Revisão de Python: Esctruturas de Controle

> Estruturas de controle são usadas para controlar o fluxo de execução do programa.

- **Estruturas Condicionais:** `if`, `elif`, `else`, `match...case`
- **Estruturas de Repetição:** `for`, `while`
- **Estruturas de Controle de Loop:** `break`, `continue`
- **Estruturas de Controle de Função:** `return`, `yield`, `pass`
- **Estruturas de Controle de Exceção:** `try`, `except`, `finally`
- **Estruturas de Controle de Contexto:** `with`

if...else

```
x = 15
if x > 5:
   x = 5
print(x)
```

```
idade = 18
if idade >= 18:
    print("Maior de Idade")
else:
    print("Menor de Idade")
```

elif

```
idade = 18
if idade < 18:
    print("Criança")
elif idade < 60:
    print("Adulto")
else:
    print("Idoso")
```

Exercícios

1. Escreva um programa que leia um número e exiba se ele é positivo, negativo ou zero.
2. Escreva um programa que leia um número e exiba se ele é par ou ímpar.
3. Escreva um programa que leia um número e exiba se ele é múltiplo de 3 e de 5.
4. Escreva um programa que leia um número e exiba se ele é múltiplo de 3 e de 5, de 3, de 5 ou de nenhum.
5. Escreva um programa que leia três valores de lados e exiba se eles formam um triângulo e, se sim, se é equilátero, isósceles ou escaleno.

while

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

for

```
for i in range(5):
    print(i)
```

- Se a variável de controle não for usada no bloco de código, pode ser substituída por um sublinhado `_`.

```
for _ in range(5):
    print("Olá, Mundo!")
```

Função Range

> `range` é uma função que gera uma sequência de números, ela pode receber um, dois ou três argumentos.

- Se receber um argumento, gera uma sequência de 0 até o número anterior ao argumento.
- Se receber dois argumentos, gera uma sequência do primeiro argumento até o número anterior ao segundo argumento.
- Se receber três argumentos, gera uma sequência do primeiro argumento até o número anterior ao segundo argumento, com um intervalo definido pelo terceiro argumento.

- Exemplos

```
for i in range(5):
    print(i)
```

Saida: `0 1 2 3 4`

```
for i in range(2, 5):
    print(i)
```

Saida: `2 3 4`

```
for i in range(1, 10, 2):
    print(i)
```

Saida: `1 3 5 7 9`

- Exemplo com incremento negativo

```
for i in range(5, 0, -1):
    print(i)
```

Saida: `5 4 3 2 1`

> **for** é usado para iterar sobre uma sequência (como uma lista, tupla, dicionário, conjunto ou string) ou outros objetos iteráveis. Veremos mais sobre isso em aulas futuras.

break e continue

- **break:** Interrompe a execução do loop.

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

- **continue:** Interrompe a execução atual do loop e continua com a próxima iteração.

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

match...case

- A partir do Python 3.10, foi introduzida a estrutura `match...case` para substituir o `if...elif...else` em situações específicas.

```
x = 5
match x:
    case 1:
        print("Um")
    case 2:
        print("Dois")
    case 3:
        print("Três")
    case _:
        print("Outro")
```

- A estrutura `match...case` é mais eficiente que o `if...elif...else` em situações onde há muitas condições.

For...Else

- O bloco `else` é executado quando o loop termina **sem interrupção**.

```
x = 113
for i in range(2, x):
    if x % i == 0:
        print("Não é primo")
        break
else:
    print("É primo")
```

- Também pode ser usado com `while`.

> Como se comport a o for...else em relação ao `continue`?

```
for i in range(5):
    if i == 3:
        continue
    print(i)
else:
    print("Fim")
```

> Python não possui do...while.

- Exemplo de simulação do...while

```
while True:
    x = int(input("Digite um número: "))
    x += 1
    if x == 5:
        break
```

Execìcios

1. Escreva um programa que exiba os números pares de 1 a 100.
2. Escreva um programa que exiba os números de 1 a 100, exceto o 5.
3. Escreva um programa que exiba os números de 100 a 1, em ordem decrescente.
4. Escreva um programa que calcule a soma dos números de 1 a 100
    i. usando a fórmula da soma de uma progressão aritmética.
    ii. usando um loop, somando os números um a um.

Pass

- A instrução `pass` é usada para criar um bloco de código vazio.
- É útil quando a sintaxe exige um bloco de código, mas a lógica do programa não.

```
x = 5
if x > 5:
    pass
else:
    print(x)
```

Try...Except

- A instrução `try` é usada para testar um bloco de código.
- A instrução `except` é usada para lidar com exceções.
- A instrução `finally` é usada para executar código, independentemente de haver uma exceção.

```
try:
    x = 5 / 0
except ZeroDivisionError:
    print("Erro de divisão por zero")
finally:
    print("Fim do programa")
```

Exceções

> Exceções são usadas para lidar com erros e situações que ocorrem raramente, mas que podem ocorrer durante a execução de um programa.

- Python possui muitos tipos de exceções embutidas, como:
  - `ZeroDivisionError`

```
try:
    x = 5 / 0
except ZeroDivisionError:
    print("Erro de divisão por zero")
```

- `ValueError`

```
try:
    x = int("abc")
except ValueError:
    print("Erro de valor inválido")
```

- `TypeError`

```
try:
    x = 5 + "abc"
except TypeError:
    print("Erro de tipo inválido")
```

- `NameError`

```
try:
    x = y
except NameError:
    print("Erro de nome inválido")
```

- `IndexError`

```
try:
    x = [1, 2, 3]
    print(x[3])
except IndexError:
    print("Erro de índice inválido")
```

- **Levantando Exceções:** É possível levantar exceções manualmente usando a instrução `raise`.

```
def fat(n):
    if n < 0:
        raise ValueError("O argumento não pode ser negativo")
    f = 1
    for i in range(1, n + 1):
        f *= i
    return f
```

- **Exceções Personalizadas:** É possível criar exceções personalizadas herdando da classe `Exception`.

```
class MeuErro(Exception):
    pass

#...
raise MeuErro("Mensagem de erro")
```

Assert

- A instrução `assert` é usada para certificar-se de que uma condição é verdadeira.
- Se a condição for falsa, a instrução `assert` levanta uma exceção `AssertionError`.

```
def fat(n):
    if n < 0:
        raise ValueError("O argumento deve ser positivo")
    f = 1
    for i in range(1, n + 1):
        f *= i
    assert f > 0, "O resultado deve ser positivo"
    return f
```

- Principalmente usado para testes e depuração.
- Pode ser desativado em tempo de execução com a opção `-O`.
- O programador deve assumir que, na versão final do programa, as asserções não serão verificadas.

Exercícios

1. Escreva um programa que ler do usuários vários números inteiros não negativos e exiba, ao final, a média dos números lidos. Quando o usuário digitar um número negativo, o programa para de ler números e exibe a média. Se o usuário não digitar qualquer número válido, o programa exibe a mensagem "Nenhum número foi digitado". Se o usuário digitar algo que não seja um número, o programa exibe a mensagem "Valor inválido", mas continua lendo os números.