Python Basic Syntax with PDF

In this article, we will learn the **Python simple syntax**. This is one of the fundamental criteria for writing code in Python.

First Python Program

Let's understand how to write and run a simple first Python program before we move on to learning its syntax.

There are two different types of modes of programming in Python:

Interactive Mode Programming

Interactive mode is a command-line shell that **provides an instant response for each statement** while executing previously issued commands in active memory.

Example:

```
# Python Interactive Mode Programming

>>> print ("Hello, Python!");
```

Output:

```
# Result when you run the command

Hello, Python!
```

Script Mode Programming

The **Script Mode Programming**, the **Python code is written into a file**. The Python interpreter reads the file, then runs it and gives the desired result.

For example, let's write a straightforward Python application in a script. In the ***test.py*** file that you have created, type in the following source code:

```
# Inside the test.py file

print ("Python is FUN!")
```

We assume that the **PATH variable** contains the **Python interpreter**.
Now, try running this program this way:

```
# command prompt or shell - running a Python
```

```
C:\Python Tutorial>py test.py
```

Output:

```
# Result when you run the command

Pytho is FUN!
```

Python Identifiers

An **identifier in Python** is a name that can be used to point to a variable, function, class, module, or other objects.

Here are some rules for naming **Python identifiers:**

- You are not allowed to use reserved **keywords** or else it will throw **SyntaxError.**
- You can't name an identifier using numbers only.
- You can't start an identifier name with a number.
- The first letter of a class name is always **capitalized**. All the other identifiers start with a small letter.
- Identifiers are **case-sensitive**, "abc" and "ABC" are different identifiers.
- If an identifier starts with a single underscore, it means that it is a private identifier.
- When an identifier starts with two underscores, it means that it is very private.
- If the end of the identifier also has two underscores, it is a language-defined special name.

Python Keywords

The **Python keywords** are shown in the list below. These are reserved words, which means you can't use them as names for constants, variables, or anything else. There are no capital letters in any of the Python keywords.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while

else	is	with
except	lambda	yield

Lines and Indentations

The **white spaces in code lines are referred to as indentation**. Python's indentation is important, while indentation in other programming languages is just used to improve readability.

The number of spaces in the **indentation** can vary, but all of the statements in a block must have the same number of spaces in the indentation. Let's look at a simple example to see how the indentation works.

```
def sample_function():

  print("Hello, Python!")

  if True:

    print("Hi")

  else:

    print("Bye!")



print("Python is Fun!")
```

We're not going to talk about the code and what it does. Instead, let's look at the different levels of indentation.

```
# Functions

def sample_function():
```

It shows where the function starts, so every line of code that belongs to the function needs to be indented by at least one level. Notice that the print statements and the if statements are set back at least one level. But what about the last statement that says ***"print"***? We can see that it is not part of the function because it is not indented at all.

Now, let's go inside the function block, which is every statement that is at least one level away from the beginning of the function.

The first print statement is one level indented because it is only under the function and not under any other loop or condition. The second print statement, on the other hand, is part of the if statement.

```
if True:

  print("Hi")
```

The if statement is only indented one level, but any block of code that needs to be written under it should be indented one level more than the if statement. The same is also for the else statement.

Python Indention Rules:

- The **backslash (“/”)** character cannot be used to split an indentation into two or more lines.
- If you try to indent the first line of code in Python, an **IndentationError** will be thrown. You can’t make the first line of code look different.
- If you use a tab or a space to indent your code, it’s best to stick with the same spacing preference for the rest of your code. **Don’t use a mix of tabs and whitespaces**, because that could cause the wrong indentation.
- For the first level of indentation, it is best to use **1 tab** (or 4 whitespaces), and for each level after that, add 4 whitespaces (or 1 tab).

Python code looks better and more organized when it is indented. It gives you a good idea of how the code works with just one look. Also, the rules for indenting are simple, and if you’re writing code on an IDE, most of them will do it for you automatically.

One problem with indentation is that it can be hard to fix even a single line’s indentation mistake if your code is long and has a lot of different levels of indentation.

Let’s look at some indentation mistakes to make sure you understand.

```
>>>    p = 5

 File "<stdin>", line 1

   p = 5

IndentationError: unexpected indent
```

As we’ve already talked about, we can’t indent the first line, so we get an **IndentationError**.

```
if True:

  print("Inside True Statement")

   print("Will throw error because of extra whitespace")
```

```
else:
   print("False")
```

An IndentationError will be raised because the second print statement uses a single indent but contains whitespace that is not permitted in Python.

```
if True:
print "True"
else:
print "False"
```

IndentationError will be thrown because the print statements inside the if and else are not indented by one level.

```
#Error Message from  Python
IndentationError: expected an indented block
```

Quotation in Python

In **Python**, string objects are made with **quotation marks**. Python can read strings with **one, two, or three sets of quotes**.

Python supports three kinds of quotation marks:

- Single (‘)
- Double (“)
- Triple (”’ or “””)

They let the string go across more than one line. For instance, the following are all legal:

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

The triple quotation marks have already been used in the comments section. In Python, strings are declared with single and double quotes.

Newline Character in Python

The n character in Python is used to start a new line. This character tells the computer that the end of a line has been reached, and any more characters will be printed on a new line.

Example:

```
sample1 = "This is an example\nThis is the sentence with new line."

print(sample1)
```

Output:

```
This is an example

This is the sentence with new line.
```

Comments in Python

A line of text that occurs in a program but is not executed by the program is referred to as a **comment in Python**. Comments in Python start with the # sign. They can go at the start of a line, after a blank space, or after some code. If a string literal has the # character, it is part of the string.

```
# First comment

print ("Python is Fun!") # Second comment
```

This leads to the following:

```
Python is Fun!
```

You can comment on more than one line in the ways below:

```
# First comment

# Second comment

# Third comment

# Fourth comment
```

The Python interpreter also doesn't care about the following triple-quoted string, which can be used as a multiline comment:

```
'''

This is a multiline

comment.

'''
```

Using Blank Lines

A line that only has whitespace on it, possibly with a comment, is called a “**blank line**“, and Python doesn’t care about it at all.

To end a multiline statement in an interactive interpreter session, you must type an empty line.

Multiple Statements on a Single Line

To write **multiple statements in a single line in Python**, you have to use the **semicolon** (;). **Semicolon** lets you put more than one statement on one line, as long as none of the statements start a new code block. Here is an example of how to use a semicolon:

```
# using semicolon

import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Command Line Arguments

Command Line Arguments refers to the arguments given after the program’s name in the command line shell of the operating system. We use **command line arguments** to keep our program’s nature as general as possible.

For example, if we wrote a program to read a CSV file, entering a CSV file from the command line would make our program work for any CSV file, making it generic.

So, how do we pass arguments on the command line in Python?

It’s simple. You need to run the python script from the terminal the way we talked about at the beginning of the article and then add the inputs.

```
# argument sample

python test_file.py arg1 arg2 ... arg N
```

Here, ***test_file.py*** is the name of the script, and *arg1* through *argN* are the *N* arguments that must be given on the command line.

You must be wondering how one could read the command line arguments. The most common way is to use the sys module.

You can also set up your ***script*** so that it will accept different choices. Command Line Arguments is a more **advanced topic** that you should study after you've learned the rest of our Python Tutorial.

Summary

In Python syntax, we covered interactive and script modes in this tutorial. We talked about identifiers, keywords, lines and indentation, multi-line statements, quotes, comments, blank lines, user input, and command-line parameters.

I hope that this **Python Basic Syntax** helps you understand how to write and execute a Python program.