Python Dentro de uma classe, quando utilizar \_ ou \_\_ antes de um método? Além disso, quando utilizar o @staticmethod? Também preciso saber qual a forma correta de acessar uma variável interna? Como criar os métodos getter e setter? Explique e dê um exemplo prático.

**O ChatGPT disse:**

Ótimas perguntas! Vamos abordar ponto por ponto com explicações **claras** e um **exemplo prático completo** em Python orientado a objetos.

**🔹 1. Diferença entre \_ e \_\_ em métodos ou atributos**

**✅ \_nome (um sublinhado):**

* **Convenção** que indica que o método ou atributo **é interno** (uso interno, "protegido").
* **Não impede** acesso externo, mas sinaliza que **não deve ser acessado diretamente** fora da classe ou subclasses.

**✅ \_\_nome (dois sublinhados):**

* **Name mangling**: o Python renomeia internamente o atributo para \_NomeDaClasse\_\_nome.
* **Dificulta** o acesso externo acidental (não impede totalmente).

**🔹 2. Quando usar @staticmethod**

* Use @staticmethod quando o método **não depende de self (a instância) nem cls (a classe)**.
* Ele se comporta como uma função comum dentro da classe, útil para **organização lógica**.

**🔹 3. Como acessar variável interna corretamente**

* Acessa-se normalmente com self.\_variavel ou self.\_\_variavel (dentro da classe).
* Fora da classe:
  + \_variavel ainda pode ser acessado (mas **não recomendado**).
  + \_\_variavel deve ser acessado como objeto.\_NomeDaClasse\_\_variavel (evite!).

**🔹 4. Como criar métodos Getter e Setter**

* Forma tradicional: usando métodos com get\_ e set\_.
* Forma moderna: usando @property.

**✅ Exemplo prático**

python

CopiarEditar

class ContaBancaria:

def \_\_init\_\_(self, titular, saldo):

self.\_titular = titular # protegido (convenção)

self.\_\_saldo = saldo # privado (name mangling)

# Getter moderno

@property

def saldo(self):

return self.\_\_saldo

# Setter moderno com verificação

@saldo.setter

def saldo(self, valor):

if valor < 0:

raise ValueError("Saldo não pode ser negativo.")

self.\_\_saldo = valor

# Método interno (protegido)

def \_log\_transacao(self, valor):

print(f"[LOG] Transação de R${valor:.2f} realizada.")

# Método privado

def \_\_atualizar\_sistema(self):

print("[Sistema atualizado]")

# Método público

def depositar(self, valor):

if valor > 0:

self.\_\_saldo += valor

self.\_log\_transacao(valor)

self.\_\_atualizar\_sistema()

else:

print("Valor inválido para depósito.")

# Método estático: não usa self

@staticmethod

def banco\_info():

return "Banco XPTO - Código 123"

# Uso

conta = ContaBancaria("Lucas", 1000)

# Acessando getter

print("Saldo:", conta.saldo)

# Usando setter

conta.saldo = 1500

# Tentativa de acesso direto

# print(conta.\_\_saldo) ❌ Erro: atributo privado

print(conta.\_ContaBancaria\_\_saldo) # ⚠️ Possível, mas NÃO recomendado

# Usando método estático

print(ContaBancaria.banco\_info())

# Fazendo depósito

conta.depositar(300)

**✅ Resumo prático:**

| **Sintaxe** | **Uso** |
| --- | --- |
| \_variavel | Interno/protegido (por convenção) |
| \_\_variavel | Privado com name mangling |
| @staticmethod | Função dentro da classe sem acesso a self nem cls |
| @property | Getter moderno |
| @<propriedade>.setter | Setter moderno com encapsulamento |