError("Valor de saque deve ser positivo.")
    if valor > self.__saldo:
        raise ValueError("Saldo insuficiente.")
    self.__saldo -= valor

# getter explícito
def obter_saldo(self):
    return self.__saldo

# representação útil
def __repr__(self):
    return f"ContaBancaria({self.titular!r}, saldo={self.__saldo})"
```

Exemplo completo - continuação

```
# Uso
  c = ContaBancaria("João", 100)
c.depositar(50)
try:
    c.sacar(200)
except ValueError as e:
    print(e)  # Saldo insuficiente.
print(c.obter_saldo())  # 150
```

- **Observações:**
  - __saldo impede acesso direto por convenção (internamente vira _ContaBancaria__saldo).
  - Regras de negócio (saldo >= 0) garantidas pelos métodos.

Getters e setters em Python: @property

```
class Conta:
    def __init__(self, titular, saldo=0):
        self.titular = titular
        self._saldo = saldo  # atributo interno

    @property
    def saldo(self):
        """Getter: permite leitura: conta.saldo"""
        return self._saldo
```

Getters e setters

```
#Continuação
    @saldo.setter
    def saldo(self, valor):
        """Setter: valida antes de atribuir: conta.saldo = novo_valor"""
        if valor < 0:
            raise ValueError("Saldo não pode ser negativo.")
        self._saldo = valor
```

Getters e setters

```
#Continuação
c = Conta("Marta", 200)
print(c.saldo)   # chama o getter
c.saldo = 150    # chama o setter (validação)
# c.saldo = -10  -> ValueError
```

- **Dica importante:** dentro do setter use sempre um atributo interno diferente do nome da propriedade (ex.: _saldo), caso contrário você entra em recursão infinita (self.saldo = ... chamaria o setter novamente).

Armadilhas e boas práticas

- **Não confundir convenção com segurança:** _X e __X não tornam algo verdadeiramente impossível de acessar. São mecanismos de proteção por convenção.
- **Evite expor atributos públicos sem necessidade**; prefira propriedades se houver lógica de validação no acesso.
- **Nunca use o mesmo nome para a propriedade e o atributo interno** (evitar recursão). Ex.: use _saldo para a implementação e saldo para a propriedade.

Armadilhas e boas práticas

- **Mantenha responsabilidades claras:** métodos devem encapsular regras (ex.: sacar() deve checar saldo e lançar exceção quando necessário).
- **Documente** a API da classe (docstrings), especialmente quando algumas operações são proibidas.

Exercícios

1. **Retangulo**

- Implemente classe Retangulo com atributos privados _largura e _altura.
- Valide que largura e altura > 0 (no setter).
- Métodos: area() e perimetro().

Exercícios

2. **Livro (biblioteca)**
   - Classe Livro com _emprestado (bool).
   - Métodos: emprestar() (raise se já emprestado), devolver().