KEYWORDS:

Every language contains words and a set of rules that would make a sentence meaningful. Similarly, in Python programming language, there are a set of predefined words, called Keywords which along with Identifiers will form meaningful sentences when used together. Python keywords cannot be used as the names of variables, functions, and classes.
In this article, we will learn about Python keywords and how to use them to perform some tasks.

Python Keywords

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

List of Keywords in Python

Keyword	Description	Keyword	Description	Keyword	Description
and	It is a Logical Operator	**False**	Represents an expression that will result in not being true.	**nonlocal**	It is a non-local variable
as	It is used to create an alias name	**finally**	It is used with exceptions	**not**	It is a Logical Operator
assert	It is used for debugging	**for**	It is used to create Loop	**or**	It is a Logical Operator
break	Break out a Loop	**from**	To import specific parts of a module	**pass**	pass is used when the user doesn't want any code to execute
class	It is used to define a class	**global**	It is used to declare a global variable	**raise**	raise is used to raise exceptions or errors.
continue	Skip the next iteration of a loop	**if**	To create a Conditional Statement	**return**	return is used to end the execution
def	It is used to define the Function	**import**	It is used to import a module	**True**	Represents an expression that will result in true.
del	It is used to delete an object	**is**	It is used to test if two variables are equal	**try**	Try is used to handle errors
elif	Conditional statements, same as else-if	**in**	To check if a value is present in a Tuple, List, etc.	**while**	While Loop is used to execute a block of statements

Keyword	Description	Keyword	Description	Keyword	Description
else	It is used in a conditional statement	lambda	Used to create an anonymous function	with	with statement is used in exception handling
except	try-except is used to handle these errors	None	It represents a null value	yield	yield keyword is used to create a generator function

Getting the List of all Python keywords

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

```
import keyword

# printing all keywords at once using "kwlist()"
print("The list of keywords is : ")
print(keyword.kwlist)
```

Output:

The list of keywords are:
['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 Keyword in Python

- **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.

True, False, and None Use in Python

- False is 0, and True is 1.
- True + True + True is 3.
- True + False + False is 1.
- None isn't equal to 0 or an empty list ([]).

```
print(False == 0)
print(True == 1)

print(True + True + True)
print(True + False + False)

print(None == 0)
print(None == [])
```

Output

True

True

3

1

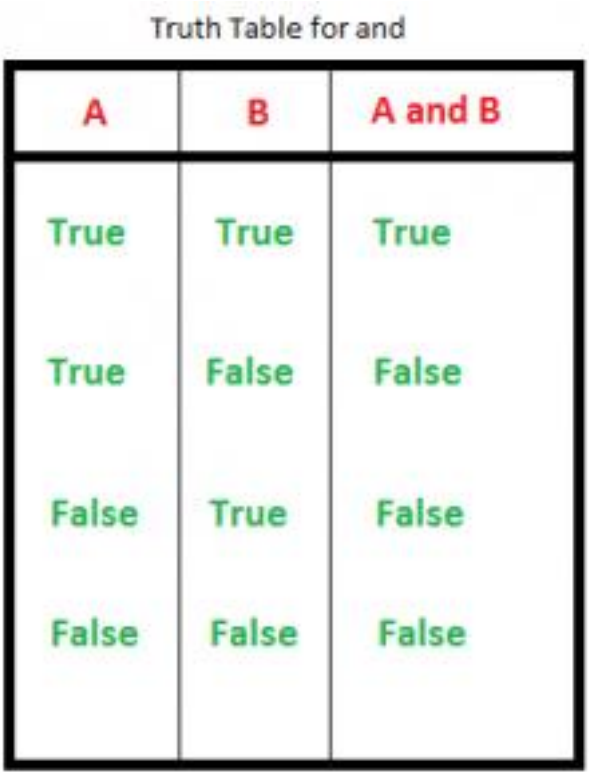
False

False

and, or, not, in, is In Python

Python and Keyword

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 **return 0**

3 and 10 **return 10**

OR

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). The following lines are straight from the Python docs 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 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 ".)

Python or Keyword

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.

Truth Table for or

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

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.

and, or, not, is and in keyword implementation in Python

The provided code demonstrates various Python operations:

1. **Logical Operations**:
   - **'or'** returns **'True'** when at least one operand is **'True'**.
   - **'and'** returns **'True'** only when both operands are **'True'**.
   - **'not'** negates the operand.
2. **Python "in" Keyword:**
   - It checks if 's' is in the string 'geeksforgeeks' and prints accordingly.
   - It loops through the string's characters.
3. **Python "is" Keyword:**
   - It checks if two empty strings (' ') are identical (returns **'True'**).
   - It checks if two empty dictionaries ({}) are identical (returns **'False'**).

```
print(True or False)
print(False and True)
print(not True)
if 's' in 'geeksforgeeks':
    print("s is part of geeksforgeeks")
else:
    print("s is not part of geeksforgeeks")
for i in 'geeksforgeeks':
    print(i, end=" ")
```

```
print("\r")
print(' ' is ' ')
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 in Python

- **for:** This keyword is used to control flow and for looping.
- **while:** Has a similar working like “for”, used to control flow and for looping.
- **break:** “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:** “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.

For, while, break, continue keyword Use in Python

The code includes a for loop and a while loop:

1. **For Loop:** Iterates from 0 to 9, printing numbers. It breaks when 6 is encountered.
2. **While Loop:** Initializes i to 0 and prints numbers from 0 to 9. It skips printing when i is 6 and continues to the next iteration.

```
for i in range(10):

    print(i, end=" ")
    if i == 6:
        break

print()
i = 0
while i < 10:
    if i == 6:
        i += 1
        continue
    else:
        print(i, end=" ")

    i += 1
```

Output

0 1 2 3 4 5 6

0 1 2 3 4 5 7 8 9

Conditional keywords in Python- 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**"

if, else, and elif keyword use in Python

The code checks the value of the variable i:

- If **'i'** is 10, it prints **"i is 10"**.
- If **'i'** is 20, it prints **"i is 20"**.
- If **'i'** is neither 10 nor 20, it prints **"i is not present."** In this case, it will print **"i is 20"** because the value of i is 20.

```
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.

Structure Keywords in Python : def, class, with, as, pass, lambda

Python def

def keyword is used to declare user defined functions.

def keyword in Python

The code defines a function named fun using the **def** keyword. When the function is called using fun(), it prints **"Inside Function."** This code demonstrates the use of the def keyword to define and call a function in Python.

```
def fun():
    print("Inside Function")


fun()
```

Output

Inside Function

class in Python

class keyword is used to declare user defined classes.

Class Keyword in Python

This code defines a Python class named **Dog** with two class attributes, **attr1** and **attr2**, and a method **fun** that prints these attributes. It creates an object **Rodge**r from the Dog class, accesses the class attributes, and calls the method. When executed, it prints the values of **attr1** and `**attr2**, and the method displays these values, resulting in the output shown.

```
class Dog:
    attr1 = "mammal"
    attr2 = "dog"

    def fun(self):
        print("I'm a", self.attr1)
        print("I'm a", self.attr2)

Rodger = Dog()
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 .

With in Python

with 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.

With Keyword in Python

This code demonstrates how to use the **with** statement to open a file named **'file_path'** in write mode **('w')**. It writes the text **'hello world !'** to the file and automatically handles the opening and closing of the file. The **with** statement is used for better resource management and ensures that the file is properly closed after the block is executed.

```
# using with statement
with open('file_path', 'w') as file:
    file.write('hello world !')
```

as in Python

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.

as Keyword In Python

This code uses the Python **math** module, which has been imported with the alias **gfg.** It calculates and prints the factorial of 5. The **math.factorial()** function is used to calculate the factorial of a number, and in this case, it calculates the factorial of 5, which is 120.

```
import math as gfg

print(gfg.factorial(5))
```

Output

120

pass in Python

pass is the null statement in python. Nothing happens when this is encountered. This is used to prevent indentation errors and used as a placeholder.

Pass Keyword in Python

The code contains a **for** loop that iterates 10 times with a placeholder statement **'pass'**, indicating no specific action is taken within the loop.

```
n = 10
for i in range(n):

    # pass can be used as placeholder
    # when code is to added later
    pass
```

Output

Lambda in Python

Lambda keyword is used to make inline returning functions with no statements allowed internally.

Lambda Keyword in Python

The code defines a lambda function **g** that takes an argument **x** and returns **x** cubed. It then calls this lambda function with the argument 7 and prints the result. In this case, it calculates and prints the cube of 7, which is 343.

```
# Lambda keyword
g = lambda x: x*x*x

print(g(7))
```

Output

343

Return Keywords in Python- Return, Yield

- **return :** This keyword is used to return from the function.
- **yield :** This keyword is used like return statement but is used to return a generator.

Return and Yield Keyword use in Python

The **'return'** keyword is used to return a final result from a function, and it exits the function immediately. In contrast, the '**yield**' keyword is used to create a generator, and it allows the function to yield multiple values without exiting. When '**return**' is used, it returns a single value and ends the function, while '**yield**'returns multiple values one at a time and keeps the function's state.

```
# Return keyword
def fun():
    S = 0
```

```
    for i in range(10):
        S += i
    return S


print(fun())

# Yield Keyword


def fun():
    S = 0

    for i in range(10):
        S += i
        yield S


for i in fun():
    print(i)
```

Output

45
0
1
3
6
10
15
21
28
36
45

Import, From in Python

import : 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.

Import, From Keyword use in Python

The **'import'** keyword is used to import modules or specific functions/classes from modules, making them accessible in your code. The **'from'** keyword is used with **'import'** to specify which specific functions or classes you want to import from a module. In your example, both approaches import the **'factorial'** function from the **'math'** module, allowing you to use it directly in your code.

```
# import keyword
from math import factorial
import math
print(math.factorial(10))

# from keyword
print(factorial(10))
```

Output

3628800

3628800

Exception Handling Keywords in Python – try, except, raise, finally, and assert

- **try :** 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 :** As explained above, this works together with "try" to catch exceptions.
- **finally :** 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:** 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**.

try, except, raise, finally, and assert Keywords use in Python

Example 1: The provided code demonstrates the use of several keywords in Python:

1. **try and except:** Used to handle exceptions, particularly the ZeroDivisionError, and print an error message if it occurs.
2. **finally:** This block is always executed, and it prints "This is always executed" regardless of whether an exception occurs.
3. **assert:** Checks a condition, and if it's False, raises an AssertionError with the message "Divide by 0 error."
4. **raise:** Raises a custom exception (TypeError) with a specified error message if a condition is not met.

```
a = 4
b = 0
try:
    k = a//b
    print(k)
except ZeroDivisionError:
    print("Can't divide by zero")

finally:
    print('This is always executed')

print("The value of a / b is : ")
assert b != 0, "Divide by 0 error"
print(a / b)

temp = "geeks for geeks"
if temp != "geeks":
    raise TypeError("Both the strings are different.")
```

Output

```
Can't divide by zero
This is always executed
The value of a / b is :
AssertionError: Divide by 0 error
```

Example 2: This code uses the raise keyword to raise a custom TypeError exception if two strings are not equal.

```
temp = "geeks for geeks"
if temp != "geeks":
    raise TypeError("Both the strings are different.")
```

Output

TypeError: Both the strings are different.

Note: For more information refer to our tutorial Exception Handling Tutorial in Python.

del in Python

del is used to delete a reference to an object. Any variable or list value can be deleted using del.

del Keyword in Python

In this code, the variables **my_variable1** and **my_variable2** are initially defined and then deleted using the **del** keyword. When you try to print them after deletion, you will encounter a **NameError**because the variables no longer exist.

```
my_variable1 = 20
my_variable2 = "GeeksForGeeks"
print(my_variable1)
print(my_variable2)
del my_variable1
del my_variable2
print(my_variable1)
print(my_variable2)
```

Output

20
GeeksForGeeks
NameError: name 'my_variable1' is not defined

Global, Nonlocal in Python

global: 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.

Global and nonlocal keywords in Python

In this code, the **'global'** keyword is used to declare global variables **'a'** and **'b'**. Then, there's a function **'add'** that adds these global variables and prints the result.

The second part of the code demonstrates the **'nonlocal'** keyword. The function **fun** contains a variable **var1**, and within the nested function gun, we use nonlocal to indicate that we want to modify the **var1** defined in the outer function **fun.** It increments the value of **var1** and prints it.

```
a = 15
b = 10
def add():
    c = a + b
    print(c)
add()
def fun():
    var1 = 10

    def gun():
        nonlocal var1

        var1 = var1 + 10
        print(var1)
```

```
    gun()
fun()
```

Output

25

20