Ordem de Procedência no escopo.

Existe uma ordem de procedência no escopo que afeta como o python (e todas as linguagens) funcionam.

Python interpreta importações, variáveis, funções e métodos. Não é uma precedência matemática mas uma sequência lógica de execução. (ex: não tem lógica eu chamar uma variável antes de criar ela).

As Importações vêm primeiro

- Quando você importa um módulo, Python carrega o código antes de qualquer outra coisa. Isso acontece na fase de inicialização do script.
- EX: import math #isso é resolvido e executado antes de qualquer coisa no python.
- Se o módulo não for encontrado, o programa nem começa — ele lança um `ModuleNotFoundError`.

Ordem de definição: variáveis, funções e métodos

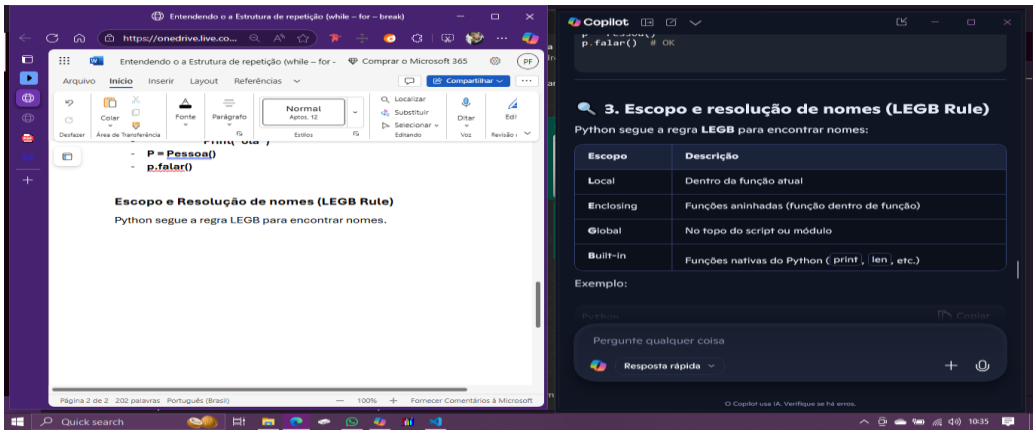
O python lê os códigos de cima para baixo, então a ordem em que você define as coisas importam muito.

- **Variáveis** precisam ser definidas antes de serem usadas/chamadas
- **EX: x = 10**
- **Print(x)**
- **Funções** podem ser chamadas depois que são definidas.
- EX: **def saudacao(): #declarando a função**
- **Print(“olá”)**
- **Saudacao() #chamando a função**
- **Métodos (em classes)** só existem depois que a classe é definida.

- EX: **class Pessoa:**
- **Def falar(self):**
- **Print(“olá”)**
- **P = Pessoa()**
- **p.falar()**

Escopo e Resolução de nomes (LEGB Rule)

Python segue a regra LEGB para encontrar nomes.



Resumo da "precedência de execução"

1. **Importações** são resolvidas primeiro.
2. **Definições** (variáveis, funções, classes) são lidas de cima para baixo.
3. **Execução** segue essa ordem.
4. **Resolução de nomes** segue o escopo LEGB.

Exemplo de uso da ordem de procedência no python:

mini script que mistura **importações**, **variáveis**, **funções**, **métodos**, e **escopos** — tudo com comentários para mostrar a ordem de execução e como Python resolve cada parte.

```
# 1️⃣ Importação (vem primeiro)
```

```python
import math

# 2️⃣ Variável global
mensagem_global = "Bem-vinda à calculadora mágica!"

# 3️⃣ Função externa
def saudacao(nome):
    print(mensagem_global)

    # 4️⃣ Variável local
    mensagem_local = f"Olá, {nome}!"
    print(mensagem_local)

    # 5️⃣ Função aninhada (escopo Enclosing)
    def mostrar_pi():
        print(f"O valor de pi é aproximadamente {math.pi:.2f}")

    mostrar_pi()

# 6️⃣ Classe com método
class Calculadora:
    def __init__(self, numero):
        self.numero = numero

    def elevar_ao_quadrado(self):
        return self.numero ** 2

# 7️⃣ Execução principal
if __name__ == "__main__":
    saudacao("Patricia")  # Chama a função com escopo local/enclosing
    calc = Calculadora(7) # Cria objeto da classe
    resultado = calc.elevar_ao_quadrado()  # Chama método
    print(f"7 ao quadrado é {resultado}")
```

O que esse código mostra:

- **Importações** são resolvidas primeiro (sem `math`, o código nem roda).
- **Variáveis globais** são acessíveis em qualquer lugar do script.
- **Funções** precisam ser definidas antes de serem chamadas.
- **Funções aninhadas** têm acesso ao escopo da função externa.
- **Classes e métodos** só funcionam depois de definidos.
- O bloco `if __name__ == "__main__"` garante que o código só execute se for o script principal (boa prática!).