**Python Keywords and Identifiers**

## Python Keywords

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False | await | else | import | pass |
| None | break | except | in | raise |
| True | class | finally | is | return |
| And | continue | for | lambda | try |
| As | def | from | nonlocal | while |
| Assert | del | global | not | with |
| Async | elif | if | or | yield |

## Python Identifiers

1. Identifiers can be a combination of letters in lowercase **(a to z)** or uppercase **(A to Z)** or digits **(0 to 9)** or an underscore \_. Names like myClass, var\_1 and print\_this\_to\_screen, all are valid example.
2. An identifier cannot start with a digit. 1variable is invalid, but variable1 is a valid name.
3. Keywords cannot be used as identifiers.

global = 1

# Python – Simple and Compound Statements

The statements which are meant for simple operations and mostly written in a single logical line of code.

**For example**, assignment statements are simple statements.

x = 10

which means, we are assigning a value “10” to the variable “x”. This we call as simple statement.

The computation statements (expression statements) also we call simple statements; these statements will compute or calculate some expressions and return the results.

**For example**, x = (10 + 15) is an expression statement.

Other than Assignment and Expression statements; the statements below also we called as Simple Statements: These are the statements formed with Python keyword(s); some of them are break, continue, return and import.

**break** Statement – We use ***break*** statement, to bypass the execution of the statements which are defined after the break statement. The execution control will go to end of the Compound Statement. Usually we use this statement, within the Compound Statements.

**continue** statement – **continue** statement is used to skip the statements execution which are defined after this statement. The execution control will go to the beginning of the Compound Statement. These statements also usually use with the Compound Statements.

Have you noticed the difference between break & continue statements? Control execution will go to the beginning of the Compound Statement when we use continue; where as for break, the control execution will go to end of the Compound Statement.

**return** statement -We use **return** statements within the function to return from the function with or without a value.

**import** statement – To import code modules to current namespace, we use **import** statement. Usually we write these statements at the beginning of the Program code.

### **Compound Statements**

A compound statement is a statement comprise of group of statements. The compound statements are usually executes, when a condition satisfies or a code block is called directly or through a function call. Compound Statements are spread into multiple logical lines; but aligned them into a particular group.

Class definitions and Function definitions are Compound Statements. I have explained more about Classes & Functions in my previous Article “”; and I do not want to explain here it again.

Other Compound Statements we have are:

#### **The conditional statement – The if statement**

The statements which are grouped with in the Conditional Compound Statement (**The if statement**) are going to execute when the particular condition is satisfied.

#### **Condition Loop Statements – The for statement AND The while statement**

**for** statement is used to iterate through the elements of a sequence; where as the statements within the **while** statement are going to execute when the condition is satisfied.

Using **while** statement also we can iterate through the elements of a sequence; but we need to write additional code to do this; whereas **for**statement syntax by default supports this.

#### **An Exception Handler – The try statement**

The group of statements with-in **try** are block are going to execute when an exception occurred. I will explain more about this in a separate Article.

|  |
| --- |
| #stmts\_example.py |
|  | # |
|  |  |
|  | # import statement |
|  | import math |
|  |  |
|  | x = 100 |
|  | index = 1 |
|  |  |
|  | # Display PI value |
|  | print("PI Value:\n", math.pi) |
|  |  |
|  | # conditional statement - The if statement |
|  | if ( x == 100 ): |
|  | x = x / 4 |
|  | print("\nThe result of (100/4) is:\n", x) |
|  |  |
|  |  |
|  | # The for statement |
|  | print("\n-- The for statement --\n") |
|  | print("Elements in the sequence are:") |
|  | sequence = [1, 2, 3, 4, 5] |
|  | for element in sequence: |
|  | print(element) |
|  |  |
|  |  |
|  | # The while statement |
|  | print("\n-- The while statement --\n") |
|  | print("Print only EVEN numbers:") |
|  | while(index < x): |
|  | if( ( index % 2 ) == 0 ): |
|  | print(index) |
|  |  |
|  | index = index + 1 |
|  |  |
|  | # The break & continue statements |
|  | print("\n-- The break & continue statements --\n") |
|  | print("Enter any value (0 - exit):") |
|  | while(1): |
|  | n = int(input()) |
|  | if ( n == 0 ): |
|  | break |
|  |  |
|  | # skip EVEN numbers to print |
|  | if( ( n % 2 ) == 0 ): |
|  | continue |
|  |  |
|  | print("You ENTERED the NUMBER : ", n) |
|  |  |
|  |  |
|  | # The try statement |
|  | print("-- The try statement --") |
|  | try: |
|  | div\_by\_0 = (1 / 0) |
|  | except: |
|  | print("Hurray!!! we caught, Divide / 0 Error") |

#### Values and types

A value is one of the most basic things in any program works with. A value may be characters i.e. ‘Hello, World!’ or a number like 1,2.2 ,3.5 etc.Values belong to different types: 1 is an integer, 2 is a float and ‘Hello, World!’ is a string etc.

###### Numbers:

###### Python supports 3 types of numbers: integers, float and complex number.

print(type(1))  
print(type(2.2))  
print(type(complex(2,3)))

----------------------------------------------------------------------------

###### Strings:

Strings are defined either with a single quote or a double quotes. The difference between the two is that using double quotes makes it easy to include apostrophes.

print(type('Hello World'))  
print(type("Today's News Paper"))

Python 3 provides a [Boolean data type](https://realpython.com/python-boolean/).

Objects of Boolean type may have one of two values, True or False:

type(True)

<class 'bool'>

>>> type(False)

<class 'bool'>

# Interpreter Vs Compiler : Differences Between Interpreter and Compiler

To convert source code into machine code, we use either a **compiler** or an **interpreter**.

## Interpreter Vs Compiler

|  |  |
| --- | --- |
| Interpreter | Compiler |
| Translates program one statement at a time. | Scans the entire program and translates it as a whole into machine code. |
| Interpreters usually take less amount of time to analyze the source code. However, the overall execution time is comparatively slower than compilers. | Compilers usually take a large amount of time to analyze the source code. However, the overall execution time is comparatively faster than interpreters. |
| No Object Code is generated, hence are memory efficient. | Generates Object Code which further requires linking, hence requires more memory. |
| Programming languages like JavaScript, Python, Ruby use interpreters. | Programming languages like C, C++, Java use compilers. |

## Working of Compiler and Interpreter

