# **Python Keywords**

**Python Keywords:**[Introduction](https://www.geeksforgeeks.org/check-string-valid-keyword-python/)

**Keywords in Python** are reserved words that can not be used as a variable name, function name, or any other identifier.

## **List of all keywords in Python**

|  |  |  |  |
| --- | --- | --- | --- |
| and | as | [assert](https://www.geeksforgeeks.org/python-assert-keyword/) | [break](https://www.geeksforgeeks.org/python-break-statement/) |
| [class](https://www.geeksforgeeks.org/python-classes-and-objects/) | [continue](https://www.geeksforgeeks.org/python-continue-statement/) | def | [del](https://www.geeksforgeeks.org/python-del-to-delete-objects/) |
| elif | else | [except](https://www.geeksforgeeks.org/python-try-except/) | False |
| [finally](https://www.geeksforgeeks.org/finally-keyword-in-python/) | [for](https://www.geeksforgeeks.org/python-for-loops/) | from | [global](https://www.geeksforgeeks.org/global-keyword-in-python/) |
| if | [import](https://www.geeksforgeeks.org/import-module-python/) | in | is |
| [lambda](https://www.geeksforgeeks.org/python-lambda/) | None | nonlocal | not |
| or | [pass](https://www.geeksforgeeks.org/python-pass-statement/) | raise | [return](https://www.geeksforgeeks.org/python-return-statement/) |
| True | [try](https://www.geeksforgeeks.org/python-try-except/) | [while](https://www.geeksforgeeks.org/python-while-loop/) | [with](https://www.geeksforgeeks.org/with-statement-in-python/) |
| [yield](https://www.geeksforgeeks.org/python-yield-keyword/) |  |  |  |

We can also get all the keyword names using the below code.

### **Example: Python Keywords List**

|  |
| --- |
| # Python code to demonstrate working of iskeyword()    # importing "keyword" for keyword operations  importkeyword    # printing all keywords at once using "kwlist()"  print("The list of keywords is : ")  print(keyword.kwlist) |

**Output:**

The list of keywords is :

[‘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’]

Let’s discuss each keyword in detail with the help of good examples.

## **True, False, None**

* **True:** This keyword is used to represent a boolean true. If a statement is true, “True” is printed.
* **False:** This keyword is used to represent a boolean false. If a statement is false, “False” is printed.
* **None:** This is a special constant used to denote a null value or a void. It’s important to remember, 0, any empty container(e.g empty list) does not compute to None.   
  It is an object of its datatype – NoneType. It is not possible to create multiple None objects and can assign them to variables.

### **Example: True, False, and None Keyword**

|  |
| --- |
| print(False==0)  print(True==1)    print(True+True+True)  print(True+False+False)    print(None==0)  print(None==[]) |

**Output**True

True

3

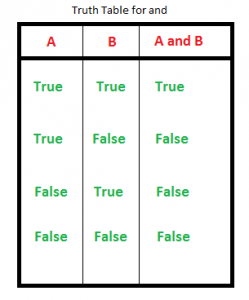
1

False

False

## **and, or, not, in, is**

* **and**: This a logical operator in python. “and” **Return the first false value. If not found return last**. The truth table for “and” is depicted below.



3 and 0 **returns 0**

3 and 10 **returns 10**

10 or 20 or 30 or 10 or 70 returns **10**

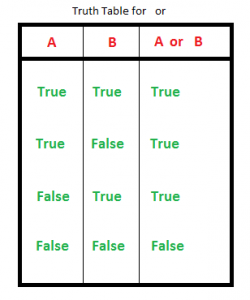
The above statements might be a bit confusing to a programmer coming from a language like **C** where the logical operators always return boolean values(0 or 1). Following lines are straight from the [python docs](https://docs.python.org/3/reference/expressions.html#boolean-operations) explaining this:

The expression x and y first evaluates x; if x is false, its value is returned; otherwise, y is evaluated and the resulting value is returned.

The expression x or y first evaluates x; if x is true, its value is returned; otherwise, y is evaluated and the resulting value is returned.

**Note** that neither and nor or restrict the value and type they return to False and True, but rather return the last evaluated argument. This is sometimes useful, e.g., if s is a string that should be replaced by a default value if it is empty, the expression s or ‘foo’ yields the desired value. Because not has to create a new value, it returns a boolean value regardless of the type of its argument (for example, not ‘foo’ produces False rather than ”.)

* **or**: This a logical operator in python. “or” Return the first True value.if not found return last. The truth table for “or” is depicted below.



3 or 0 **returns 3**

3 or 10 **returns 3**

0 or 0 or 3 or 10 or 0 returns **3**

* **not:** This logical operator inverts the truth value. The truth table for “not” is depicted below.
* **in:** This keyword is used to check if a container contains a value. This keyword is also used to loop through the container.
* **is:** This keyword is used to test object identity, i.e to check if both the objects take the same memory location or not.

### **Example: and, or, not, is and in keyword**

|  |
| --- |
| # showing logical operation  # or (returns True)  print(TrueorFalse)    # showing logical operation  # and (returns False)  print(FalseandTrue)    # showing logical operation  # not (returns False)  print(notTrue)    # using "in" to check  if's'in'geeksforgeeks':      print("s is part of geeksforgeeks")  else:      print("s is not part of geeksforgeeks")    # using "in" to loop through  fori in'geeksforgeeks':      print(i, end=" ")    print("\r")    # using is to check object identity  # string is immutable( cannot be changed once allocated)  # hence occupy same memory location  print(' 'is' ')    # using is to check object identity  # dictionary is mutable( can be changed once allocated)  # hence occupy different memory location  print({} is{}) |

**Output:**

True

False

False

s is part of geeksforgeeks

g e e k s f o r g e e k s

True

False

## **Iteration Keywords – for, while, break, continue**

* [**for**](https://www.geeksforgeeks.org/python-for-loops/)**:** This keyword is used to control flow and for looping.
* [**while**](https://www.geeksforgeeks.org/python-while-loop/)**:** Has a similar working like “for”, used to control flow and for looping.
* [**break**](https://www.geeksforgeeks.org/python-break-statement/)**:** “break” is used to control the flow of the loop. The statement is used to break out of the loop and passes the control to the statement following immediately after loop.
* [**continue**](https://www.geeksforgeeks.org/python-continue-statement/)**:** “continue” is also used to control the flow of code. The keyword skips the current iteration of the loop but does not end the loop.

### **Example: For, while, break, continue keyword**

|  |
| --- |
| # Using for loop  fori inrange(10):        print(i, end =" ")        # break the loop as soon it sees 6      ifi ==6:          break    print()    # loop from 1 to 10  i =0  whilei <10:        # If i is equals to 6,      # continue to next iteration      # without printing      ifi ==6:          i+=1          continue      else:          # otherwise print the value          # of i          print(i, end =" ")        i +=1 |

**Output**0 1 2 3 4 5 6

0 1 2 3 4 5 7 8 9

## **Conditional keywords – if, else, elif**

* **if** : It is a control statement for decision making. **Truth expression forces control to go in “if” statement block.**
* **else** : It is a control statement for decision making. **False expression forces control to go in “else” statement block.**
* **elif** : It is a control statement for decision making. It is short for “**else if**“

### **Example: if, else, and elif keyword**

|  |
| --- |
| # Python program to illustrate if-elif-else ladder  #!/usr/bin/python    i =20  if(i ==10):      print("i is 10")  elif(i ==20):      print("i is 20")  else:      print("i is not present") |

**Output**i is 20

**Note:** For more information, refer to out [Python if else Tutorial](https://www.geeksforgeeks.org/python-if-else/).

## **def**

def keyword is used to declare user defined functions.

### **Example: def keyword**

|  |
| --- |
| # def keyword  deffun():      print("Inside Function")    fun() |

**Output**Inside Function

## **Return Keywords – Return, Yield**

* [**return :**](https://www.geeksforgeeks.org/python-return-statement/) This keyword is used to return from the function.
* [**yield :**](https://www.geeksforgeeks.org/python-yield-keyword/) This keyword is used like return statement but is used to return a generator.

### **Example: Return and Yield Keyword**

|  |
| --- |
| # Return keyword  deffun():      S =0        fori inrange(10):          S +=i      returnS    print(fun())    # Yield Keyword  deffun():      S =0        fori inrange(10):          S +=i          yieldS    fori infun():      print(i) |

**Output**45

0

1

3

6

10

15

21

28

36

45

## **class**

[**class**](https://www.geeksforgeeks.org/python-classes-and-objects/) keyword is used to declare user defined classes.

### **Example: Class Keyword**

|  |
| --- |
| # Python3 program to  # demonstrate instantiating  # a class      classDog:        # A simple class      # attribute      attr1 ="mammal"      attr2 ="dog"        # A sample method      deffun(self):          print("I'm a", self.attr1)          print("I'm a", self.attr2)    # Driver code  # Object instantiation  Rodger =Dog()    # Accessing class attributes  # and method through objects  print(Rodger.attr1)  Rodger.fun() |

**Output**mammal

I'm a mammal

I'm a dog

**Note:** For more information, refer to our [Python Classes and Objects Tutorial](https://www.geeksforgeeks.org/python-classes-and-objects/) .

## **With**

[**with**](https://www.geeksforgeeks.org/with-statement-in-python/) keyword is used to wrap the execution of block of code within methods defined by context manager. This keyword is not used much in day to day programming.

### **Example: With Keyword**

|  |
| --- |
| # using with statement  with open('file\_path', 'w') as file:      file.write('hello world !') |

## **as**

**as** keyword is used to create the alias for the module imported. i.e giving a new name to the imported module. E.g import math as mymath.

### **Example: as Keyword**

|  |
| --- |
| importmath as gfg    print(gfg.factorial(5)) |

**Output**120

## **pass**

[**pass**](https://www.geeksforgeeks.org/python-pass-statement/)is the null statement in python. Nothing happens when this is encountered. This is used to prevent indentation errors and used as a placeholder.

### **Example: Pass Keyword**

|  |
| --- |
| n =10  fori inrange(n):    # pass can be used as placeholder  # when code is to added later  pass |

## **Lambda**

[**Lambda**](https://www.geeksforgeeks.org/python-lambda/) keyword is used to make inline returning functions with no statements allowed internally.

### **Example: Lambda Keyword**

|  |
| --- |
| # Lambda keyword  g =lambdax: x\*x\*x    print(g(7)) |

**Output**343

## **Import, From**

* [**import**](https://www.geeksforgeeks.org/import-module-python/) **:** This statement is used to include a particular module into current program.
* **from :** Generally used with import, from is used to import particular functionality from the module imported.

### **Example: Import, From Keyword**

|  |
| --- |
| # import keyword  importmath  print(math.factorial(10))    # from keyword  frommath importfactorial  print(factorial(10)) |

**Output**3628800

3628800

## **Exception Handling Keywords – try, except, raise, finally, and assert**

* [**try :**](https://www.geeksforgeeks.org/python-try-except/) This keyword is used for exception handling, used to catch the errors in the code using the keyword except. Code in “try” block is checked, if there is any type of error, except block is executed.
* [**except :**](https://www.geeksforgeeks.org/python-try-except/) As explained above, this works together with “try” to catch exceptions.
* [**finally :**](https://www.geeksforgeeks.org/finally-keyword-in-python/) No matter what is result of the “try” block, block termed “finally” is always executed.
* **raise:** We can raise an exception explicitly with the raise keyword
* [**assert:**](https://www.geeksforgeeks.org/python-assert-keyword/) This function is used for **debugging purposes**. Usually used to check the correctness of code. If a statement is evaluated to be true, nothing happens, but when it is false, “**AssertionError**” is raised. One can also **print a message with the error, separated by a comma**.

### **Example: try, except, raise, finally, and assert Keywords**

|  |
| --- |
| # initializing number  a =4  b =0    # No exception Exception raised in try block  try:      k =a//b # raises divide by zero exception.      print(k)    # handles zerodivision exception  exceptZeroDivisionError:      print("Can't divide by zero")    finally:      # this block is always executed      # regardless of exception generation.      print('This is always executed')    # assert Keyword  # using assert to check for 0  print("The value of a / b is : ")  assertb !=0, "Divide by 0 error"  print(a /b) |

**Output**

Can't divide by zero

This is always executed

The value of a / b is :

AssertionError: Divide by 0 error

**Note:** For more information refer to our tutorial [Exception Handling Tutorial in Python.](https://www.geeksforgeeks.org/python-exception-handling/)

## **del**

[**del**](https://www.geeksforgeeks.org/python-del-to-delete-objects/) is used to delete a reference to an object. Any variable or list value can be deleted using del.

### **Example: del Keyword**

|  |
| --- |
| my\_variable1 =20  my\_variable2 ="GeeksForGeeks"    # check if my\_variable1 and my\_variable2 exists  print(my\_variable1)  print(my\_variable2)    # delete both the variables  delmy\_variable1  delmy\_variable2    # check if my\_variable1 and my\_variable2 exists  print(my\_variable1)  print(my\_variable2) |

**Output**

20

GeeksForGeeks

NameError: name 'my\_variable1' is not defined

## **Global, Nonlocal**

* [**global:**](https://www.geeksforgeeks.org/global-keyword-in-python/) This keyword is used to define a variable inside the function to be of a global scope.
* **non-local :** This keyword works similar to the global, but rather than global, this keyword declares a variable to point to variable of outside enclosing function, in case of nested functions.

### **Example: Global and nonlocal keywords**

|  |
| --- |
| # global variable  a =15  b =10    # function to perform addition  defadd():      c =a +b      print(c)    # calling a function  add()    # nonlocal keyword  deffun():      var1 =10        defgun():          # tell python explicitly that it          # has to access var1 initialized          # in fun on line 2          # using the keyword nonlocal          nonlocal var1            var1 =var1 +10          print(var1)        gun()  fun() |

**Output**25

20

**Note:** Formore information, refer to our [Global and local variables tutorial in Python](https://www.geeksforgeeks.org/global-local-variables-python/).

This article is contributed by [**Manjeet Singh(S. Nandini)**](https://www.facebook.com/manjeet.04.singh). If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://write.geeksforgeeks.org/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.  
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.