

Python Reserved Words List - Your Complete Guide

Lista de Palavras Reservadas Python - Seu Guia Completo

Rajat Gupta

Software Developer

Published on Mon Jun 20 2022

Tradução: Alexandre Cardoso Garcia Leite
Python Developer

Publicado em 20 de Julho de 2024

Repositório: github.com/aleitebr

There is a restriction while naming identifiers that there are some restricted words that are built-in to Python which cannot be used as an identifier. Python reserved words (also called keywords) with a predefined meaning and syntax in the language which Python uses for its syntax and internal processing. In this tutorial, we will discuss what those keywords are.

Há uma restrição quando você nomeia “identificadores” que são palavras restritas da linguagem **Python** que não podem ser usados como um “identificador”. Palavras do **Python reservadas** (também chamadas de palavras chaves) têm significados pré-definidos e são usadas pelo **Python** para processamento interno e dentro de uma específica sintaxe. Neste tutorial, nós discutiremos quais são essas palavras chaves.

Reserved words in Python

Palavras Reservadas em Python

Here is the list of all the reserved words in Python.

Note - This list may change with different versions of Python. Python 3 has 33 while

Python 2 has 30 reserved words. The `print` was removed from Python 2 keywords and added as a built-in Python function.

Aqui é a lista de todas as palavras reservadas em **Python**. **Nota:** Esta lista pode mudar com diferentes versões do **Python**. **Python 3** tem 33 palavras enquanto o **Python 2** tem 30 palavras reservadas. A “função” `print` foi removida das palavras chaves do **Python 2** e adicionada como uma “função” **Python** “pré-definida”.

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

All the keywords except `True`, `False` and `None` are in lowercase and they must be written as they are. These cannot be used as variable names, function names, or any other identifiers.

Todas as “palavras chaves” com exceção de **True**, **False** e **None** estão em letras minúsculas e elas devem ser escritas desta forma. Elas não podem ser usadas como nome de “variáveis”, “funções”, ou qualquer outro “identificador”.

If any of the keywords is used as a variable, then we will get the error message `SyntaxError: invalid syntax`

Se qualquer dessas “palavras chaves” forem usadas como uma “variável”, então nós receberemos a mensagem de erro `SyntaxError: invalid syntax`

Keywords

False

It is a boolean operator that represents the opposite of `True`.

É um operador booleano que representa o oposto de ***True***.

Input:

Entrada:

```
>> print( 5 == 10 )
```

Output:

Saída:

False

Since the values are not actually equal, it returns `False`.

Desde que os valores não sejam iguais, o *console* retorna ***False***.

def

The `def` function is used to define a function or a method in Python.

O ***def*** é usado para definir uma função em Python.

Input:

```
>>> def welcome(name):
```

```
...     print (f"{name}, Welcome to Flexiple")
```

Output:

Bruno, Welcome to Flexiple

A function 'welcome' is defined using the def statement

A função 'welcome' é definida usando a declaração **def**.

if

An if statement is used to make a conditional statement. If the condition is True, then some action is performed.

Uma declaração **if** é usada como uma declaração condicional. Se a condição é **True** (verdadeira), então alguma ação é executada.

Input:

```
>>> age = 17

>>> if age >= 16:

...     print("Você é elegível para votar.")
```

Output:

Você é elegível para votar.

Since the age is greater than 16, the condition is True and the print command is executed.

Desde que a variável *age* é maior ou igual a 16, a condição é **True** (verdadeira) e o comando print é executado.

raise

The raise statement is used to raise an error. These errors are visible in the traceback and they cancel the execution of the program if not handled properly.

O comando **raise** é usado para forçar um erro. Esses erros são visíveis pelo rastreador de erros e eles cancelam a execução do programa se não forem tratados adequadamente.

Input:

```
>>> enter = "nick"

>>> if not type(enter) is int:

...     raise TypeError("Somente inteiros são permitidos.")
```

Output:

```
TypeError: Somente inteiros são permitidos.
```

TypeError is raised if the variable does not contain integers.

TypeError é lançado se a variável não conter números inteiros.

None

There is no null value in Python. The None is an object that represents the absence of a value. It is like an empty object.

Não há valor nulo em **Python**. O **None** é um objeto que representa a ausência de um valor. É como um objeto vazio.

Input:

```
>>> age = None

>>> if age is None:

...     print("Resultado Inválido")
```

Output:

Resultado Inválido

The age variable has no value to it. This satisfies the condition in the if statement.

A variável **age** não tem nenhum valor adicionado a ela. Isto satisfaz a condição do comando **if**.

del

The del statement is used to delete an object in Python.

O comando **del** é usado para remover um objeto em Python.

Input:

```
>>> names = [ 'Bruno', 'Maurício', 'Clayton' ]

>>> del names[1]

>>> print( names )
```

Output:

['Bruno', 'Clayton']

'Maurício' is removed from the list.

'Maurício' é removido da lista.

import

This statement is used to import modules to the project.

Este comando é usado para importar módulos para um projeto.

Input:

```
>>> import math
>>> print (math.pi)
```

Output:

3.141592653589793

This statement will import the math library into the project.

Este comando irá importar a biblioteca **math** para o projeto.

return

This keyword is used to exit a function or a method and return some value.

Esta palavra-chave é usada para sair de uma função ou método e retornar algum valor.

Input:

```
>>> def soma(a, b):
...     return a + b
>>> print( soma(2, 5) )
```

Output:

7

The function returns the sum of the two variables.

A função retorna a soma de duas variáveis.

True

It is a boolean operation that represents if the value is True.

Este é uma operação booleana que representa se o valor é **True** (verdadeiro).

Input:

```
>>> print(10 == 10)
```

Output:

True

The values are same so it returns True.

Os valores são os mesmos, então é retornado o valor **True** (verdadeiro).

elif

Shorthand for else if, checks if some other condition holds when the condition in the if statement is false.

Abreviação para **else if**, checa se alguma outra condição é verdadeira quando a condição no comando **if** é **False**.

Input:

```
>>> age = 16
>>> if age > 18:
...     print("Você deve registrar-se no TRE para votar.")
... elif age >= 16:
...     print("Você é elegível para votar, mas deve se registrar.")
... elif age < 16:
...     print("Você ainda não e elegível para votar.")
```

Output:

Você é elegível para votar, mas deve se registrar.

The condition in the if statement was not True. Hence, the elif statement looked for other conditions that are True

A condição no comando **if** não é **True** (verdadeira). Portanto, o comando **elif** é procurado para testar outras condições que sejam **True**.

in

The in statement is used to check if an element is present in an iterable like list or tuple.

O declarador **in** é usado para checar se um elemento está presente em um objeto indexado..

Input:

```
>>> nomes = ['Bruno', 'Maurício', 'Clayton']  
  
>>> existe = 'Clayton' in nomes  
  
>>> print(existe)
```

Output:

True

'Clayton' is present in the names list.

'Clayton' está presente na lista *nomes*.

try

The try statement is used to make a try... except statement. The try statement starts a block of code that is tried to execute. If it fails, the except block catches the error.

A declaração **try** é usada para fazer um comando **try ... except**. O comando **try** inicia um bloco de código que tenta ser executado. Se ele falha, o bloco **except** trata o erro.

Input:

```
>>> a = 5
>>> b = 0
>>> try:
...     div = a / b
... except ZeroDivisionError:
...     print("Divisor não pode ser zero.")
```

Output:

Divisor não pode ser zero.

The divisor is 0, which is not possible. So the except block catches the error.

O divisor é 0, que não é possível. Então o bloco **except** trata o erro.

and

It is one of the logical operators that returns True if both of the statements are True. The Truth table for the and operator is as follows:

and é um operador lógico que retorna **True** se ambos os argumentos são **True**.

A tabela verdade para o operador **and** é a que segue:

A	B	A and B
---	---	---------

True	True	True
True	False	False
False	True	False
False	False	False

Input:

```
>>> a = 5
```

```
>>> b = 10
```

```
>>> c = 15
```

```
>>> resultado = b > a and c > b
```

```
>>> print(resultado)
```

Output:

True

Both the statements are True and that is why the and operator returns True.
Ambas as operações retornam **True** e, portanto, o operador **and** retorna **True**.

else

Conditional statement that tells to perform alternate action if the condition in the if statement is False.

Comando condicional que diz ao interpretador para fazer uma ação alternativa caso a condição do comando **if** for **False**.

Input:

```
>>> age = 15
>>> if age >= 16:
...     print("Você é elegível para votar.")
... else:
...     print("Você não é elegível para votar.")
```

Output:

Você não é elegível para votar.

The condition in the if statement was not True, so the alternate action is executed.

A condição no comando if não resulta **True**, então a ação alternativa é executada.

is

This statement is used to check if the two variables are equal.

Este comando é usado para checar se duas variáveis são iguais.

Input:

```
>>> a = [2, 3]
>>> b = [2, 3]
>>> c = a
>>> print(b is a)
>>> print(c is a)
```

Output:

False
True

If result is *True*, both the variables point to the same memory place. Otherwise, the variables point to the different places.

Se o resultado é **True**, ambas as variáveis apontam para o mesmo lugar na memória. Caso contrário, as variáveis apontam para diferentes lugares.

while

This statement is used to start a while loop. It continues iteration until a condition is no longer True.

Este comando é usado para iniciar um “loop” (executar os mesmos comandos várias vezes). Ele continua até que a condição não seja mais **True** (verdadeiro).

Input:

```
>>> vogais = ['a', 'e', 'i', 'o', 'u']
>>> i = 0
>>> while i < 3:
...     print(vogais[i])
...     i = i + 1
```

Output:

a
e
i

The loop will run till the value of *i* is smaller than three.

Os comandos dentro do bloco **while** são repetidos até que *i* seja igual a três.

as

The `as` statement in Python re-assigns a returned object to a new identifier. Basically, it creates an alias.

O comando **as** em Python ressignifica um objeto para um novo identificador. Basicamente, ele cria um “apelido” para o identificador.

Input:

```
>>> import datetime as dt
>>> hoje = dt.date.today()
>>> print(hoje)
```

Output:

2024-07-19

The `datetime` is identified as `dt` in the code.

`datetime` é identificado como (“**as**”) `dt` no código.

except

Part of the `try... except` errorhandling structure in Python. Tells what to do when an exception occurs.

Parte do **try .. except** é uma estrutura que trata erros no Python. Diz o que fazer quando um erro (“**exception**”) ocorre.

Input:

```
>>> a = 5
>>> b = 0
>>> try:
...     div = a / b
... except ZeroDivisionError:
...     print("Divisor não pode ser zero.")
```

Output:

Divisor não pode ser zero.

The divisor is 0, which is not possible. So the except block catches the error.

O divisor é 0, que não é possível. Então o bloco **except** trata o erro.

lambda

A lambda function in Python is an anonymous function. It can take any number of arguments but only have a single expression.

A função **lambda** em Python é uma função anônima. Ela pode pegar qualquer número de argumentos, mas somente possui uma expressão simples.

Input:

```
>>> numeros = [1, 2, 3, 4]
>>> cubo_numeros = map(lambda(x: x ** 3 : numeros)
>>> print(list(cubo_numeros))
```

Output:

[1, 8, 27, 64]

The cube of the variable is an anonymous function.

O cubo da variável é uma função anônima.

with

The with statement is used to simplify the exception handling.

O comando **with** é usado para simplificar o tratamento de erros. No exemplo abaixo, ele abre o arquivo, lê e fecha o arquivo após execução, liberando todos os recursos de volta ao sistema sem a necessidade do programador escrever muito código.

Input:

```
>>> import os
>>> os.chdir("C:/Users/Usuario/txt")
>>> with open("filosofia_da_educacão.txt", "r", encoding="utf-8") as arquivo:
...     print(arquivo.read())
```

Output:

A característica marcante do pensamento filosófico é a reflexão. A reflexão sobre o próprio conhecimento. Para a filosofia da educação podemos deduzir que a reflexão ocorre sobre o próprio processo educativo. Por exemplo, como as teorias e as práticas se dão no contexto do processo ensino-aprendizagem.

assert

The assert statement in Python is used for debugging.

Input:

```
>>> def divide(a, b):
...     assert b != 0, "Divisor não pode ser zero."
...     return a / b

>>> divide(5, 0)
```

Output:

```
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    divide(5, 0)
  File "<pyshell#47>", line 2, in divide
    assert b != 0, "Divisor não pode ser zero."
AssertionError: Divisor não pode ser zero.
```


finally

The finally statement is an optional part of try... except error. It always executes the code regardless of whether an error was thrown or not.

O comando **finally** é uma parte opcional do bloco **try ... except**. Ele sempre executa o código independente de um erro ter ocorrido ou não.

Input:

```
>>> a = 5
>>> b = 0
>>> try:
...     div = a / b
... except ZeroDivisionError:
...     print("Divisor não pode ser zero.")
... finally:
...     print("Tratamento de erro completo.")
```

Output:

```
Divisor não pode ser zero.
Tratamento de erro completo.
```

The print statement under the finally will always execute no matter if there is an error or not.

O comando print sob o comando **finally** irá sempre executar independente se houver erro ou não.

nonlocal

This keyword is used in functions inside functions to access **nonlocal** variables.

Esta palavra chave é usada em funções dentro de funções para acessar as variáveis não locais.

Input:

```
>>> def funcao_global():
...     x = "Valor Global"
...     def funcao_local():
...         nonlocal x
...         x = "Valor Local"
...     funcao_local()
...     return x

>>> print(funcao_global())
```

Output:

```
Valor Local
```

yield

The yield function ends a function and returns an iterator.

O comando **yield** finaliza uma função e retorna um “iterador”.

Input:

```
>>> def nums():
...     i = 0
...     while True:
...         yield i
...         i += 1

>>> for num in nums():
...     print(num)
...     if num > 10:
...         break
```

Output:

```
0
1
2
3
4
5
6
7
8
9
10
11
```

This is an infinity loop and will never end.

O “iterador” nunca finaliza, por isso, tivemos que colocar um comando **break** para finalizar o bloco **for**.

break

It is a control flow statement used to come out of a loop.

O comando **break** é um controlador de fluxo usado para sair a qualquer hora de um “loop” (laço).

Exemplo usado no comando **yield**.

for

The keyword is used to create a for loop.

Este comando é usado para criar um laço.

Input:

```
>>> num = [1, 2, 3]
>>> cubo = []
>>> for numero in num:
...     cubo.append(numero ** 3)
>>> print(cubo)
```

Output:

```
[1, 8, 27]
```

This loop will run till all the elements in the list are gone through it.

Este “laço” irá executar o comando `cubo.append(numero ** 3)` para cada um dos elementos na lista.

not

It is another logical operator that returns False when the value is True and vice versa.

The truth table for not operator:

Este é um outro operador lógico que retorna **False** quando o valor é **True** e vice-versa.

A tabela verdade para o operador **not**:

A	not A
True	False
False	True

class

The class keyword is used to define a class in Python.

A palavra chave **class** é usada para definir uma classe em Python.

from

This statement is used when you can to include a specific part of the module.
Este comando é usado quando for possível incluir apenas uma parte específica de um módulo.

Input:

```
>>> from math import sqrt
>>> print(sqrt(9))
```

Output:

```
3.0
```

The whole math module is not imported, only a specific function is imported into the project.

O módulo math não é totalmente importado, somente uma função específica é importada para o projeto.

or

It is a logical operator that returns True if any one of the statements is True.
Here is the truth table for or operator

É um operador lógico que retorna **True** se alguma das expressões for **True**.
Aqui é a tabela verdade para o operador **or**.

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Input:

```
>>> expressao1 = True
>>> expressao2 = False
>>> resultado = expressao1 or expressao2
>>> print(resultado)
```

Output:

```
True
```

One of the statements is True and according to the truth table, the or operator will return True.

Se uma das expressões retornar **True** (verdadeira), de acordo com a tabela verdade, o operador **or** retornará **True**.

continue

It is a control flow statement used to continue to the next iteration of a loop. Unlike break, the continue statement does not exit the loop.

Este comando de controle do fluxo do loop é usado para ir para a próxima iteração sem sair do “loop” (laço).

Input:

```
>>> eleitores = {"Maurício": 16, "André": 15, "Ricardo": 21}
>>> for chave in eleitores:
...     if eleitores[ chave ] >= 18:
...         print(f"{chave} é obrigado a votar.")
...     elif eleitores[ chave ] >= 16:
...         continue
...     else:
...         print(f"{chave} ainda não pode votar.")
```

Output:

```
André ainda não pode votar.
Ricardo é obrigado a votar.
```

The condition in the if statement may be satisfied by the second element. The user should print something, but it prefers to continue the loop.

A condição no comando **if** talvez seja satisfeita pelo segundo item do dicionário. O usuário poderia imprimir alguma coisa, mas ele prefere continuar o “loop” (laço).

global

Accessing a global variable is simple as any other variable but to modify a global variable, you need to use the global keyword.

Acessar uma variável global é simples como qualquer outra variável, mas para modificar uma variável global, você precisa usar a chave **global**.

Input:

```
>>> idade = 18
>>> def mudar_idade():
...     global idade
...     idade = 21
>>> mudar_idade()
>>> print(idade)
```

Output:

21

The `idade` variable is a global variable and we cannot change its value without using the global statement.

A variável `idade` é uma variável global e nós não podemos mudá-la sem usar o comando **global**.

pass

It is a null statement in Python that will do nothing.

É um comando nulo em Python que não faz coisa alguma.

Input:

```
>>> def sub(a, b):  
...     pass
```

```
>>> class Student:  
...     pass
```

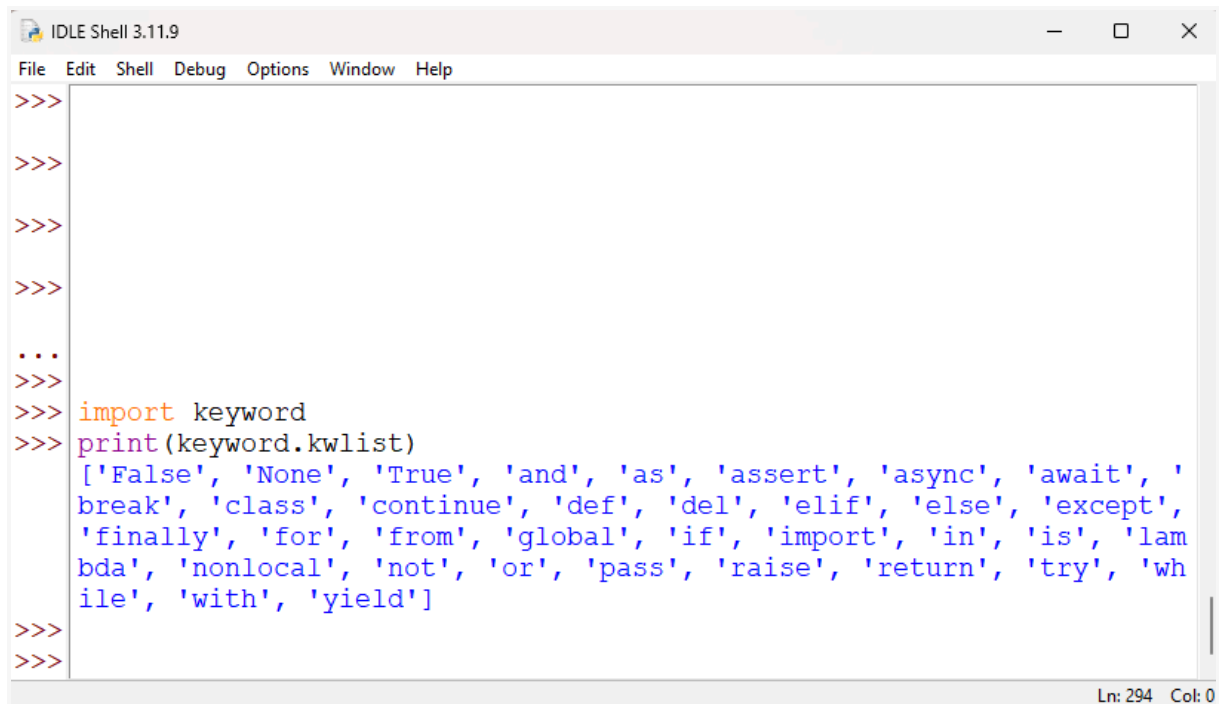
It is used as a placeholder for future code. It simply prevents getting errors when an empty code is run.

É usado para guardar um lugar para um código futuro. Ela simplesmente previne que o código seja executado sem erros.

Display all keywords

We can display the full list of all the keywords in the current version of Python by typing the following command in the Python interpreter.

Nós podemos mostrar a lista completa de palavras-chaves na versão corrente do Python, teclando o seguinte comando no interpretador do Python.

A screenshot of the IDLE Shell 3.11.9 window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows a Python prompt '>>>' followed by several empty lines, then '...', and then the command 'import keyword' followed by 'print(keyword.kwlist)'. The output is a list of 35 Python keywords: ['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']. The status bar at the bottom right shows 'Ln: 294 Col: 0'.

```
>>>
>>>
>>>
>>>
...
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
>>>
>>>
```

And to find out the number of reserved words in Python.
E para finalizar o número de palavras reservadas no Python.

Input:

```
>>> num_palavras = len(keyword.kwlist)
>>> print(f"No. de palavras chaves no Python é {num_palavras}")
```

Output:

No. de palavras chaves no Python é 35

Check if the name is included in the reserved word list in Python

To check if the name is a part of the list of reserved keywords in Python, we can use the `keyword.iskeyword()` function.

Para checar se uma palavra é parte da lista de palavras chaves no Python, nós podemos usar a função `iskeyword()`.

Input:

```
>>> import keyword
>>> print(keyword.iskeyword("global"))
>>> print(keyword.iskeyword("print"))
```

Output:

```
True
False
```

Closing Thoughts

Palavras Finais

Python reserved words designate special language functionality. No other [variable](#) can have the same name as these keywords. We read about all the reserved keywords and how to check if the name is a keyword or not. One can learn about more Python concepts [here](#).

As palavras reservadas em Python designam a funcionalidade especial da linguagem. Nenhuma outra variável pode ter o mesmo nome que essas palavras-chaves. Nós lemos sobre todas as palavras-chaves e checamos se um nome é uma palavra-chave ou não. Você pode aprender mais sobre a linguagem Python [aqui](#).