4 - Estruturas Condicionais

O que são?

As estruturas condicionais são um conceito fundamental na lógica de programação. Elas permitem que os programas tomem decisões e executem ações diferentes com base em condições específicas, proporcionando maior flexibilidade e controle sobre o fluxo de execução.

Essas condições são baseadas em expressões booleanas, que podem retornar True (Verdadeiro) ou False (Falso). Dessa forma, podemos definir o que o programa deve executar se o resultado for verdadeiro e o que ele deve fazer se for falso.

As estruturas condicionais mais comuns são o if (se), que executa um bloco de código se a condição for verdadeira, e o else (senão), que executa um bloco de código alternativo se a condição for falsa.

Além disso, para comparar essas estruturas condicionais, utilizamos os operadores lógicos e comparativos.

4.1 - If

Para criar uma estrutura condicional simples, composta por um único desvio, podemos utilizar a palavra reservada if. O comando irá testar a expressão lógica, e em caso de retorno verdadeiro as ações presentes no bloco de código do if serão executadas.

Sintaxe:

```
if condição:
   #código executado se a condição for verdadeira
```

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

Explicação:

Neste exemplo, o programa verifica se a variável idade é maior ou igual a 18. Se for, ele imprime a mensagem.

Dica:

Lembre-se que em Python a indentação (espaço à esquerda do código) é obrigatória para indicar que um bloco pertence ao if.

```
#idade = int(input("Digite sua idade:"))
idade = 61
if idade > 18:
  print("Você já tem mais de 18 anos") if
idade > 60:
  print("Você já é idoso")
```

4.2 – Else

Para criar uma estrutura condicional com dois desvios, podemos utilizar as palavras reservadas if e else. Como sabemos se a expressão lógica testada no if for verdadeira, então o bloco de código do if será executado. Caso contrário o bloco de código do else será executado.

Sintaxe:

```
if condição:
   # código se
verdadeiro else:
   # código se falso

idade = 17
if idade > 18:
  print("Você já é maior de idade")
else:
  print("Você não é maior de idade")
```

4.3 - Ifelse

Em alguns cenários queremos mais de dois desvios, para isso podemos utilizar a palavra reservada elif. O elif é composto por uma nova expressão lógica, que será testada e caso retorne verdadeiro o bloco de código do elif será executado. Não existe um número máximo de elifs que podemos utilizar, porém evite criar grandes estruturas condicionais, pois elas aumentam a complexidade do código.

Sintaxe:

```
if condição1:
   # código se condição1 for verdadeira
elif condição2:
   # código se condição2 for verdadeira
else:
   # código se todas forem falsas


idade = 6
if idade >= 18:
  print("Você já tem mais de 18 anos")
elif idade < 18:
  print("Você é uma criança")
else:
   print("Você não é maior de idade")
```

Dica:

Use elif para evitar criar muitos if separados, o que pode deixar o código confuso.

4.4 - If aninhado

Podemos criar estruturas condicionais aninhadas, para isso basta adicionar estruturas if/elif/else dentro do bloco de código de estruturas if/elif/else.

```
idade = 20
ingresso = False
if idade >= 18:
    if ingresso:
        print("Entrada permitida")
    else:
        print("Compre um ingresso")
else:
    print("Você não pode entrar")
```

4.4 - If Ternário (operador ternário)

O if ternário permite escrever uma condição em uma única linha. Ele é composto por três partes, a primeira parte é o retorno caso a expressão retorne verdadeiro, a segunda parte é a expressão lógica e a terceira parte é o retorno caso a expressão não seja atendida.

Sintaxe:

```
valor = resultado1 if condicao else resultado2
```

```
idade = 20
# Usando o if ternário
status = "Maior de idade" if idade >= 18 else "Menor de idade"
print(status)  # Saída: Maior de idade


numero = int(input("Digite um número:"))
par_ou_impar = ["Par" if numero % 2 == 0 else "Ímpar"]
print(par_ou_impar)  # Saída: ['Ímpar', 'Par', 'Ímpar', 'Par', 'Ímpar']
```

4.5 - Operadores Lógicos

Operadores Lógicos em si não são condicionais mas são usados para combinar condições em Python. Eles ajudam a construir expressões mais complexas ao trabalhar com condicionais. Há três principais operadores lógicos:

- **4.5.1 and (e) :**

  O operador and retorna True se todas as condições forem verdadeiras. Se alguma condição for falsa, o resultado será False.

```
idade = 25
renda = 3000

if idade >= 18 and renda >= 2500:
   print("Aprovado para o empréstimo!")
else:
   print("Não atende aos requisitos.")
```

- **4.5.2 or (ou) :**

  O operador or retorna True se pelo menos uma das condições for verdadeira. Retorna False apenas se todas forem falsas.

```
idade = 16
renda = 4000

if idade >= 18 or renda >= 2500:
   print("Pode receber uma proposta especial.")
else:
   print("Não atende aos critérios.")
```

- **4.5.3 not (não)** :

  O operador not inverte o valor lógico da condição. Se a condição for True, not a transforma em False, e vice-versa.

```
tem_divida = False

if not tem_divida:
   print("Pode solicitar um novo empréstimo.")
else:
   print("Resolva suas dívidas primeiro.")
```

a	b	a and b	a or b
False	False	False	False
True	False	False	True
False	True	False	True
True	True	True	True

a	not a
True	False
False	True

Exercício 1: Calculadora simples

Enunciado: Faça um programa em que o usuário forneça dois números inteiros e uma operação matemática (+, -, *, /) e o código exiba o resultado da operação.

```
numero1 = int(input("Digite um numero inteiro:"))
numero2 = int(input("Digite outro numero inteiro:"))
operacao = input("Digite uma operação matemática: (+;-;*;/)")
if operacao == '-':
  resultado = numero1 - numero2
  print("O resultado da operação é ",resultado)
elif operacao == '+':
  resultado = numero1 + numero2
  print("O resultado da operação é ",resultado)
elif operacao == '*':
  resultado = numero1 * numero2
  print("O resultado da operação é ",resultado)
elif operacao == '/':
  resultado = numero1 / numero2
  print("O resultado da operação é",resultado)
else:
  print("Operador inválido")
```

Exercício 2: Nota final

Enunciado: Peça ao usuário para inserir sua nota de uma prova e exiba a classificação:

- Nota maior ou igual a 90: "Excelente"
- Nota entre 70 e 89: "Bom"
- Nota menor que 70: "Precisa melhorar"

```
nota = int(input("Digite sua nota:"))

if nota >= 90:
  print("Excelente")
elif nota >= 70 and nota <=89:
  print("Bom")
elif nota < 70:
  print("Precisa melhorar")
```

5 - Estruturas de Repetição (loops)

As estruturas de repetição são fundamentais na programação, permitindo que um bloco de código seja executado várias vezes sem precisar ser reescrito manualmente. Isso é útil quando queremos automatizar tarefas repetitivas ou quando precisamos
processar grandes volumes de dados.

- **O que são estruturas de repetição?**

Em Python, as estruturas de repetição são usadas para executar um conjunto de instruções enquanto uma condição for verdadeira (ou por um número específico de vezes). As principais estruturas de repetição em Python são:

5.1 O Loop while:

O loop while repete um bloco de código enquanto uma condição for verdadeira. Ele é ideal para situações em que não sabemos, de antemão, quantas vezes o loop será executado, mas temos uma condição para encerrar o processo.

Sintaxe

```
while condição:
   # bloco de código que será repetido
```

Exemplo

Vamos criar um programa que imprime os números de 1 a 5:

Manualmente:

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

Com o Loop while:

```
contador = 1
while contador <= 5:
   print(contador)
   contador += 1  # Incrementa o contador em 1 a cada repetição
```

Exemplo com condição de parada

Um loop while também pode ser usado para verificar uma condição de entrada. Por exemplo:

```
resposta = ""
while resposta != "sair":
   resposta = input("Digite algo (ou 'sair' para encerrar): ")
   print("Você digitou:", resposta)
```

Exemplo:

Faça um código que exiba o resultado de S:

$$S = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + ... + \frac{99}{50}$$

```
a = 1
b = 1
s = 0

while a<= 99 and b<=50:
   s = s+(a/b)
   a = a+2
   b = b+1
x = round(s,2) #round função de aproximação de casas decimais
print(x)
```

5.2 O Loop for:

O loop for é mais usado quando sabemos quantas vezes queremos que o código seja repetido ou queremos iterar sobre uma lista, intervalo ou sequência.

Sintaxe

```
for variável in sequência:
   # bloco de código que será repetido
```

- variável: Uma variável que assume o valor de cada elemento da sequência em cada iteração.
- sequência: A sequência de elementos sobre a qual você deseja iterar (lista, string, etc.).

```
for i in range(1,6,1):
  print(i)

# Iterando sobre uma lista
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
  print(fruta)

# Iterando sobre uma string
for letra in "Python":
  print(letra)

# Iterando sobre um range de
números for i in range(5):
  print(i)
```

Função range()

A função range() é muito utilizada com o loop for para gerar uma sequência de números. Ela possui três parâmetros opcionais:

```
range(start, stop, step)
```

- start: O valor inicial da sequência(default:0).
- stop: O valor final da sequência (não incluído).
- step: O incremento entre cada valor (default: 1).

```
for i in range(2, 10, 2):
  print(i)  # Imprime os números pares de 2 a 8
```

Resumo:

- **while:** Ideal quando você não sabe o número de iterações antecipadamente, mas tem uma condição de parada.
- **for:** Ideal quando você sabe o número exato de repetições ou quando deseja iterar sobre uma sequência (como uma lista, string, etc.).

EXEMPLO:

Vamos criar um código que soma números inteiros a partir de 1 até alcançar um total de 100. Quando o total chega a 100, o loop é interrompido.

```
total = 0
numero = 1

while total < 100:
   total += numero # total = total +
   numero numero += 1 # numero =
   numero + 1
   print(f"Número: {numero - 1}, Total: {total}")
```

EXEMPLO: JOGO DE ADIVINHAÇÃO:

Crie um jogo onde o computador escolhe um número aleatório entre 1 e 100 e o jogador tenta adivinhar. Utilize um loop while para continuar o jogo até o jogador acertar. Dê dicas ao jogador, como "O número é maior" ou "O número é menor".

```
import random #Este módulo fornece funções para gerar números

aleatórios numero_secreto = random.randint(1, 100)
tentativas = 0

print("Bem-vindo ao jogo da adivinhação!")
print("Tente adivinhar o número entre 1 e 100.")

while True:
  chute = int(input("Digite seu chute: "))
  tentativas += 1

  if chute < numero_secreto:
      print("O número secreto é maior.")
  elif chute > numero_secreto:
      print("O número secreto é menor.")
  else:
     print(f"Parabéns! Você acertou em {tentativas} tentativas!")
     break
```

EXERCÍCIOS:

- **Imprimindo números:**

1. Crie um programa que utilize um loop for para imprimir todos os números pares de 0 a 100.
2. Modifique o programa para imprimir os números ímpares de 1 a 99 utilizando um loop while.
3. Crie um programa que utilize um loop for aninhado para imprimir a tabuada do 2 ao 9.

- **Fatorial:**

4. Crie um programa que calcule o fatorial de um número utilizando um loop for. O usuário deve informar o número.

- **Soma dos números:**

5. Crie um programa que peça ao usuário para digitar vários números e calcule a soma deles utilizando um loop while. O programa deve parar quando o usuário digitar 0.

- **Números primos:**

6. Crie um programa que verifique se um número é primo. Utilize um loop for para verificar se o número é divisível por algum número entre 2 e a sua raiz quadrada.

- **Contagem de vogais:**

7. Crie um programa que peça ao usuário para digitar uma frase e conte quantas vogais existem nessa frase utilizando um loop for.

- **Par ou Ímpar:**

8. Peça ao usuário para digitar um número e verifique se ele é par ou ímpar.

- **Máximo e Mínimo:**

9. Peça ao usuário para digitar 5 números e exiba o maior e o menor entre eles.

- **Tabuáda:**

10. Peça ao usuário para digitar um número e imprima a tabuada desse número de 1 a 10.
