Try / Except

Try / Except

▼ Estrutura básica

> ? O bloco `try-except` é usado para capturar e tratar erros (exceções) durante a execução de um programa. Quando um erro ocorre dentro do bloco `try`, a execução do programa não é interrompida abruptamente; em vez disso, o fluxo do código pode ser controlado por um bloco `except`.

```
try:
    # Código que pode gerar um erro
except TipoDeErro:
    # Código para tratar o erro
```

▼ Capturar erro específico

- Exemplo para um erro específico:

```
try:
    div = 8/0
except ZeroDivisionError:
    print("Erro: Divisão por zero!")
```

```
# Exibe: Erro: Erro: Divisão por zero!
```

- Exemplo para múltiplos erros específicos:

```
try:
    lista = [1, 2, 3]
    print(lista[5])  # Isso gera IndexError
except ZeroDivisionError:
    print("Erro: Divisão por zero!")
except IndexError:
    print("Erro: Índice fora do alcance da lista!")

# Exibe: Erro: Índice fora do alcance da lista!
```

- Exemplo para tratar dois erros na mesma exceção:

```
try:
    num = int(input("Digite um número: "))
    resultado = 10 / num
except (ZeroDivisionError, ValueError) as erro:
    print("Erro: Divisão por zero!")
except IndexError:
    print(f"Ocorreu um erro: {erro.__class__.__name__} - {e}")
```

  - ZeroDivisionError: é exibido caso `num = 0`, retornando um erro de divisão por 0
  - ValueError: acontece quando o tipo do dado está certo, mas o valor não faz sentido para a operação. Exemplo: `num = int ( "abc" )`

▼ Try / else / finally (**else** e **finally**)

- `else`: Executa se **nenhuma** exceção for levantada.
- `finally`: Sempre executa, independentemente de ocorrer erro ou não.
- Exemplo:

```
try:
    num = int(input("Digite um número: "))
```

```
    print(f"Você digitou: {num}")
except ValueError:
    print("Erro: Entrada inválida, digite um número inteiro.")
else:
    print("Nenhum erro ocorreu.")
finally:
    print("Fim da execução.")
```

▼ Obter nome da exceção (**as**)

- O `as` é usado para atribuir um nome a um objeto dentro de um `except`, permitindo acessar a exceção capturada para obter detalhes sobre o erro.
- Exemplo: Usando `as` para acessar a mensagem da exceção

```
try:
    x = int("abc")  # Isso causa um ValueError
except ValueError as erro:
    print(f"Ocorreu um erro do tipo: {erro}")

# Ocorreu um erro: invalid literal for int() with base 10: 'abc'
```

  - Explicação: `as erro` armazena a exceção na variável `erro`, permitindo acessar a mensagem associada a ela.

- Exemplo: Usando `as` para obter o nome da exceção:

```
try:
    x = 10 / 0  # Gera ZeroDivisionError
except Exception as erro:
    print(f"Ocorreu um erro: {e.__class__.__name__} - {erro}")

# Ocorreu um erro: ZeroDivisionError - division by zero
```

  - Explicação: `erro.__class__.__name__` retorna o nome da exceção, enquanto `erro` exibe a mensagem do erro.

- Exemplo 3: Usando `as` com múltiplas exceções

```
try:
    lista = [1, 2, 3]
    print(lista[5])  # Isso gera IndexError
except (ZeroDivisionError, IndexError) as erro:
    print(f"Ocorreu um erro: {e.__class__.__name__} - {erro}")

# Erro: IndexError - list index out of range
```

    - Explicação: O `except` captura dois tipos de erro e armazena o erro ocorrido na variável `erro`.

▼ Obter nome da exceção (**`__class__.__name__`**)

- É possível capturar uma exceção e obter o nome da sua classe usando `__class__.__name__`. Isso é útil para exibir dinamicamente o tipo do erro sem precisar tratá-lo de forma específica.
- Exemplo:

```
try:
    x = int("abc")  # Isso causa um ValueError
except Exception as erro:
    print(f"Ocorreu um erro do tipo: {erro.__class__.__name__}")
    # Exibe: Ocorreu um erro do tipo: ValueError
```

▼ Toda exceção é uma classe + **class Exception**

- Podemos usar a classe **`Exception`** para capturar qualquer erro, independentemente do tipo. Isso é útil quando não sabemos exatamente quais exceções podem ocorrer. As demais exceções são subclasses de **`Exception`**, herdando suas características.
- Exemplo de uso:

```
try:
    num = int(input("Digite um número: "))
    resultado = 10 / num  # Pode gerar ZeroDivisionError
except Exception as erro:
    print(f"Ocorreu um erro: {e.__class__.__name__} - {erro}")
```

▼ Lançando exceções (**raise**)

- O `raise` é usado para lançar exceções manualmente. Ele permite interromper a execução normal do programa e indicar que ocorreu um erro, que pode ser tratado com um bloco `try-except`.
- Sintaxe: `raise` `Exception` `(` `"Mensagem de erro"` `)`
- Exemplo para lançar erro simples:

```
def dividir(a, b):
    if b == 0:
        raise ZeroDivisionError("Não é possível dividir por zero.")
    return a / b

print(dividir(10, 2))  # Exibe: 5.0
print(dividir(10, 0))  # Gera um erro ZeroDivisionError
```

- Exemplo usando `raise` dentro de um `try-except`:

```
try:
    raise ValueError("Este é um erro de valor.")
except ValueError as erro:
    print(f"Erro capturado: {erro}")
# Exibe: Este é um erro de valor.
```

- Exemplo de `raise` sendo usado para tratar erros:

```
def nao_aceito_zero(d):
    if d == 0:
        raise ZeroDivisionError('Você está tentando dividir por zero')
    return True

def deve_ser_int_ou_float(n):
    tipo_n = type(n)
    if not isinstance(n, (float, int)):
        raise TypeError(
            f'"{n}" deve ser int ou float. '
            f'"{tipo_n.__name__}" enviado.'
        )
```

```
    return True

def divide(n, d):
    deve_ser_int_ou_float(n)
    deve_ser_int_ou_float(d)
    nao_aceito_zero(d)
    return n / d

print(divide(8, '0'))
```