Estruturas Condicionais em Python

1) Por que precisamos de condições?

Até agora, os programas que escrevemos executavam **todas as linhas em ordem**, sempre da mesma forma. Mas, na vida real, muitas situações exigem **escolhas**:

- Se chover, levo guarda-chuva.
- Se a nota for maior ou igual a 6, o aluno está aprovado.
- Se o valor do carro for acima de 80 km/h, aplicamos multa.

☞ Em programação, usamos **estruturas condicionais** para representar esse tipo de decisão. No Python, a principal estrutura de decisão é o `if`.

---

2) Estrutura básica do `if`

O comando `if` em Python significa **"se"**. Ele é usado quando queremos que o programa **tome uma decisão**:

- **Se** uma coisa for verdadeira → execute algo.
- **Se não** for verdadeira → ignore.

🔹 O que é uma condição?

Uma **condição** é uma pergunta que o computador responde com apenas duas possibilidades:

- **Verdadeiro (True)**
- **Falso (False)**

Exemplos do mundo real:

- "Está chovendo?" → pode ser **verdadeiro** ou **falso**.
- "A idade é maior ou igual a 18?" → pode ser **verdadeiro** ou **falso**.
- "O número digitado é maior que 10?" → pode ser **verdadeiro** ou **falso**.

🔹 Representando no Python

Em Python, as condições são escritas com **operações de comparação**. Alguns exemplos:

```
10 > 5      # True, porque 10 é maior que 5
3 < 1       # False, porque 3 não é menor que 1
7 == 7      # True, porque 7 é igual a 7
8 != 2      # True, porque 8 é diferente de 2
```

☞ Cada vez que escrevemos uma comparação, o Python responde com **True ou False**.

🔹 Como funciona o `if`

Formato básico:

```
if <condição>:
    bloco
```

- `<condição>` → é a pergunta que resulta em **True** ou **False**.
- `bloco` → é o conjunto de instruções que o Python só executa **se a condição for True**.

🔹 Exemplo prático

```
idade = 20

if idade >= 18:
    print("Você é maior de idade")
```

Explicação:

1. O programa guarda o valor `20` na variável `idade`.
2. A condição `idade >= 18` é avaliada → `20 >= 18` → **True**.
3. Como o resultado é verdadeiro, o bloco (`print(...)`) é executado.

Se a idade fosse `15`, a condição `15 >= 18` seria **False**, e nada seria executado.

🔹 Indentação (recuo)

Em Python, o bloco do `if` é definido pelo **espaço no início da linha** (indentação). Por exemplo:

```
if True:
    print("Essa linha está dentro do if")
print("Essa linha está fora do if")
```

Saída:

```
Essa linha está dentro do if
Essa linha está fora do if
```

⚠ Atenção: em Python, **sem recuo correto, o programa dá erro**.

Resumo

- Condição = pergunta respondida com **True ou False**.
- O `if` executa o bloco apenas quando a condição é **True**.
- O recuo (indentação) define o que faz parte do bloco.

---

3) Exemplo: qual número é maior?

```
a = int(input("Primeiro valor: "))
b = int(input("Segundo valor: "))

if a > b:
    print("O primeiro número é o maior!")

if b > a:
    print("O segundo número é o maior!")
```

- Se `a > b` for verdadeiro, a primeira mensagem aparece.
- Se `b > a` for verdadeiro, a segunda mensagem aparece.
- Se os dois números forem iguais, **nenhuma das mensagens aparece**.

☞ O Python **sempre executa a linha do `if`**, mas só entra no bloco se a condição for verdadeira.

---

4) Exemplo prático: idade do carro

```
idade = int(input("Digite a idade do seu carro: "))

if idade <= 3:
    print("Seu carro é novo")

if idade > 3:
    print("Seu carro é velho")
```

- Para idades 0, 1, 2, 3 → imprime “carro novo”.
- Para idades maiores que 3 → imprime “carro velho”.

Note que o programa ainda não trata valores inválidos (ex.: idade negativa).

---

5) Blocos com mais de uma linha

Um bloco pode conter várias instruções:

```
x = 10
if x > 5:
    print("x é maior que 5")
    print("Essa é a segunda linha do mesmo bloco")
```

O bloco acaba quando a indentação muda (voltamos a escrever mais à esquerda).

---

6) Exemplo real: cálculo de imposto progressivo

Um caso comum em programação é calcular valores por **faixas**. Exemplo: imposto de renda simplificado.

```
salario = float(input("Digite o salário para cálculo do imposto: "))

base = salario
imposto = 0

if base > 3000:
    imposto = imposto + ((base - 3000) * 0.35)
    base = 3000

if base > 1000:
    imposto = imposto + ((base - 1000) * 0.20)

print("Salário: R$%6.2f Imposto a pagar: R$%6.2f" % (salario, imposto))
```

Aqui usamos uma variável **auxiliar (`base`)** para não perder o valor original do salário.

---

7) O `else`: e se a condição for falsa?

Até agora, vimos que o `if` executa um bloco **somente quando a condição é verdadeira**. Mas e quando a condição é **falsa**? Muitas vezes também queremos executar **uma ação alternativa**. Para isso usamos o `else`.

🔹 Como funciona

O `else` significa **"senão"** em português. Sua estrutura é:

```
if <condição>:
    bloco_se_verdadeiro
else:
    bloco_se_falso
```

- Se a condição for **True**, o Python executa o primeiro bloco.
- Se a condição for **False**, o Python ignora o primeiro bloco e executa o bloco do `else`.

🔹 Exemplo prático: idade do carro

```

```

```
idade = int(input("Digite a idade de seu carro: "))

if idade <= 3:
    print("Seu carro é novo")
else:
    print("Seu carro é velho")
```

Explicação passo a passo:

1. O programa pede a idade do carro.
2. Verifica a condição `idade <= 3`.
    - Se for **verdadeira**, imprime `"Seu carro é novo"`.
    - Caso contrário (**falsa**), imprime `"Seu carro é velho"`.

Dessa forma, garantimos que **uma das mensagens sempre será exibida**, diferente do exemplo anterior (sem `else`), onde era possível não imprimir nada se as condições não fossem satisfeitas.

🔹 Por que usar o `else`?

1. **Clareza**: deixa o código mais fácil de ler, já que mostra claramente a ação alternativa.
2. **Evita duplicação**: não precisamos escrever outro `if` para a condição contrária.
3. **Garantia de saída**: com `else`, sempre haverá uma resposta, independentemente do valor.

🔹 Atenção ao alinhamento

No Python, a indentação é fundamental:

- O `else` deve estar **na mesma coluna do `if`**.
- Se estiver mal alinhado, o interpretador dará erro.

Exemplo incorreto (gera erro):

```
if idade <= 3:
    print("Seu carro é novo")
  else:   # ERRO: alinhamento incorreto
    print("Seu carro é velho")
```

Resumo

- O `else` é usado para definir o que acontece **quando a condição do `if` é falsa**.
- Ele deixa o programa mais **curto, claro e confiável**.
- Sempre escreva `if` e `else` **alinhados na mesma coluna**.

---

8) Estruturas aninhadas

Até agora vimos o `if` e o `else` funcionando sozinhos. Mas, em muitas situações, uma **única verificação não é suficiente**: precisamos tomar uma decisão **dentro de outra decisão**. É aí que entra o **aninhamento** (colocar um `if` dentro de outro).

🔹 O que significa aninhar?

- **Aninhar** = colocar uma estrutura dentro de outra.
- No caso do `if`, significa que a execução do segundo `if` **só acontece se o primeiro já tiver sido avaliado**.
- Visualmente, isso cria mais **níveis de indentação** (mais espaços para a direita).

🔹 Exemplo prático: conta de celular

Suponha que uma operadora cobre tarifas diferentes de acordo com a quantidade de minutos usados no mês:

- Menos de 200 minutos → R$ 0,20 por minuto
- Entre 200 e 400 minutos → R$ 0,18 por minuto
- Mais de 400 minutos → R$ 0,15 por minuto

Código em Python:

```
minutos = int(input("Quantos minutos você utilizou este mês: "))

if minutos < 200:
    preco = 0.20
else:
    if minutos < 400:
        preco = 0.18
    else:
        preco = 0.15

print("Você vai pagar este mês: R$%6.2f" % (minutos * preco))
```

🔹 Como o programa pensa

1. O usuário digita os minutos.
2. O programa verifica a primeira condição: `minutos < 200`.
    - Se for **verdadeiro**, define `preco = 0.20`.
    - Se for **falso**, entra no bloco do `else`.
3. Dentro do `else`, há outro `if`: `minutos < 400`.

- Se for **verdadeiro**, define `preco = 0.18`.
- Caso contrário, entra no `else` final e define `preco = 0.15`.

4. No fim, calcula o valor da conta multiplicando minutos × preço.

🔹 Problema do aninhamento

Embora funcione bem, o **aninhamento pode deixar o código confuso**:

- Cada novo `if` exige **mais recuo (indentação)**.
- O programa fica com formato de **escada**, difícil de acompanhar.
- Em problemas com muitas categorias, pode virar um "labirinto" de `if` e `else`.

Exemplo exagerado:

```
if cond1:
    if cond2:
        if cond3:
            if cond4:
                print("Chegamos ao fundo da escada!")
```

Resumo

- Estruturas aninhadas permitem tomar **decisões dentro de outras decisões**.
- São úteis, mas devem ser usadas com cuidado.
- Muitos níveis de aninhamento tornam o código difícil de ler.
- Mais adiante veremos como o `elif` ajuda a **evitar excesso de aninhamento** e deixar o código mais limpo.

---

9) O `elif`: deixando o código mais limpo

Quando precisamos testar **múltiplas condições**, poderíamos escrever vários `if` ou usar **ifs aninhados**. Mas isso rapidamente deixa o código cheio de escadas e difícil de entender.

Para resolver esse problema, o Python oferece o **`elif`**, que significa **"else if"** (ou "senão, se...").

🔹 Estrutura do `elif`

A estrutura básica é:

```
if condição1:
    bloco1
elif condição2:
    bloco2
elif condição3:
```

```
    bloco3
else:
    bloco_final
```

- O Python avalia as condições **de cima para baixo**.
- Assim que encontrar uma condição **verdadeira**, executa o bloco correspondente e **ignora os demais**.
- Se nenhuma condição for verdadeira, cai no `else` (se existir).

🔹 Exemplo prático: categorias de produtos

Imaginemos uma tabela de preços:

Categoria	Preço
1	R$ 10
2	R$ 18
3	R$ 23
4	R$ 26
5	R$ 31

Código em Python:

```
categoria = int(input("Digite a categoria do produto: "))

if categoria == 1:
    preco = 10
elif categoria == 2:
    preco = 18
elif categoria == 3:
    preco = 23
elif categoria == 4:
    preco = 26
elif categoria == 5:
    preco = 31
else:
    print("Categoria inválida, digite um valor entre 1 e 5!")
    preco = 0

print("O preço do produto é: R$%6.2f" % preco)
```

🔹 Como o programa funciona

1. O usuário digita a categoria.
2. O Python testa a primeira condição (`categoria == 1`).

- Se for verdadeira, define `preco = 10` e **não avalia as outras**.

3. Caso contrário, passa para o próximo `elif`.
4. Se nenhuma condição for satisfeita, cai no `else` e mostra mensagem de erro.

🔹 Vantagem do `elif`

Comparemos:

Com aninhamento (ruim):

```
if categoria == 1:
    preco = 10
else:
    if categoria == 2:
        preco = 18
    else:
        if categoria == 3:
            preco = 23
        # ... e assim por diante
```

Com `elif` (melhor):

```
if categoria == 1:
    preco = 10
elif categoria == 2:
    preco = 18
elif categoria == 3:
    preco = 23
```

O código fica **mais curto, mais legível e menos propenso a erros de indentação**.

Resumo

- Use `elif` quando precisar verificar **várias condições diferentes**.
- O Python testa as condições de cima para baixo e executa apenas o primeiro bloco verdadeiro.
- O `else` final é opcional, mas útil para tratar casos não previstos (ex.: entradas inválidas).
- O `elif` deixa o código **limpo e direto**, evitando “escadas” de `if` aninhados.

Resumindo os conceitos

- `if` → executa um bloco **se** a condição for verdadeira.
- `else` → executa o bloco **quando o if é falso**.

- `elif` → cria múltiplas condições sem precisar de aninhamentos exagerados.
- Blocos no Python são definidos por **indentação** (recuo).
- Condições permitem ao programa **tomar decisões** e mudar o fluxo da execução.
