TRATAMENTO DE ERROS E EXCEÇÕES

Erros em Python

- Os erros ocorridos em programas Python podem ser de dois tipos:
  - **Erros de sintaxe**
  - **Exceções**

Erros de Sintaxe

- Como o nome sugere, esse erro é causado pela *__sintaxe incorreta__* no código.
- Provoca o *__encerramento__* do programa.

In [22]:
```
valor = 10000

if(valor > 2999)
    print("Valor acima do limite".)

# Ocorre erro de sintaxe devido a ausencia de dois-pontos
```

```
  Cell In[22], line 3
    if(valor > 2999)
                    ^
SyntaxError: expected ':'
```

Exceções

- Exceções ocorrem quando o programa está sintaticamente correto, mas o código resulta em um *__erro__*.
- São erros ocorridos em *__tempo de execução__*.
- Uma exceção interrompe a execução do programa.

In [23]:
```
valor = 10000

total = valor / 0
print(total)

# Ocorre uma exceção ao executar uma divisão por zero
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[23], line 3
      1 valor = 10000
----> 3 total = valor / 0
      4 print(total)
      6 # Ocorre uma exceção ao executar uma divisão por zero

ZeroDivisionError: division by zero
```

Diferentes tipos de Exceções

- Existem várias tipos de exceções que podem ser geradas em um programa.
- Aqui estão alguns dos tipos mais comuns de exceções em Python:
  - `TypeError` : quando uma operação ou função é aplicada a um objeto do tipo errado, como adicionar uma string a um inteiro.
  - `NameError` : quando um nome de variável ou função não é encontrado no escopo atual.
  - `ValueError` : quando uma função ou método é chamado com um parâmetro ou entrada inválido, como tentar converter uma string em um inteiro quando a string não representa um inteiro válido.
  - `ZeroDivisionError` : quando é feita uma tentativa de dividir um número por zero.

Tratamento de Exceções

- Um programa bem desenvolvido deve detectar e *__tratar__* suas possíveis exceções.
- Ao tratar uma exceção, podemos lidar com os erros de forma elegante e *__evitar que o programa falhe__*.

Exemplo

- `TypeError` : Esta exceção ocorre quando uma operação ou função é aplicada a um objeto do tipo errado.
- Aqui está um exemplo:

In [24]:
```
x = 5
y = "hello"
z = x + y       # provoca um TypeError
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[24], line 3
      1 x = 5
      2 y = "hello"
----> 3 z = x + y       # provoca um TypeError

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Instrução Try e Except

- As instruções `try` e `except` são usadas para capturar e manipular exceções em Python.
  - As instruções que podem gerar exceções são mantidas dentro do bloco `try` .
  - As instruções que lidam com a exceção são escritas dentro do bloco `except` .

In [25]:
```
try:
    x = 5
    y = "hello"
    z = x + y
```

```
except TypeError:
    print("Ocorreu um erro, pois não é possível somar uma string com um inteiro")
```

```
Ocorreu um erro, pois não é possível somar uma string com um inteiro
```

Exemplo

In [26]:
```
try:
    a = int(input('Informe um número: '))
    b = int(input('Informe um número: '))
    c = a / b
    print('Resultado da Divisão:', c)
except ZeroDivisionError:
    print("Não é possível fazer uma divisão por zero.")
```

```
Informe um número: 10
Informe um número: 0
Não é possível fazer uma divisão por zero.
```

Capturando exceções específicas

- Uma instrução `try` pode ter mais de uma cláusula `except`.
- Um `except` para cada tipo de exceção que será tratada

In [27]:
```
try:
    a = int(input('Informe um número: '))
    b = int(input('Informe um número: '))
    c = a / b
    print('Resultado da Divisão:', c)
except ZeroDivisionError:
    print("Não é possível fazer uma divisão por zero.")
except ValueError:
    print("O valor informado não é um número inteiro.")
```

```
Informe um número: 30
Informe um número: 2.5
O valor informado não é um número inteiro.
```

Exceção do tipo Exception

- Se quisermos tratar qualquer tipo de erro "**genérico**", podemos inserir o tipo de exceção `Exception`
  - Captura qualquer exceção.

In [28]:
```
try:
    a = int(input('Informe um número: '))
    b = int(input('Informe um número: '))
    c = a / n                          # Foi utilizado a variável n, que não definida
    print('Resultado da Divisão:', c)
except ZeroDivisionError:
    print('Não é possível fazer uma divisão por zero.')
except ValueError:
    print('O valor informado não é um número inteiro.')
except Exception:
    print('Um erro inesperado aconteceu.')
```

```
Informe um número: 30
Informe um número: 2
Um erro inesperado aconteceu.
```

Exceções com else

- É possível completar um comando `try` com um bloco `else` que introduz um trecho de código que será executado quando *__nenhuma exceção__* ocorre.

In [29]:
```
try:
    a = int(input('Informe um número: '))
    b = int(input('Informe um número: '))
    c = a / b
    print('Resultado da Divisão:', c)
except ZeroDivisionError:
    print('Não é possível fazer uma divisão por zero.')
except ValueError:
    print('O valor informado não é um número inteiro.')
else:
    print('Operação realizada com sucesso.')
```

```
Informe um número: 30
Informe um número: 2
Resultado da Divisão: 15.0
Operação realizada com sucesso.
```

Exceções com finally

- É possível também inserir um bloco `finally`.
- O bloco `finally` é executado em todas as situações, ocorrendo , ou não ocorrendo exceções.

In [30]:
```
try:
    a = int(input('Informe um número: '))
    b = int(input('Informe um número: '))
    c = a / b
    print('Resultado da Divisão:', c)
except ZeroDivisionError:
    print('Não é possível fazer uma divisão por zero.')
except ValueError:
    print('O valor informado não é um número inteiro.')
else:
    print('Operação realizada com sucesso.')
finally:
    print('Final do Programa')
```

```
Informe um número: 20
Informe um número: 2
Resultado da Divisão: 10.0
Operação realizada com sucesso.
Final do Programa
```

Gerando Exceções

- A instrução `raise` permite forçar a ocorrência de uma exceção específica.

- Permite indicar o tipo da exceção a ser gerada.

In [31]:
```
try:
    raise NameError          # gera uma exceção
except NameError:
    print ("Ocorreu uma exceção")
```

```
Ocorreu uma exceção
```

Gerando Exceções

- A instrução `raise` costuma ser utilizada para realizar validações.

In [32]:
```
try:
    a = int(input('Informe um número: '))
    b = int(input('Informe um número: '))
    c = a / b
    if a < 0 or b < 0:
        raise TypeError           # gera exceção TypeError se algum número for negati
    print('Resultado da Divisão:', c)
except ZeroDivisionError:
    print('Não é possível fazer uma divisão por zero.')
except ValueError:
    print('O valor informado não é um número inteiro.')
except TypeError:
    print('O valor informado é negativo.')
```

```
Informe um número: 40
Informe um número: -2
O valor informado é negativo.
```

Vantagens do Tratamento de Exceções:

- *__Confiabilidade aprimorada__*: ao lidar com exceções adequadamente, você pode impedir que seu programa trave ou produza resultados incorretos devido a erros ou entrada inesperados.
- *__Código mais limpo__*: com o tratamento de exceções, você pode evitar o uso de instruções condicionais complexas para verificar erros, levando a um código mais limpo e legível.
- *__Manutenção facilitada__*: o tratamento de exceções permite separar o código de tratamento de erros da lógica do programa principal, facilitando a leitura e a manutenção do código.

Desvantagens do Tratamento de Exceções

- *__Sobrecarga de desempenho__*: o interpretador do Python precisa executar trabalho adicional para capturar e manipular a exceção.
- *__Maior complexidade do código__*: o tratamento de exceções pode tornar seu código mais complexo, especialmente se você tiver que lidar com vários tipos de exceções ou implementar uma lógica complexa de tratamento de erros.