

Algoritmos e Lógica de Programação

80 horas // 4 h/semana

Exceções em Python

Bloco: Try – Except

Aula 16

Prof. Piva

Para começar...

Este é um tópico extra, para melhorar a legibilidade e o processo de construção de scripts em Python

O que são Erros...

E quais são os mais comuns?

O que é um erro?

- Digite e escreva a seguinte linha de código...

```
Print("pyPRO – Seja um profissional Python")
```

O que é um erro?

- Digite e escreva a seguinte linha de código...

`Print("pyPRO – Seja um profissional Python")`

```
>>> Print("pyPRO - Seja um profissional Python")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
>>>
```

O que é um erro?

- Digite e escreva a seguinte linha de código...

```
print("pyPRO – Seja um profissional Python")
```

O que é um erro?

- Digite e escreva a seguinte linha de código...

```
print("pyPRO – Seja um profissional Python)
```

```
>>> Print("pyPRO – Seja um profissional Python)
      File "<stdin>", line 1
        Print("pyPRO – Seja um profissional Python)
                                     ^
SyntaxError: EOL while scanning string literal
>>>
```

Erros mais comuns...

- `SyntaxError`
- `NameError`
- `TypeError`
- `IndexError`
- `ValueError`
- `KeyError`
- `AttributeError`
- `IndentationError`

Erros mais comuns...

- **SyntaxError**
- NameError
- TypeError
- IndexError
- ValueError
- KeyError
- AttributeError
- IndentationError

```
>>> def = 3.5
      File "<stdin>", line 1
        def = 3.5
            ^
SyntaxError: invalid syntax
```

```
>>> print([])
      File "<stdin>", line 1
        print([])
              ^
SyntaxError: closing parenthesis ')' does not match opening parenthesis '['
```

Nesse erro... O interpretador Python não reconhece o que você escreveu como parte da linguagem.

Erros mais comuns...

- `SyntaxError`
- `NameError`
- `TypeError`
- `IndexError`
- `ValueError`
- `KeyError`
- `AttributeError`
- `IndentationError`

```
>>> Print("teste")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
```

```
>>> pyPro()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pyPro' is not defined
```

Nesse erro... Variável ou função não estão definidas.

Erros mais comuns...

- `SyntaxError`
- `NameError`
- `TypeError`
- `IndexError`
- `ValueError`
- `KeyError`
- `AttributeError`
- `IndentationError`

```
>>> len(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
```

Nesse erro... Ação ou operação é aplicada a um tipo indevido (para aquela ação/operação)

Erros mais comuns...

- `SyntaxError`
- `NameError`
- `TypeError`
- `IndexError`
- `ValueError`
- `KeyError`
- `AttributeError`
- `IndentationError`

```
>>> lista = [1,2,3,4]
>>> print(lista[4])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Nesse erro... Tentamos acessar um elementos fora do escopo (range) do tipo de dados.

Erros mais comuns...

- `SyntaxError`
- `NameError`
- `TypeError`
- `IndexError`
- `ValueError`
- `KeyError`
- `AttributeError`
- `IndentationError`

```
>>> int('a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'a'
```

Nesse erro... Uma função/operação integrada da linguagem recebe um argumento com tipo correto mas seu valor não está adequado para a função/operação desejada.

Erros mais comuns...

- `SyntaxError`
- `NameError`
- `TypeError`
- `IndexError`
- `ValueError`
- `KeyError`
- `AttributeError`
- `IndentationError`

```
>>> dic = {"nome": "Piva"}
>>> print(dic['sobrenome'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'sobrenome'
>>>
```

Nesse erro... Tentamos acessar um dicionário com uma chave que não existe.

Erros mais comuns...

- `SyntaxError`
- `NameError`
- `TypeError`
- `IndexError`
- `ValueError`
- `KeyError`
- `AttributeError`
- `IndentationError`

```
>>> tupla = (1,4,89,56)
>>> print(tupla.sort())
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
>>>
```

Nesse erro... Uma variável não tem um atributo / função indicado(a).

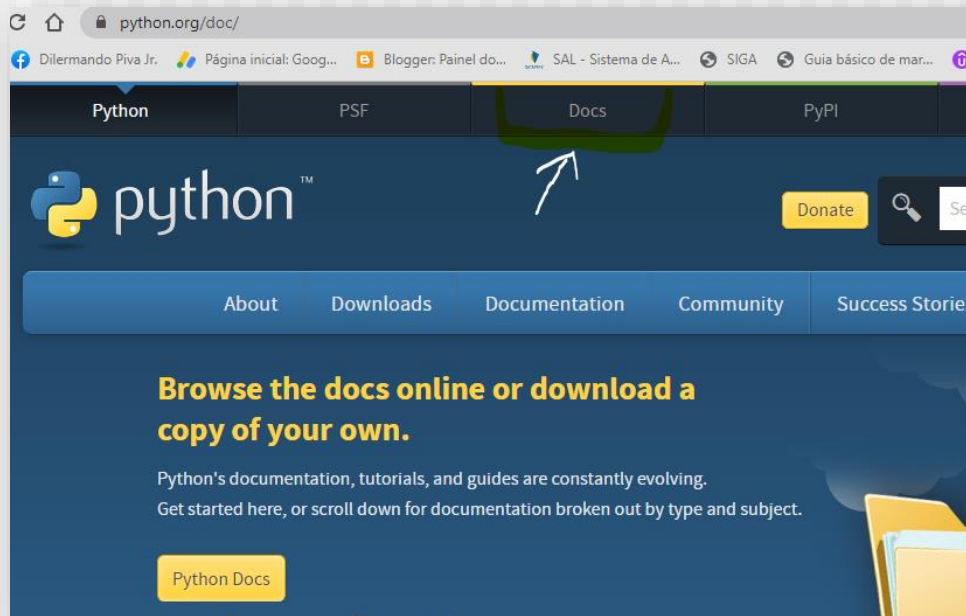
Erros mais comuns...

- `SyntaxError`
- `NameError`
- `TypeError`
- `IndexError`
- `ValueError`
- `KeyError`
- `AttributeError`
- `IndentationError`

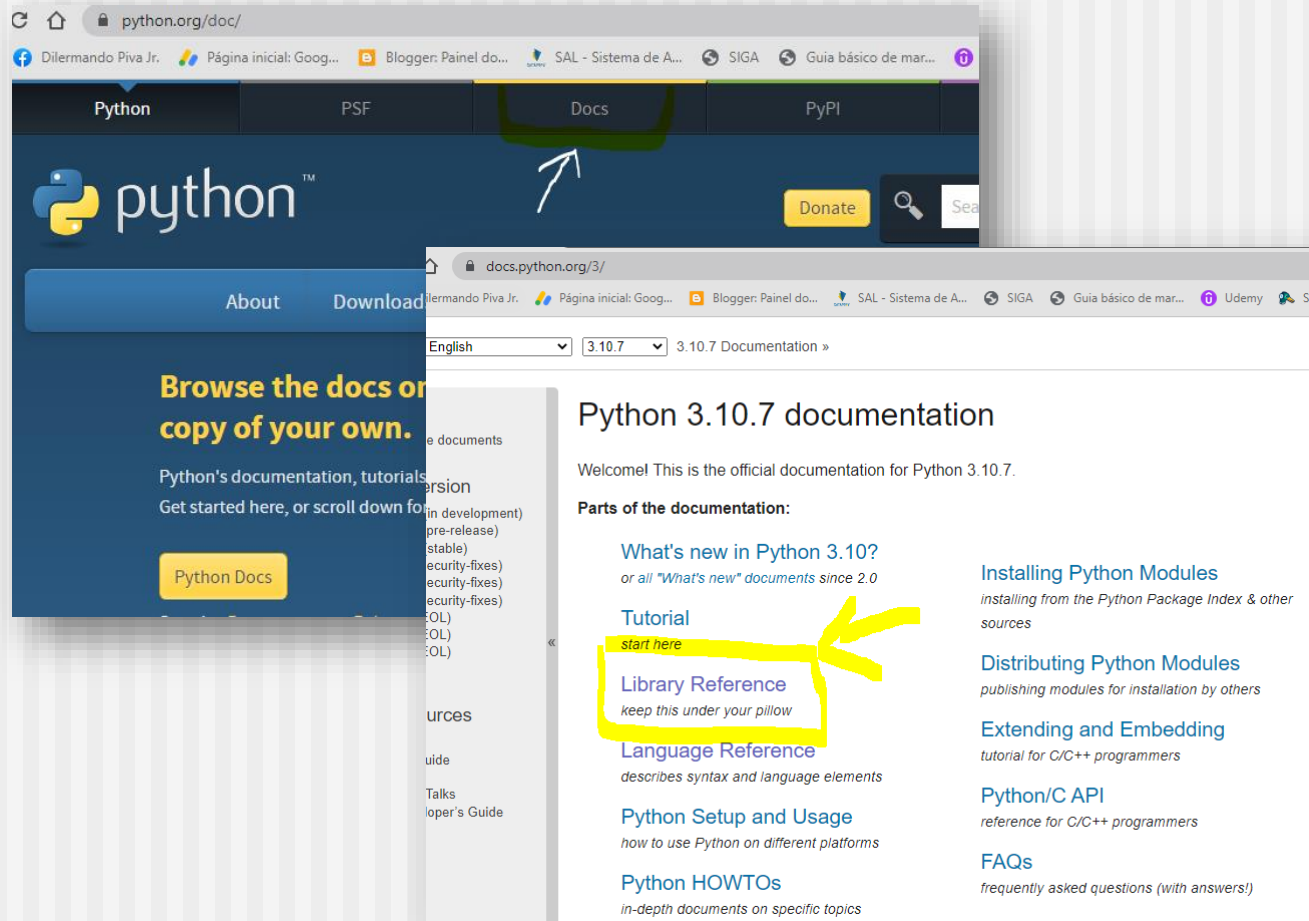
```
>>> def nova():  
... print()  
    File "<stdin>", line 2  
      print()  
      ^  
IndentationError: expected an indented block  
>>>
```

Nesse erro... Ocorre quando não respeitamos os 4 espaços de indentação (controle de bloco)

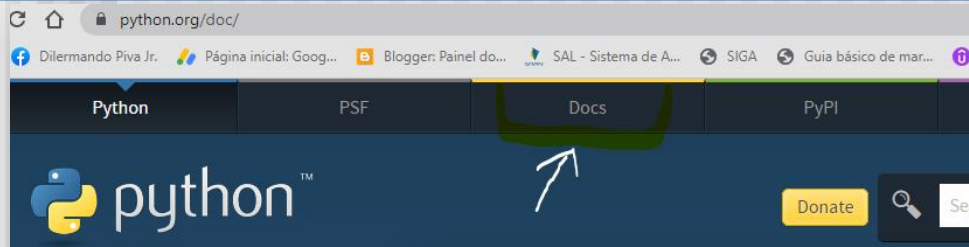
São muitos “tipos de Erros”



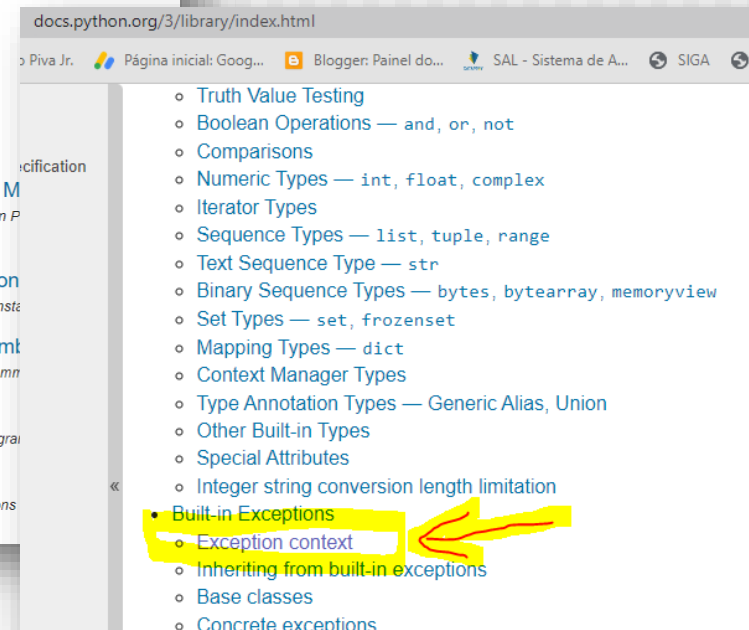
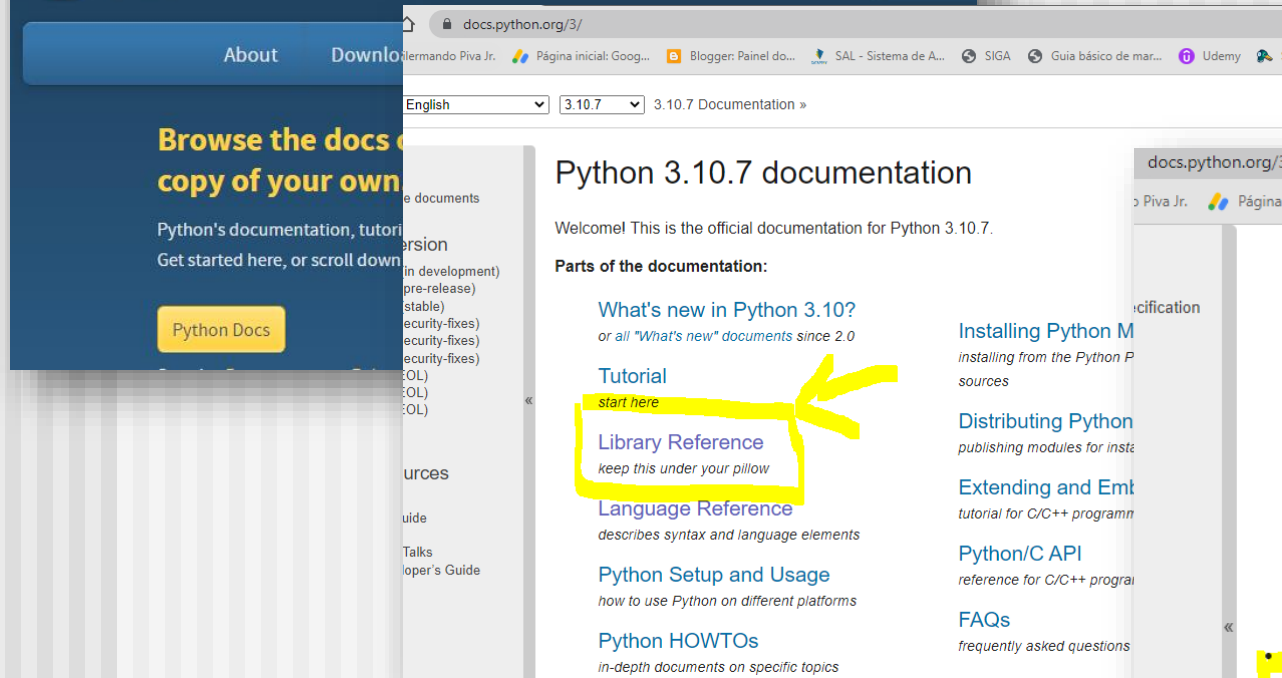
São muitos “tipos de Erros”



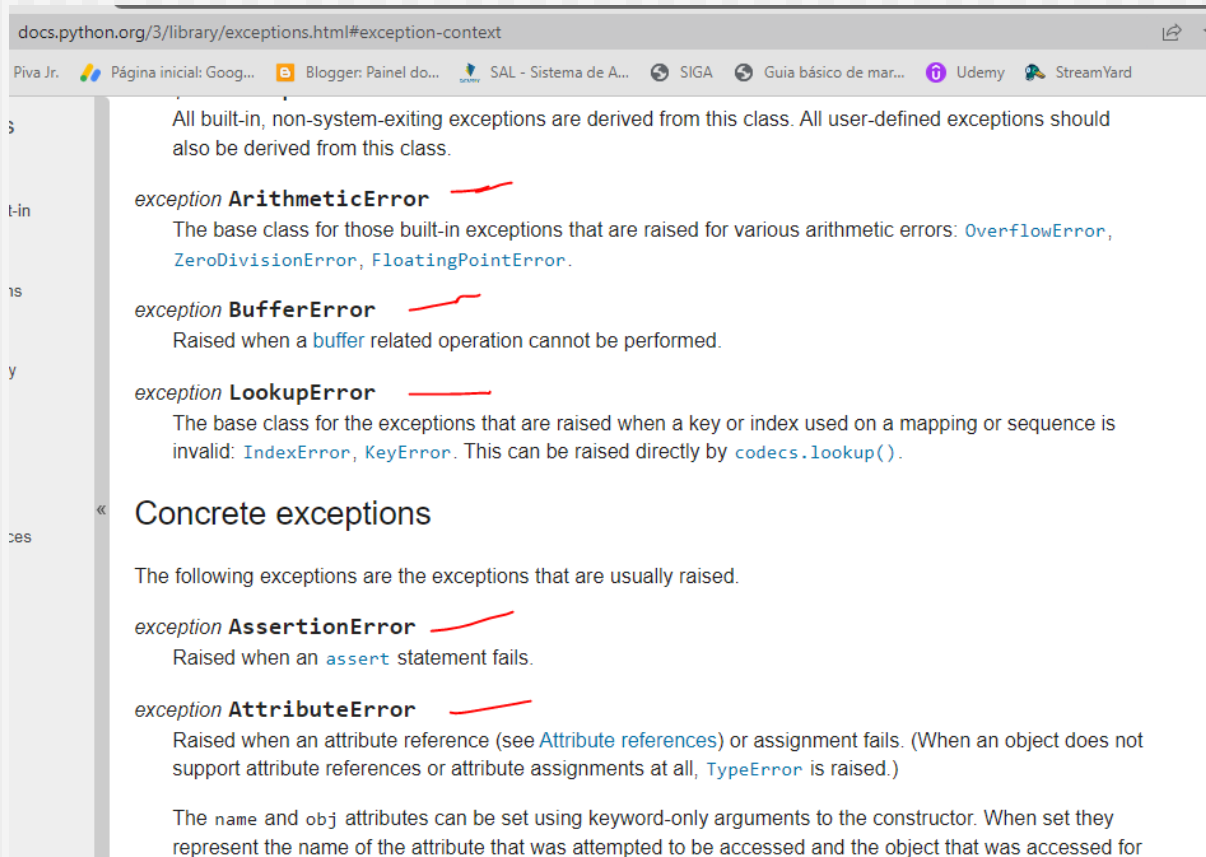
São muitos “tipos de Erros”



Exceptions e Errors – São sinônimos



São muitos “tipos de Erros”



docs.python.org/3/library/exceptions.html#exception-context

Piva Jr. | Página inicial: Goog... | Blogger: Painel do... | SAL - Sistema de A... | SIGA | Guia básico de mar... | Udemý | StreamYard

All built-in, non-system-exiting exceptions are derived from this class. All user-defined exceptions should also be derived from this class.

exception ArithmeticError

The base class for those built-in exceptions that are raised for various arithmetic errors: `OverflowError`, `ZeroDivisionError`, `FloatingPointError`.

exception BufferError

Raised when a `buffer` related operation cannot be performed.

exception LookupError

The base class for the exceptions that are raised when a key or index used on a mapping or sequence is invalid: `IndexError`, `KeyError`. This can be raised directly by `codecs.lookup()`.

Concrete exceptions

The following exceptions are the exceptions that are usually raised.

exception AssertionError

Raised when an `assert` statement fails.

exception AttributeError

Raised when an attribute reference (see [Attribute references](#)) or assignment fails. (When an object does not support attribute references or attribute assignments at all, `TypeError` is raised.)

The `name` and `obj` attributes can be set using keyword-only arguments to the constructor. When set they represent the name of the attribute that was attempted to be accessed and the object that was accessed for

O bloco Try/Except

Tratando os erros
de maneira mais
adequada!

O que é o bloco Try/Except

Em outras linguagens... Try / Catch
(ex. Java)

Permite que se antecipe ao lançamento de uma exceção (Erro) e então tal erro seja tratado de maneira mais adequada (sem, necessariamente, ter que parar a execução do programa)

O que é o bloco Try/Except

Sintaxe:

try:

 // verificação

 // tente fazer alguma coisa

except:

 // caso ocorra algum problema

 // faça isso...

O que é o bloco Try/Except

Exemplo 1 (genérico):

```
try:  
    pypro()  
except:  
    print("ocorreu algum problema !")
```


O que é o bloco Try/Except

Exemplo 2 (específico):

```
try:  
    pypro()  
except NameError:  
    print("Você está utilizando uma função inexistente!")
```

O que é o bloco Try/Except

Exemplo 3 (específico):

```
try:  
    len(9.4)  
except TypeError as erro:  
    print(f"A aplicação gerou o seguinte erro {erro}!")
```

O que é o bloco Try/Except

Exemplo 4 (vários tipos específico/genérico):

```
try:
    len(9.4)
except TypeError as erro1:
    print(f'Ocorreu TypeError {erro1}!')
except NameError as erro2:
    print(f'Ocorreu NameError {erro2}!')
except:
    print("Ocorreu um erro diferente!")
```

O que é o bloco Try/Except

Exemplo 5 (usando em uma função):

```
def retorna_valor(dicionario, chave):  
    try:  
        return dicionario[chave]  
    except KeyError:  
        return None  
    except TypeError:  
        return None
```

```
dicio = {"nome" = "Piva"}  
print(retorna_valor(dicio, 5))
```

O bloco Try/Except/Else/Finally

Tratando os erros
completamente...

O que é o bloco Try/Except/Else

A utilização apenas de Try/Except trata apenas a ocorrência de um erro.... E se não ocorrer?

Caso o erro não ocorra o bloco ELSE será executado.

Try/Except/Else

Exemplo:

try:

```
    num = int(input('informe um número: '))
```

except:

```
    print('Valor incorreto!')
```

else:

```
    print(f'Você digitou {num}')
```

Try/Except/Else/ FINALLY

Além de Try, Except e Else, existe a cláusula FINALLY.

FINALLY sempre será executado (existindo ou não a ocorrência de um erro)

Try/Except/Else/Finally

Exemplo:

```
try:
    n1 = int(input('informe um número: '))
except ValueError:
    print('Você não digitou um valor válido!')
else:
    print(f'Você digitou o número {n1}!')
finally:
    print("Fim da execução...")
```

Try/Except/Else/Finally

Exemplo – Juntando tudo!!

```
def divisao(a,b):  
    return a/b  
  
# ...  
try:  
    n1 = int(input('informe o dividendo: '))  
except:  
    print('Precisa ser um número!')  
else:  
    try:  
        n2 = int(input('informe o divisor: '))  
    except:  
        print('Precisa ser um número!')  
    else:  
        try:  
            print(divisao(n1/n2))  
        except:  
            print("Erro na divisão!")  
finally:  
    print("fim da execução!")
```

Try/Except/Else/Finally

Exemplo – Melhorando a Função divisão()

```
def divisao(a, b):  
    try:  
        return int(a) / int(b)  
    except (ValueError, ZeroDivisionError) as erro:  
        return f'Ocorreu um problema: {erro}'  
  
# ...  
n1 = int(input('informe o dividendo: '))  
n2 = int(input('informe o divisor: '))  
print(divisao(n1/n2))
```

VAMOS PARA A PRÁTICA ?!!!



Python:

Exceções (*Bloco: Try – Except*)

Exceções

- Quando um programa encontra dificuldades não previstas, diz-se que uma condição excepcional ou uma *exceção* ocorreu
 - Um erro é uma exceção mas nem toda exceção é um erro
- Para poder representar tais eventos, Python define os chamados objetos de exceção (*exception objects*)
- Se a condição excepcional não é prevista (e tratada), o programa termina com uma mensagem de rastreamento:

```
>>> 1/0
```

Traceback (most recent call last):

File "<pyshell#0>", line 1, in -toplevel-

1/0

ZeroDivisionError: integer division or modulo by zero

Objetos de Exceção

- Cada exceção individual corresponde a um *objeto de exceção*, que por sua vez é uma instância de alguma *classe de exceção*
 - No exemplo anterior, tal objeto é instância da classe `ZeroDivisionError`
- Diz-se que o programa gerou ou levantou (*raised*, em inglês) uma condição de exceção na forma de um objeto
- Um programa bem elaborado precisa capturar (*catch*, em inglês) tais objetos e tratá-los para que a execução não seja abortada

Avisos

- Existem condições excepcionais menos sérias que não provocam o levantamento de um objeto de exceção, mas apenas são exibidas sob a forma de um aviso
- Por exemplo,

```
>>> import regex
```

Warning (from warnings module):

File "__main__", line 1

DeprecationWarning: the regex module is deprecated; please use the re module

- Neste caso, o interpretador nos sinaliza que o módulo regex é antigo e que foi substituído por outro mais atualizado chamado re
- O programa não falha, mas o programador fica ciente que provavelmente deve reescrever seu programa usando o módulo re para evitar obsolescência

O comando *raise*

- Para sinalizar a ocorrência de uma condição excepcional, pode-se usar o comando *raise* que tem uma das formas:
 - *raise classe*
 - *raise classe, mensagem*
 - *raise classe (mensagem)*
- Onde classe é uma das classes de exceção definidas pelo Python
 - Para saber todos os tipos de exceção consulte o manual
 - Se quiser uma classe genérica use a classe *Exception*
 - Uma listagem pode ser obtida escrevendo

```
>>> import exceptions
>>> dir(exceptions)
['ArithmeticError', 'AssertionError', 'AttributeError', ...
```

Exemplo

```
>>> raise Exception
```

Traceback (most recent call last):

```
File "<pyshell#3>", line 1, in -toplevel-  
    raise Exception
```

Exception

```
>>> raise Exception,"Deu bode"
```

Traceback (most recent call last):

```
File "<pyshell#5>", line 1, in -toplevel-  
    raise Exception,"Deu bode"
```

Exception: Deu bode

```
>>> raise Exception("Deu Bode")
```

Traceback (most recent call last):

```
File "<pyshell#7>", line 1, in -toplevel-  
    raise Exception("Deu Bode")
```

Exception: Deu Bode

Algumas Classes de Exceção

Classe	Descrição
Exception	Classe base para todas as exceções
AttributeError	Falha no acesso ou atribuição a atributo de classe
IOError	Falha no acesso a arquivo inexistente ou outros de E/S
IndexError	Índice inexistente de seqüência
KeyError	Chave inexistente de dicionário
NameError	Variável inexistente
SyntaxError	Erro de sintaxe (código errado)
TypeError	Operador embutido aplicado a objeto de tipo errado
ValueError	Operador embutido aplicado a objeto de tipo certo mas valor inapropriado
ZeroDivisionError	Divisão ou módulo por zero

Criando uma Classe de Exceção

- Basta criar uma classe da forma habitual derivando-a da classe Exception
- Não é preciso redefinir qualquer método
- Ex.:

```
>>> class MinhaExcecao(Exception): pass
```

```
>>> raise MinhaExcecao("Deu bode!")
```

Traceback (most recent call last):

File "<pyshell#11>", line 1, in -toplevel-

raise MinhaExcecao("Deu bode!")

MinhaExcecao: Deu bode!

Capturando Exceções

- Para capturar uma exceção possivelmente levantada por um trecho de código, pode-se usar a construção try/except:

try:

Código

except *Exceções*:

Código de tratamento da exceção

- Sendo que *Exceções* pode ser:

- *Classe*
- *Classe as var*
- *(Classe1,...,ClasseN)*
- *(Classe1,...,ClasseN) as var*

- Onde:

- *Classe*, *Classe1* e *ClasseN* são nomes de classes de exceção
- *Var* é uma variável à qual é atribuída um objeto de exceção

Exemplo 1

```
>>> try:
```

```
    a = int(input("Entre com um número "))
```

```
    b = int(input("Entre com outro número "))
```

```
    print (a, "/", b, "=", a/b)
```

```
except ZeroDivisionError:
```

```
    print ("Ooops, segundo número não pode ser zero!")
```

Entre com um número 1

Entre com outro número 0

1 / 0 = Ooops, segundo número não pode ser zero!

Exemplo 2

```
>>> try:
```

```
    a = int(input("Entre com um numero "))
```

```
    b = int(input("Entre com outro numero "))
```

```
    print (a, "/", b, "=", a/b)
```

```
except (ZeroDivisionError,TypeError):
```

```
    print ("Ooops, tente novamente!")
```

Entre com um numero 1

Entre com outro numero "a"

1 / a = Ooops, tente novamente!

Exemplo 3

```
>>> try:
```

```
    a = int(input("Entre com um numero "))
```

```
    b = int(input("Entre com outro numero "))
```

```
    print (a, "/", b, "=", a/b)
```

```
except (ZeroDivisionError,TypeError) as e:
```

```
    print ("Ooops, deu erro:",e)
```

Entre com um numero 1

Entre com outro numero "z"

1 / z = Ooops, deu erro: unsupported operand type(s) for /: 'int' and 'str'

Mais *except*

- É possível tratar diferentemente as diversas exceções usando duas ou mais cláusulas `except`
- Se quisermos nos prevenir contra qualquer tipo de erro, podemos usar uma cláusula `except` sem nome de classe
 - Outra opção é usar a classe `Exception`, que é base para todas as exceções e portanto casa com qualquer exceção
- Se não quisermos tratar um erro em uma cláusula `except`, podemos passá-la adiante usando o comando `raise`
 - Nesse caso, podemos usar um `raise` sem argumentos ou passar explicitamente um objeto de exceção

Exemplo 4

```
>>> try:
```

```
    a = int(input("Entre com um numero "))
```

```
    b = int(input("Entre com outro numero "))
```

```
    print (a, "/", b, "=", a/b)
```

```
except ZeroDivisionError:
```

```
    print ("Ooops, divisão por zero")
```

```
except TypeError:
```

```
    print ("Ooops, você não deu um número")
```

```
except:
```

```
    print ("Deu um bode qualquer")
```

Entre com um numero 2

Entre com outro numero fads2312

Deu um bode qualquer

Exemplo 5

```
>>> try:
    a = int(input("Entre com um numero "))
    b = int(input("Entre com outro numero "))
    print (a, "/", b, "=", a/b)
except (ZeroDivisionError,TypeError) as e:
    print ("Ooops, deu erro:",e)
except Exception as e:
    print ("Deu bode não previsto:",e)
    raise
```

Entre com um numero a

Entre com outro numero

Deu bode não previsto: EOF when reading a line

Traceback (most recent call last):

File "<pyshell#52>", line 3, in -toplevel-

```
b = int(input("Entre com outro numero "))
```

EOFError: EOF when reading a line

A cláusula *else*

- É possível completar um comando try com uma cláusula else que introduz um trecho de código que só é executado quando *nenhuma* exceção ocorre:

try:

Código

except *Exceções*:

Código de tratamento da exceção

else:

Código executado se não ocorrem exceções

Exemplo 6

```
>>> while True:
```

```
    try:
```

```
        a = int(input("Entre com um numero "))
```

```
        b = int(input("Entre com outro numero "))
```

```
        print (a, "/", b, "=", a/b)
```

```
    except Exception as e:
```

```
        print ("Deu bode:",e)
```

```
        print ("Tente novamente")
```

```
    else:
```

```
        break
```

Entre com um numero 1

Entre com outro numero xxx

Deu bode: name 'xxx' is not defined

Tente novamente

Entre com um numero 1

Entre com outro numero 2

1 / 2 = 0

A cláusula *finally*

- A cláusula `finally` pode ser usada para se assegurar que mesmo que ocorra algum erro, uma determinada sequência de comandos vai ser executada
 - Pode ser usada para restabelecer alguma variável para um valor default, por exemplo
- A cláusula `finally` e cláusulas `except` são mutuamente exclusivas
 - Exceções nesse caso não são tratadas
 - É possível combinar ambas usando comandos `try` aninhados, mas normalmente não há muito uso para isso

Exemplo 7

```
>>> try:
    try:
        x = int(input("Entre com um número"))
    finally:
        print ("restabelecendo um valor para x")
        x = None
except:
    print ("Deu bode")
```

Entre com um número 2xx
restabelecendo um valor para x
Deu bode

EXERCÍCIOS

Try-Except...

EXERCÍCIO 1

Cadastre os dados pessoais de um número ilimitado de pessoas. Os dados são os seguintes:

- **Sobrenome**
- **Idade**
- **Altura (em centímetros – número inteiro)**
- **Peso (em kg.)**

Para finalizar a entrada dos dados, basta digitar <enter> sem nenhuma informação no campo sobrenome.

Ao final, mostrar uma listagem em ordem alfabética das pessoas, suas idades, altura e peso...

Como resumo dessa tabela, mostrar a média da idade, altura e peso.

Fazer a validação dos dados não permitindo que sejam inseridos dados inválidos nos respectivos campos.