PROGRAMAÇÃO EM PYTHON

Fernando F. Santos

python

SENAI

AULA 19

31/03/2023

MENSAGENS DE ERRO

- Veremos os tipos de erros que podem ocorrer em um programa Python.
- Além do tipo de erro, o Python mostra a linha onde o interpretador foi interrompido, dando uma grande dica do que precisa ser corrigido em seu código.

SYNTAXERROR

Erros de sintaxe, também conhecidos como erros de análise, ocorrem quando o interpretador não consegue ler o que você escreveu, o que pode ser ocasionado por erros de digitação ou símbolos esquecidos. Esse é um erro muito comum para pessoas que estão iniciando os estudos em Python.

```
numero = 10
texto = "Olá, mundo

print("Número:", numero)
print("Texto:", texto)
```

```
File "../syntax_error.py", line 2
    texto = "Olá, mundo
                       ^
SyntaxError: EOL while scanning string literal
```

IDENTATIONERROR

Erro de indentação, este erro é comum entre programadores de outras linguagens quando iniciam o aprendizado com Python. Também ocorre muito quando o programador utiliza tabulação e espaços ao mesmo tempo em seu código (o que não é aconselhado).

```
File "../Identation_error.py", line 5
    print("O resultado da soma é:", x)
    ^
IndentationError: unexpected indent
```

```
def soma(n1, n2):
    return n1 + n2

x = soma(10, 78)
    print("O resultado da soma é:", x)
```

Indentação inesperada.

KEYERROR

Este erro ocorre quando tentamos acessar uma chave inexistente em um dicionário.

```
dic = {"codigo": 1, "nome": "Evaldo", "celular": "(99)99999-9999"}

print(dic["codigo"])
print(dic["nome"])
print(dic["telefone"])
```

```
1
Traceback (most recent call last):
Evaldo
  File "../key_error.py", line 5, in <module>
    print(dic["telefone"])
KeyError: 'telefone'
```

NAMEERROR

Este erro ocorre quando o interpretador Python não consegue identificar um nome fornecido. Seja uma variável, função ou qualquer outra referência que não seja identificada.

```
x = 10
y = 55

print("O resultado de x + y é:", X + y)
```

```
.../name_error.py", line 4, in <module>
    print("O resultado de x + y é:", X + y)
NameError: name 'X' is not defined
```

```
def soma(x, y):
    return x + y

resultado = somar(10, 75)
```

```
.../name_error2.py", line 4, in <module>
    resultado = somar(10, 75)
NameError: name 'somar' is not defined
```

VALUEERROR

O ValueError ocorre quando existe um problema com o conteúdo do objeto ao qual você tentou atribuir o valor.

```
print(int(5.75))
print(int("78"))
print(int("Olá"))
```

```
5
78
../value_error.py", line 3, in <module>
    print(int("Olá"))
ValueError: invalid literal for int() with base 10: 'Olá'
```

```
inteiro = 5
lista = []
lista.remove(inteiro)
```

```
.../value_error2.py", line 3, in <module>
    lista.remove(inteiro)
ValueError: list.remove(x): x not in list
```

TYPEERROR

Um tipo em Python pode ser considerado como a especificação para uma categoria de dados relacionados. Embora possamos definir nossos próprios tipos, eles tendem a cair em um conjunto de categorias pré-definidas.

Estes são os tipos mais comuns que estão incorporados na linguagem Python:

- **int** – Números inteiros
- **float** – Números decimais
- **str** – uma sequência de caracteres
- **list** – uma lista ordenada
- **tuple** – uma lista ordenada imutável
- **boolean** – verdadeiro ou falso

TYPEERROR

- Quando chamamos uma função ou usamos um operador, normalmente o interpretador Python espera ser informado parâmetros de um tipo específico. Por exemplo, o operador de adição “+” espera receber dois parâmetros de tipo numérico, ou seja, integer ou float.

```
>>> 125 + "Olá"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> _
```

```
inteiro = 30
x = inteiro[1]
```

```
.../type_error.py", line 2, in <module>
   x = inteiro[1]
TypeError: 'int' object is not subscriptable
```

INDEXERROR

Um índice em Python representa uma posição em uma sequência. Mais comumente, isso se refere a uma posição em uma string ou uma lista. A indexação em Python, como na maioria das linguagens de programação, começa com zero.

```
lista_cores = ['Branco', 'Azul', 'Verde', 'Amarelo']

print(lista_cores[0])
print(lista_cores[3])
print(lista_cores[4])
```

```
Branco
Amarelo
Traceback (most recent call last):
  File "P:/PCloud/Udemy/MeusCursos/PythonParaTodos/Secao 15 - Mensagens de
erro/codigo/index_error.py", line 5, in <module>
    print(lista_cores[4])
IndexError: list index out of range

Process finished with exit code 1
```

ATTRIBUTEERROR

Um atributo em Python é uma propriedade associada a um determinado tipo de objeto. Em outras palavras, os atributos de um determinado objeto são os dados e habilidades que cada tipo de objeto possui inerentemente.

```
class teste(object):
    def __init__(self, nome, telefone):
        self.nome = nome
        self.telefone = telefone

t = teste("João Silva", "(00)00000-0000")
print(t.nome)
print(t.telefone)
print(t.idade)
```

```
.../attribute_error.py", line 9, in <module>
  print(t.idade)
AttributeError: 'teste' object has no attribute 'idade'
João Silva
(00)00000-0000

Process finished with exit code 1
```

EXCEÇÕES

Quando ocorre uma falha no programa em tempo de execução, uma exceção é gerada. Caso a exceção não seja tratada, ela se propaga por meio das chamadas de funções até o módulo principal, interrompendo a execução do programa.

EXCEÇÕES

No código ao lado, foi solicitado um número ao usuário, mas não está sendo feito nenhum tratamento caso o usuário informe uma string que não seja um número.

Veja que, ao informar a string “Olá”, gerou uma exceção do tipo ValueError.

Veja também que logo em seguida está uma divisão, e não estamos tratando caso o usuário informe zero e ocorra um erro de divisão por zero.

```
valor = input("Informe um número: ")

soma = int(valor) + 10

resultado = soma / int(valor)
```

```
“.../excecao.py"
Informe um número: Olá
Traceback (most recent call last):
  File “.../excecao.py", line 3, in <module>
    soma = int(valor) + 10
ValueError: invalid literal for int() with base 10: 'Olá'
```

```
“.../excecao.py"
Informe um número: 0
Traceback (most recent call last):
  File “.../excecao.py", line 5, in <module>
    resultado = soma / int(valor)
ZeroDivisionError: division by zero
```

TRY, EXCEPT

Para tratar exceções utilizamos a instrução try. Caso ocorra alguma exceção em um bloco tratado com try, podemos tratar esta exceção utilizando a instrução except.

Sintaxe:

```
try:
    bloco_de_código
except tipo_de_erro:
    bloco_de_código_executado_em_caso_de_excecao
```

EXEMPLO

```
valor = input("Informe um número: ")
try:
    soma = int(valor) + 10
    resultado = soma / int(valor)
except:
    print("Ocorreu um erro no sistema.")
```

Veja que foi feito um tratamento de exceção geral, sem especificar o tipo de exceção, assim, para qualquer exceção, o trecho de **except** vai ser executado.

EXEMPLO 2

```
valor = input("Informe um número: ")
try:
    soma = int(valor) + 10
    resultado = soma / int(valor)
except ZeroDivisionError:
    print("Ocorreu um erro de divisão por zero [ZeroDivisionError].")
except ValueError:
    print("Ocorreu um erro de valor [ValueError].")
```

Agora estão sendo tratados dois tipos de exceção, sendo assim, podemos tomar uma decisão diferente para cada exceção levantada.

TRY...ELSE

- O tratamento de exceções pode conter um bloco else, que é executado quando não ocorre nenhum erro.

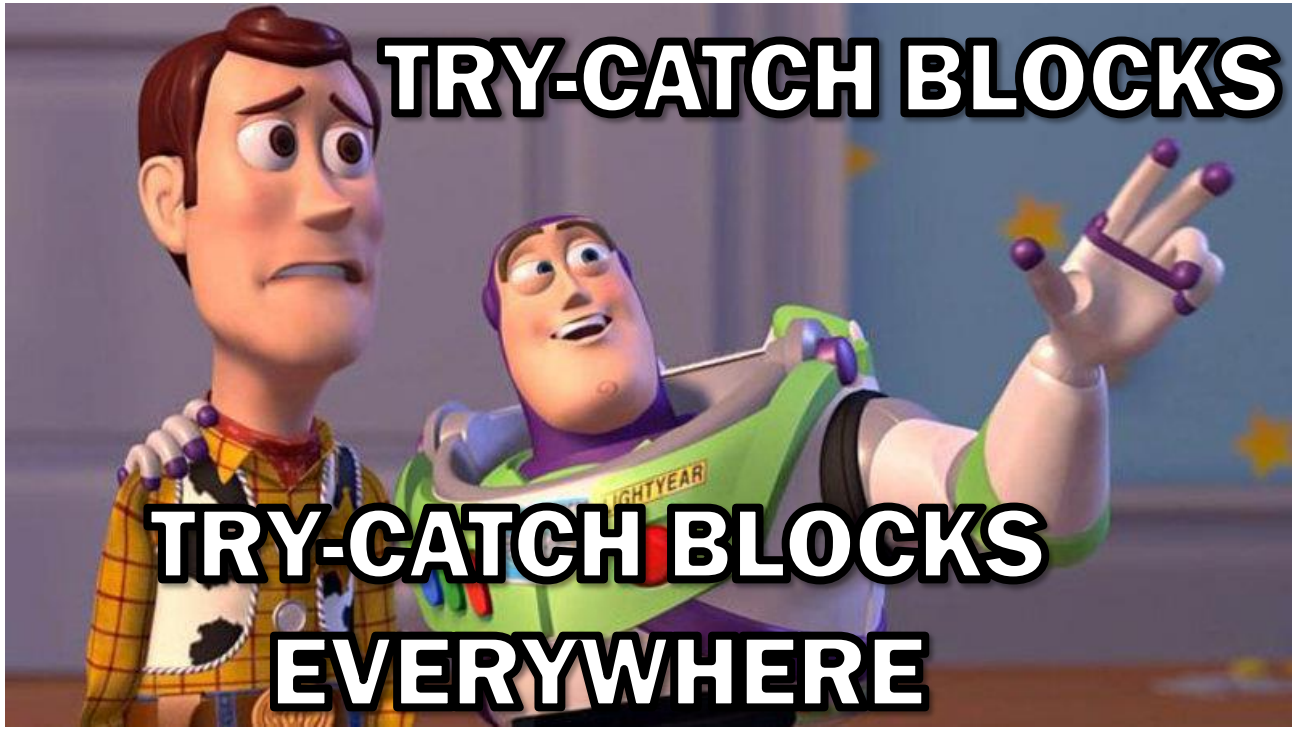
```
valor = input("Informe um número: ")
try:
    soma = int(valor) + 10
    resultado = soma / int(valor)
except ZeroDivisionError:
    print("Ocorreu um erro de divisão por zero [ZeroDivisionError].")
except ValueError:
    print("Ocorreu um erro de valor [ValueError].")
else:
    print("Não deu erro!!!")
```

TRY...FINALLY

```
valor = input("Informe um número: ")
try:
    soma = int(valor) + 10
    resultado = soma / int(valor)
except ZeroDivisionError:
    print("Ocorreu um erro de divisão por zero [ZeroDivisionError].")
except ValueError:
    print("Ocorreu um erro de valor [ValueError].")
else:
    print("Não deu erro!!!")
finally:
    print("Esta linha será executada ocorrendo erro ou não!!!")
```

O tratamento de exceções pode conter um bloco finally, que é executado sempre, ocorrendo erro ou não. É muito utilizado quando você precisa realizar alguma ação mesmo que dê erro no sistema. Pode ser utilizada para liberar recursos que foram usados no bloco try, tais como conexões com bancos de dados e arquivos abertos.

Quando tratamos a exceção o sistema não é interrompido, ele continua seu fluxo normalmente.



CAPTURANDO A MENSAGEM DA EXCEÇÃO

- A cláusula except pode especificar uma variável após o nome da exceção. A variável é ligada a uma instância de exceção com os argumentos armazenados em instance.args. A instância de exceção define __str __() para que os argumentos possam ser impressos diretamente, sem ter que fazer referência a .args.

À partir do **Python 3.6** a sintaxe correta é: except TIPO_EXCECAO as VARIÁVEL: except Exception as e:

```
valor = input("Informe um número: ")

try:
    soma = int(valor) + 10
    resultado = soma / int(valor)

    print("Soma:", soma)
    print("Resultado:", resultado)
except Exception as e:
    print("Ocorreu um erro. Veja o erro:", e)
    print("Ocorreu um erro. Veja o erro:", e.args)
```

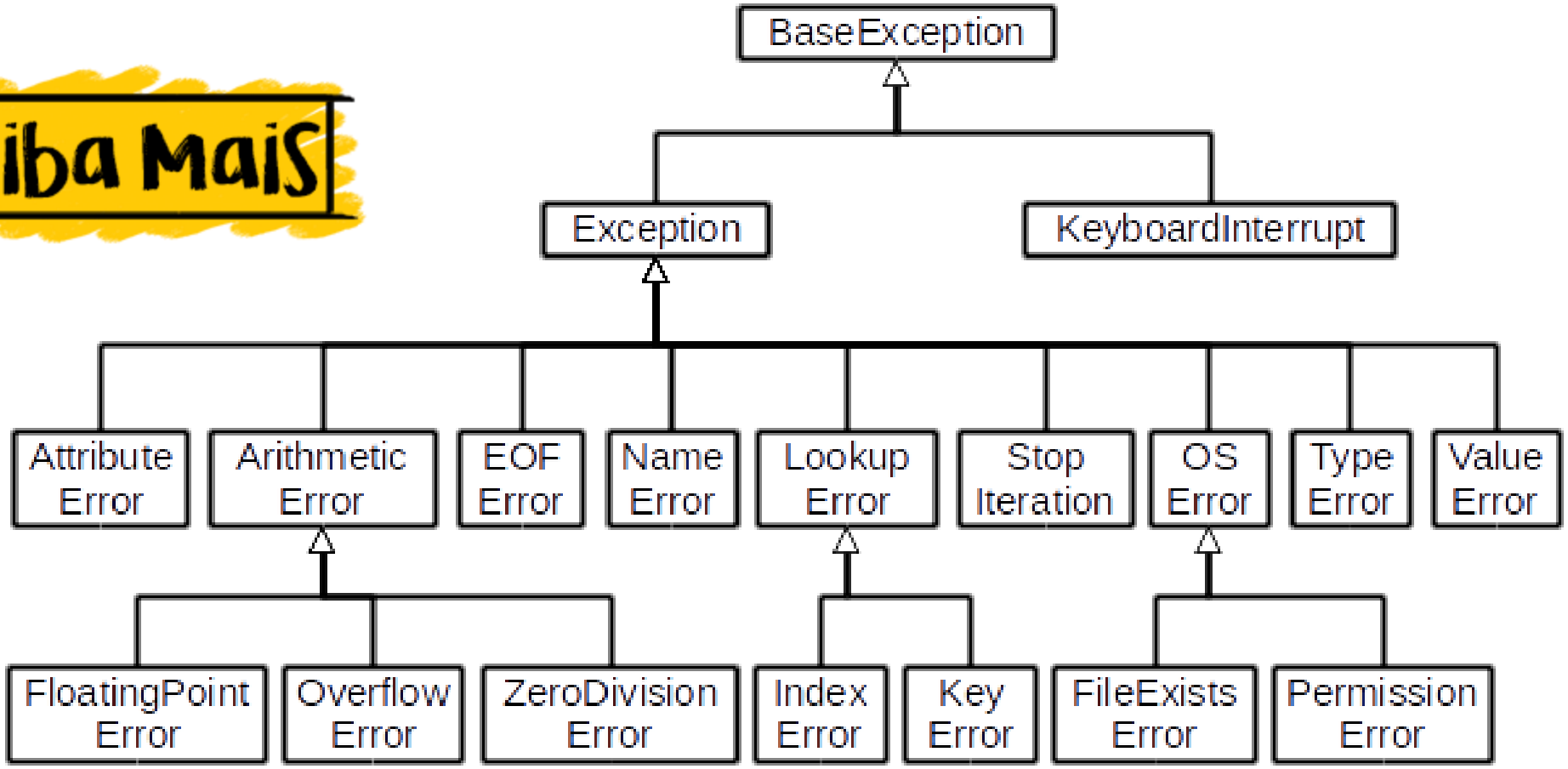
LEVANTANDO UMA EXCEÇÃO

Você pode criar/forçar uma exceção usando o raise.

```
def converte_inteiro(string):
    if string not in ("0","1","2","3","4","5","6","7","8","9"):
        raise ValueError("Informe um número de 0 a 9!!!")
    return int(string)
numero = input("Informe um número de 0 a 9: ")
print(10 + converte_inteiro(numero))
```

HIERARQUIA DE EXCEÇÕES

Saiba Mais



TRACEBACK

O traceback é a informação que o interpretador Python fornece para nos ajudar a identificar o problema em nosso código. Ele contém algumas informações úteis:

- A sequência de chamadas de funções até o ponto onde ocorreu o problema;
- O número da linha onde o erro foi gerado;
- O tipo de erro que ocorreu, bem como uma pequena mensagem informativa sobre o ocorrido.

Podemos utilizar o módulo **tracebak** para capturar a informação da exceção e utilizá-la por exemplo, para gravar os erros em um arquivo de log.

```
“.../excecao.py"
Informe um número: Olá
Traceback (most recent call last):
  File “.../excecao.py", line 3, in <module>
    soma = int(valor) + 10
ValueError: invalid literal for int() with base 10: 'Olá'
```

DURANTE A MANIPULAÇÃO DA EXCEÇÃO OCORREU OUTRA EXCEÇÃO

```
valor = input("Informe um número: ")

try:
    soma = int(valor) + 10
    resultado = soma / int(valor)

    print("Soma:", soma)
    print("Resultado:", resultado)
except:
    x = 10 / 0
    print("ERRROOOOO!!!!")
```

```
...\python.exe "…/erro_manipula_excecao.py"
Informe um número: Olá
Traceback (most recent call last):
  File "…/erro_manipula_excecao.py", line 4, in <module>
    soma = int(valor) + 10
ValueError: invalid literal for int() with base 10: 'Olá'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "…/erro_manipula_excecao.py", line 10, in <module>
    x = 10 / 0
ZeroDivisionError: division by zero

Process finished with exit code 1
```

EXCEÇÕES DEFINIDAS PELO USUÁRIO

```
class Error(Exception):
    """Classe base para outras exceções"""
    pass
class numeroMenorError(Error):
    """Exceção levantada quando o valor
informado é menor do que o esperado"""
    pass
class numeroMaiorError(Error):
    """Exceção levantada quando o valor
 informado é maior do que o esperado"""
    pass
numero_da_sorte = 20
```

```
while True:
    try:
        numero = int(input("Informe um número: "))
        if numero < numero_da_sorte:
            raise numeroMenorError
        elif numero > numero_da_sorte:
            raise numeroMaiorError
        break
    except numeroMenorError:
        print("Você informou um número menor, tente novamente!")
    except numeroMaiorError:
        print("Você informou um número maior, tente novamente!")
print("Acertou!!!")
```

EXERCÍCIO

Utilize o arquivo Exercicio1.py para fazer os tratamentos de erros necessários