Python Statement, Indentation and Comments

In this, you will learn about Python statements, why indentation is important and use of comments in programming.

Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement. if statement, for statement, while statement, etc. are other kinds of statements which will be discussed later.

Multi-line statement

In Python, the end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

This is an explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ], and braces { }. For instance, we can implement the above multi-line statement as:

```
a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)
```

Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

```
colors = ['red',
```

```
        'blue',
        'green']
```

We can also put multiple statements in a single line using semicolons, as follows:

```
a = 1; b = 2; c = 3
```

Python Indentation

Most of the programming languages like C, C++, and Java use braces { } to define a block of code. Python, however, uses indentation.

A code block (body of a function, loop, etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally, four whitespaces are used for indentation and are preferred over tabs. Here is an example.

```
for i in range(1,11):
    print(i)
    if i == 5:
        break
```

The enforcement of indentation in Python makes the code look neat and clean. This results in Python programs that look similar and consistent.

Indentation can be ignored in line continuation, but it's always a good idea to indent. It makes the code more readable. For example:

```
if True:
    print('Hello')
    a = 5
```

and

```
if True: print('Hello'); a = 5
```

both are valid and do the same thing, but the former style is clearer.

Incorrect indentation will result in IndentationError.

Python Comments

Comments are very important while writing a program. They describe what is going on inside a program, so that a person looking at the source code does not have a hard time figuring it out.

You might forget the key details of the program you just wrote in a month's time. So taking the time to explain these concepts in the form of comments is always fruitful.

In Python, we use the hash (**#**) symbol to start writing a comment.

It extends up to the newline character. Comments are for programmers to better understand a program. Python Interpreter ignores comments.

```
#This is a comment
#print out Hello
print('Hello')
```

Multi-line comments

We can have comments that extend up to multiple lines. One way is to use the hash(**#**) symbol at the beginning of each line. For example:

```
#This is a long comment
#and it extends
#to multiple lines
```

Another way of doing this is to use triple quotes, either ''' or """.

These triple quotes are generally used for multi-line strings. But they can be used as a multi-line comment as well. Unless they are not docstrings, they do not generate any extra code.

```
"""This is also a
perfect example of
multi-line comments"""
```

To learn more about comments, visit Python Comments.

Docstrings in Python

A docstring is short for documentation string.

Python docstrings (documentation strings) are the string literals that appear right after the definition of a function, method, class, or module.

Triple quotes are used while writing docstrings. For example:

```
def double(num):
    """Function to double the value"""
    return 2*num
```

Docstrings appear right after the definition of a function, class, or a module. This separates docstrings from multiline comments using triple quotes.

The docstrings are associated with the object as their __doc__ attribute.

So, we can access the docstrings of the above function with the following lines of code:

```
def double(num):
    """Function to double the value"""
    return 2*num
print(double.__doc__)
```

Output

Function to double the value