What is Python Syntax?

The Python syntax defines all the set of rules that are used to create sentences in Python programming.

For example – We have to learn grammar to learn the English language. In the same way, you will need to learn and understand the Python syntax in order to learn the Python language.

Example of Python Syntax

Python is a popular language because of its elegant syntax structure.

Let's take a quick look at a simple Python program and you will get an idea of how programming in Python looks like.

```
#Simple Python Program to see if a user is eligible to vote or not.

# getting user's name

print("Enter your name:")

name = input()


# getting user's age

print("Enter your age:")

age = int(input())


# condition to check if user is eligible or not

if( age >= 18 ):

   print( name, ' is eligible to vote.')

else:

   print( name, ' is not eligible to vote.')
```

Output

```
Enter your name:
Harsh
Enter your age:
19
Harsh is eligible to vote.
```

Types of Syntax Structures in Python

1. Python Line Structure

A Python program comprises logical lines. A **NEWLINE** token follows each of those. The interpreter ignores blank lines.

The following line causes an error.

>>> print("Hi

How are you?")

Output:

SyntaxError: EOL while scanning string literal

2. Python Multiline Statements

This one is an important Python syntax. We saw that Python does not mandate semicolons.

A new line means a new statement. But sometimes, you may want to split a statement over two or more lines.

It may be to aid readability. You can do so in the following ways.

a. Use a backward slash

>>> print("Hi\

how are you?")

Output:

Hihow are you?

You can also use it to distribute a statement without a string across lines.

>>> a\

=\

10

>>> print(a)

Output:

10

b. Put the String in Triple Quotes

>>> print("""Hi

how are you?""")

Output:

Hi
how are you?

However, you can't use backslashes inside a docstring for statements that aren't a string.

>>> """b

=

10"""

Output:

'b=10'

>>> print(b)

Output:

Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
print(b)
NameError: name 'b' is not defined

3. Python Comments

Python Syntax **'Comments'** let you store tags at the right places in the code. You can use them to explain complex sections of code.

The interpreter ignores comments. Declare a comment using an octothorpe (#).

>>> #This is a comment

Python does not support general multiline comments like **Java** or **C++**.

4. Python Docstrings

A docstring is a documentation string. As a comment, this Python Syntax is used to explain code.

But unlike comments, they are more specific. Also, they are retained at runtime.

This way, the programmer can inspect them at runtime. Delimit a docstring using three double-quotes. You may put it as a function's first line to describe it.

>>> def func():

"""

This function prints out a greeting

"""

print("Hi")

>>> func()

Output:

Hi

5. Python Indentation

Since Python doesn't use curly braces to delimit blocks of code, this Python Syntax is mandatory.

You can indent code under a function, loop, or class.

>>> if 2>1:

print("2 is the bigger person")

print("But 1 is worthy too")

Output:

2 is the bigger person
But 1 is worthy too

You can indent using a number of tabs or spaces, or a combination of those.

But remember, indent statements under one block of code with the same amount of tabs and spaces.

>>> if 2>1:

print("2 is the bigger person");

print("But 1 is worthy too");

Output:

SyntaxError: unindent does not match any outer indentation level

6. Python Quotations

Python supports the single quote and the double quote for string literals. But if you begin a string with a single quote, you must end it with a single quote.

The same goes for double-quotes.

The following string is delimited by single quotes.

>>> print('We need a chaperone');

Output:

We need a chaperone

This string is delimited by double-quotes.

>>> print("We need a 'chaperone'")

Output:

We need a ‘chaperone’

Notice how we used single quotes around the word chaperone in the string? If we used double quotes everywhere, the string would terminate prematurely.

>>> print("We need a "chaperone"")

Output:

SyntaxError: invalid syntax

8. Python Blank Lines

If you leave a line with just whitespace, the interpreter will ignore it.

9. Python Identifiers

An identifier is a name of a program element, and it is **user-defined**. This Python Syntax uniquely identifies the element.

There are some rules to follow while choosing an identifier:

- An identifier may only begin with A-Z, a-z, or an underscore(_).
- This may be followed by letters, digits, and underscores- zero or more.
- Python is case-sensitive. Name and name are two different identifiers.
- A reserved keyword may not be used as an identifier. The following is a list of keywords.

and	def	False	import	not	True
as	del	finally	in	or	try
assert	elif	for	is	pass	while
break	else	from	lambda	print	with
class	except	global	None	raise	yield
continue	exec	if	nonlocal	return	

Apart from these rules, there are a few naming conventions that you should follow while using this Python syntax:

- Use uppercase initials for class names, lowercase for all others.
- Name a private identifier with a leading underscore (_username)
- Name a strongly private identifier with two leading underscores (__password)
- Special identifiers by Python end with two leading underscores.

10. Python Variables

In Python, you don't define the type of the variable. It is assumed on the basis of the value it holds.

```
>>> x=10
>>> print(x)
```

Output:

```
10
>>> x='Hello'
>>> print(x)
```

Output:

Hello

Here, we declared a variable x and assigned it a value of 10. Then we printed its value. Next, we assigned it the value ‘Hello’ and printed it out.

So, we see, a variable can hold any type of value at a later instant. Hence, Python is a **dynamically typed language**.

11. Python String Formatters

Now let us see the different types of String formatters in Python:

a. % Operator

You can use the % operator to format a string to contain text as well as values of identifiers. Use %s where you want a value to appear.

After the string, put a % operator and mention the identifiers in parameters.

>>> x=10

printer="HP"

>>> print("I just printed %s pages to the printer %s" % (x, printer))

Output:

I just printed 10 pages to the printer HP

b. Format Method

The format method allows you to format a string in a similar way. At the places, you want to put values, put 0,1,2,.. in curly braces.

Call the format method on the string and mention the identifiers in the parameters.

>>> print("I just printed {0} pages to the printer {1}".format(x, printer))

Output:

I just printed 10 pages to the printer HP

You can also use the method to print out identifiers that match certain values.

>>> print("I just printed {x} pages to the printer {printer}".format(x=7, printer='HP'))

Output:

I just printed 7 pages to the printer HP

c. f-strings

If you use an f-string, you just need to mention the identifiers in curly braces. Also, write ‘f’ right before the string, but outside the quotes used.

>>> print(f"I just printed {x} pages to the printer {printer}")

Output:

I just printed 10 pages to the printer HP