

# ASIMOV

---

## Lidando com erros e exceções

Nesta aula aprenderemos sobre Erros e Manipulação de Exceções em Python. Você definitivamente já encontrou erros nesse ponto no curso. Por exemplo:

In [1]:

```
print('Hello)
```

```
File "<ipython-input-1-db8c9988558c>", line 1
    print('Hello)
            ^
```

**SyntaxError:** EOL while scanning string literal

Observe como obtemos um `SyntaxError`, com a descrição adicional de que era um EOL (End of Line Error). Isso é suficiente para que percebamos que esquecemos um único apóstrofe no final da linha. Compreender estes vários tipos de erro irá ajudá-lo a depurar seu código muito mais rápido.

Este tipo de erro e descrição é conhecido como uma Exceção. Mesmo que uma declaração ou expressão seja sintaticamente correta, pode causar um erro quando tentamos executá-la. Os erros detectados durante a execução são chamados de exceções e não são incondicionalmente fatais.

Você pode verificar a lista completa de exceções embutidas [aqui](#). Vamos aprender a lidar com erros e exceções em nosso próprio código.

## try e except

A terminologia básica e a sintaxe usadas para lidar com erros no Python são as instruções **try** e **except**. O código que pode causar uma exceção ocorre é colocado no bloco *try* e o tratamento da exceção é implementado no *do bloco de código except*\*. O formulário de sintaxe é:

```
try:
    Você tenta fazer algo aqui...
    ...
except ExceptionI:
    Se causar a ExceptionI, roda isso.
except ExceptionII:
    Se causar a ExceptionII, roda isso.
    ...
else:
    Se não causar excessões, roda isso.
```

Nós também podemos apenas verificar qualquer exceção apenas com `except`: Para obter uma melhor compreensão de tudo isso, vamos verificar um exemplo: Vamos ver algum código que abre e grava um arquivo:

```
In [ ]:
try:
    f = open('testfile','w')
    f.write('Test write this')
except IOError:
    # Isso só irá verificar se há uma exceção IOError e, em seguida, executar
    print("Error: Could not find file or read data")
else:
    print("Content written successfully")
    f.close()
```

Agora, vamos ver o que aconteceria se não tivéssemos permissão de escrita (abrindo apenas com 'r'):

```
In [ ]:
try:
    f = open('testfile','r')
    f.write('Test write this')
except IOError:
    # This will only check for an IOError exception and then execute this pri
    print("Error: Could not find file or read data")
else:
    print("Content written successfully")
    f.close()
```

Ótimo! Observe como imprimimos apenas uma declaração! O código ainda correu e conseguimos continuar fazendo ações e executando blocos de código. Isso é extremamente útil quando você deve ter em conta possíveis erros de entrada em seu código. Você pode estar preparado para o erro e continuar executando o código, em vez de seu código apenas quebrar como vimos acima.

Também poderíamos ter imprimido caso não tivéssemos certeza de qual seria a exceção. Por exemplo:

```
In [ ]:
try:
    f = open('testfile','r')
    f.write('Test write this')
except:
    print("Error: Could not find file or read data")
else:
    print("Content written successfully")
    f.close()
```

Ótimo! Agora, na verdade, não precisamos memorizar essa lista de tipos de exceção! Agora, e se continuássemos querendo executar código após a ocorrência da exceção? É aí que o **finally** entra.

## finally

O `finally`: o bloco de código sempre será executado, independentemente de existir uma exceção no bloco de código `try`. A sintaxe é:

```
try:
    Seu código aqui
...
Devido a qualquer exceção, este código pode ser ignorado!
finally:
    Este bloco de código sempre seria executado.
```

Por exemplo:

```
In [ ]:
try:
    f = open("testfile", "w")
    f.write("Test write statement")
finally:
    print("Always execute finally code blocks")
```

Nós podemos usar finally em conjunto com except. Vamos ver um novo exemplo que levará em consideração um usuário que coloque a entrada errada:

```
In [ ]:
def askint():
    try:
        val = int(raw_input("Entre um inteiro: "))
    except:
        print("Parece que você não digitou um inteiro")

    finally:
        print("Executei!")
    print(val)
```

```
In [ ]:
askint()
```

Observe como obtivemos um erro ao tentar imprimir val (porque nunca foi atribuído corretamente). Repare isso perguntando ao usuário e verificando se o tipo de entrada é um número inteiro:

```
In [ ]:
def askint():
    try:
        val = int(raw_input("Please enter an integer: "))
    except:
        print("Looks like you did not enter an integer!")
        val = int(raw_input("Try again-Please enter an integer: "))
    finally:
        print("Finally, I executed!")
    print(val)
```

```
In [ ]:
askint()
```

Como podemos continuar continuando a verificar? Podemos usar um loop while!

```
In [ ]:
def askint():
    while True:
        try:
            val = int(raw_input("Please enter an integer: "))
        except:
            print("Looks like you did not enter an integer!")
```

```
        continue
    else:
        print('Yep thats an integer!')
        break
    finally:
        print("Finally, I executed!")
print(val)
```

In [ ]: askint(1)