02

August 26, 2019

#Print Statement
The **print** statement can be used in the following different ways :

```
- print("Hello World")
- print("Hello", <Variable Containing the String>)
- print("Hello" + <Variable Containing the String>)
- print("Hello %s" % <variable containing the string>)
```

```
In [1]: print("Hello World")
```

```
Hello World
```

In Python, single, double and triple quotes are used to denote a string. Most use single quotes when declaring a single character. Double quotes when declaring a line and triple quotes when declaring a paragraph/multiple lines.

```
In [2]: print('Hey')
```

```
Hey
```

```
In [4]: print("""My name is ABC and

        I love Python.""")
```

```
My name is ABC and

I love Python.
```

Strings can be assigned to variable say *string1* and *string2* which can called when using the print statement.

```
In [5]: string1 = 'World'
        print('Hello', string1)

        string2 = '!'
        print('Hello', string1, string2)
```

```
Hello World
Hello World !
```

String concatenation is the “addition” of two strings. Observe that while concatenating there will be no space between the strings.

```
In [6]: print('Hello' + string1 + string2)
```

```
HelloWorld!
```

%s is used to refer to a variable which contains a string.

```
In [7]: print("Hello %s" % string1)
```

```
Hello World
```

Similarly, when using other data types

```
- %s -> string
- %d -> Integer
- %f -> Float
- %o -> Octal
- %x -> Hexadecimal
- %e -> exponential
```

This can be used for conversions inside the print statement itself.

```
In [8]: print("Actual Number = %d" %18)
        print("Float of the number = %f" %18)
        print("Octal equivalent of the number = %o" %18)
        print("Hexadecimal equivalent of the number = %x" %18)
        print("Exponential equivalent of the number = %e" %18)
```

```
Actual Number = 18
Float of the number = 18.000000
Octal equivalent of the number = 22
Hexadecimal equivalent of the number = 12
Exponential equivalent of the number = 1.800000e+01
```

When referring to multiple variables parenthesis is used.

```
In [9]: print("Hello %s %s" %(string1,string2))
```

```
Hello World !
```

##Other Examples
The following are other different ways the print statement can be put to use.

```
In [10]: print("I want %%d to be printed %s" %'here')
```

```
I want %d to be printed here
```

```
In [11]: print('_A'*10)
```

```
_A_A_A_A_A_A_A_A_A_A
```

```
In [12]: print("Jan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug")
```

```
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
```

```
In [13]: print("I want \\n to be printed.")
```

```
I want \n to be printed.
```

```
In [15]: print("""
         Routine:
         \t- Eat
         \t- Sleep\n\t- Repeat
         """)
```

```

Routine:
        - Eat
        - Sleep
        - Repeat

```

#PrecisionWidth and FieldWidth
Fieldwidth is the width of the entire number and precision is the width towards the right. One can alter these widths based on the requirements.
The default Precision Width is set to 6.

```
In [16]: "%f" % 3.121312312312
```

```
Out[16]: '3.121312'
```

Notice upto 6 decimal points are returned. To specify the number of decimal points, '%(fieldwidth).(precisionwidth)f' is used.

```
In [17]: "%.5f" % 3.121312312312
```

```
Out[17]: '3.12131'
```

If the field width is set more than the necessary than the data right aligns itself to adjust to the specified values.

```
In [18]: "%9.5f" % 3.121312312312
```

```
Out[18]: '  3.12131'
```

Zero padding is done by adding a 0 at the start of fieldwidth.

```
In [19]: "%020.5f" % 3.121312312312
```

```
Out[19]: '00000000000003.12131'
```

For proper alignment, a space can be left blank in the field width so that when a negative number is used, proper alignment is maintained.

```
In [20]: print("% 9f" % 3.121312312312)
         print("% 9f" % -3.121312312312)
```

```
 3.121312
-3.121312
```

'+' sign can be returned at the beginning of a positive number by adding a + sign at the beginning of the field width.

```
In [21]: print("%+9f" % 3.121312312312)
         print("% 9f" % -3.121312312312)
```

```
+3.121312
-3.121312
```

As mentioned above, the data right aligns itself when the field width mentioned is larger than the actualy field width. But left alignment can be done by specifying a negative symbol in the field width.

```
In [22]: "%-9.3f" % 3.121312312312
```

```
Out[22]: '3.121    '
```