



escola  
britânica de  
artes criativas  
& tecnologia

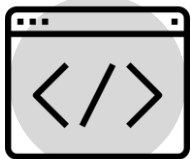
# Python para Análise de Dados



# DE OLHO NO CÓDIGO



# Tratamento de erros



Confira boas práticas da linguagem Python por assunto relacionado às aulas.

- **Resolva erros de sintaxe**





- **Trate erros em tempo de execução**

- **Referência Bibliográfica**



# Resolva erros de sintaxe

De olho no código

- 
 Verifique o clássico `=` ao invés de `==` dentro de uma condicional.
- 
 Assegure-se de que cada string no código tenha as aspas correspondentes.
- 
 Verifique se a endentação está consistente. Você pode endentar com espaços ou com tabulações, mas é melhor não misturá-los. Cada nível deve ser aninhado com a mesma quantidade.
- 
 Verifique a existência do sinal de dois pontos no final do cabeçalho de cada declaração composta, incluindo as declarações `for`, `while`, `if`, e `def`.



# Resolva erros de sintaxe

De olho no código

- Se você tem *strings* de multilinhas criadas usando três aspas (simples ou duplas), assegure-se de que você terminou a *string* apropriadamente. Uma string terminada de forma inapropriada ou não terminada pode gerar um erro de `invalid token` (objeto inválido) no final do seu programa, ou ele pode tratar a parte seguinte do programa como uma *string* até chegar à próxima *string*. No segundo caso, pode ser que o interpretador nem mesmo produza uma mensagem de erro!
- Um conjunto de parênteses, colchetes ou chaves não fechados corretamente faz com que o Python continue com a próxima linha como parte da declaração anterior. Geralmente, um erro ocorre quase imediatamente na linha seguinte.



# Resolva erros de sintaxe

De olho no código

- Certifique-se que você não está utilizando uma palavra reservada de Python para um nome de variável.. Python tem 33 palavras reservadas:

<code>and</code>	<code>def</code>	<code>for</code>	<code>is</code>	<code>raise</code>	<code>False</code>
<code>as</code>	<code>del</code>	<code>from</code>	<code>lambda</code>	<code>return</code>	<code>None</code>
<code>assert</code>	<code>elif</code>	<code>global</code>	<code>nonlocal</code>	<code>try</code>	<code>True</code>
<code>break</code>	<code>else</code>	<code>if</code>	<code>not</code>	<code>while</code>	
<code>class</code>	<code>except</code>	<code>import</code>	<code>or</code>	<code>with</code>	
<code>continue</code>	<code>finally</code>	<code>in</code>	<code>pass</code>	<code>yield</code>	

Para aprofundar seus conhecimentos em variáveis, acesse o **Apêndice A: Depuração**, do livro [Aprenda computação com Python](#).



# Trate erros em tempo de execução

De olho no código



Esses são alguns dos erros de tempo de execução pelo uso incorreto de dados mais comuns:

- NameError (Erro de Nome):**  
 A variável utilizada que não existe no ambiente atual. Lembre-se que variáveis locais são locais. Você não pode referenciá-la fora da função onde ela foi definida.
- IndexError (Erro de Índice):**  
 O índice usado para acessar uma lista, *string* ou tupla não existe no objeto, ou seja, é maior que seu comprimento menos um. Imediatamente antes do ponto do erro, adicione `print` para mostrar o valor do índice e o comprimento do objeto.
- KeyError (Erro de Chave):**  
 Você está tentando acessar um elemento de um dicionário utilizando um valor de chave que o dicionário não contém.
- AttributeError (Erro de Atributo):** Você está tentando acessar um atributo ou método que não existe em um objeto.

# Trate erros em tempo de execução

De olho no código

## ● **TypeError (Erro de Tipo):** Existem várias causas possíveis:

- Quando um valor é utilizado de forma imprópria..
- O número errado de argumentos é passado para uma função ou método. Para métodos, verifique sua definição e confira se o primeiro parâmetro chama-se `self`. Então verifique a chamada ao método; certifique-se de que está chamando o método em um objeto com o tipo correto e fornecendo os argumentos adequados.
- Há uma incompatibilidade entre os itens em um formato de *string* e os itens passados para conversão. Isto pode acontecer se o número de itens não for igual ou se uma conversão inválida é chamada. Por exemplo: Passar uma *string* para a formatação de conversão `%f`.

Para saber mais tipos de exceções. Acesse a seção **Exceções concretas**, no site [Python](#).






# Trate erros em tempo de execução

De olho no código



- 
 É possível escrever programas que tratam exceções específicas. A instrução `try` funciona da seguinte maneira:
  - Primeiramente, a cláusula `try` é executada.
  - Se nenhuma exceção ocorrer, a cláusula `except` é ignorada e a execução da instrução `try` é finalizada.
  - Se ocorrer uma exceção durante a execução de uma cláusula `try`, as instruções remanescentes na cláusula são ignoradas. Se o tipo da exceção ocorrida tiver sido previsto em algum `except`, essa cláusula `except` é executada, e então depois a execução continua após o bloco `try/except`.
  - Se a exceção levantada não corresponder a nenhuma exceção listada na cláusula de exceção, então ela é entregue a uma instrução `try` mais externa. Se não existir nenhum tratador previsto para tal exceção, trata-se de uma exceção não tratada e a execução do programa termina com uma mensagem de erro.

# Trate erros em tempo de execução

De olho no código



- A instrução `raise` permite ao programador forçar a ocorrência de um determinado tipo de exceção.
- Se um programa para e parece não estar fazendo nada, dizemos que ele “travou”. Geralmente isto significa que ele foi pego num laço infinito ou numa recursão infinita.

- Para saber mais como levantar exceções, leia a seção **8.4 Levantando exceções**, do site [Python](#).
- Para saber mais sobre isso, leia a seção **A.2.2 Meu programa trava**, do site [Aprendendo computação com Python](#).
- Para saber mais sobre testes unitários, leia a seção **Framework de Testes Unitários**, do site [Python](#).



# Trate erros em tempo de execução

De olho no código

- 
 A instrução `try` possui outra cláusula opcional, cuja finalidade é permitir a implementação de ações de limpeza, que sempre devem ser executadas independentemente da ocorrência de exceções.
- 
 Se uma cláusula `finally` estiver presente, a cláusula `finally` será executada como a última tarefa antes da conclusão da instrução `try`. A cláusula `finally` executa se a instrução `try` produz uma exceção.

Para saber os pontos mais complexos de quando ocorre uma exceção, leia a seção **8.7 Definindo ações de limpeza**, do site [Phyton](#).



# Referências Bibliográficas

- <https://www.python.org/>
- <https://aprendendo-computacao-com-python.readthedocs.io/en/latest/index.html>
- <https://mange.ifrn.edu.br/python/aprenda-com-py3/index.html>



# Bons estudos!

