002 Python print() function

print() function

The print statement has been replaced with a print() function, with keyword arguments to replace most of the special syntax of the old print statement.

The print statement can be used in the following ways :

- print("Good Morning")
- print("Good", <Variable Containing the String>)
- print("Good" + <Variable Containing the String>)
- print("Good %s" % <variable containing the string>)

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.

Commands

```
print(<el_1>, ..., sep=' ', end='\n', file=sys.stdout,
flush=False)
```

- **Use 'file=sys.stderr' for errors.**
- **Use 'flush=True' to forcibly flush the stream.**

Pretty Print

```
from pprint import pprint
pprint(<collection>, width=80, depth=None)
```

- **Levels deeper than 'depth' get replaced by '...'.**

Double Quotes Use:

Example:

```
print("Python is very simple language")
```

Output:

```
Python is very simple language
```

Single Quotes Use:

Example:

```
print('Hello')
```

Output:

```
Hello
```

Triple Quotes Use:

Example:

```
print("""Python is very popular language.

It is also friendly language.""")
```

Output:

```
Python is very Popular Language.
It is also friendly language.
```

Variable Use:

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

Example:

```
str1 = 'Wel'

print(str1,'come')
```

Output:

```
Wel come
```

Example:

```
str1 = 'Welcome'
```

```python
str2 = 'Python'

print(str1, str2)
```

Output:

```
Welcome Python
```

String Concatenation:

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

Example:

```python
str1 = 'Python'

str2 = ':'

print('Welcome' + str1 + str2)
```

Output:

```
WelcomePython:
```

Using as String:

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

Example:

```python
str1 = 'Python'

print("Welcome %s" % str1)
```

Output:

```
Welcome Python
```

Using other data types:

Similarly, when using other data types

- %d -> Integer

- %e -> exponential
- %f -> Float
- %o -> Octal
- %x -> Hexadecimal

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

Using as Integer:

Example:

```
print("Actual Number = %d" %15)
```

Output:

```
Actual Number = 15
```

Using as Exponential:

Example:

```
print("Exponential equivalent of the number = %e" %15)
```

Output:

```
Exponential equivalent of the number = 1.500000e+01
```

Using as Float:

Example:

```
print("Float of the number = %f" %15)
```

Output:

```
Float of the number = 15.000000
```

Using as Octal:

Example:

```
print("Octal equivalent of the number = %o" %15)
```

Output:

```
Octal equivalent of the number = 17
```

Using as Hexadecimal:

Example:

```
print("Hexadecimal equivalent of the number = %x" %15)
```

Output:

```
Hexadecimal equivalent of the number = f
```

Using multiple variables:

When referring to multiple variables parenthesis is used.

Example:

```
str1 = 'World'

str2 = ':'

print("Python %s %s" %(str1,str2))
```

Output:

```
Python World :
```

Other Examples of Print Statement:

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

Example-1:

% is used for %d type word

```
print("Welcome to %%Python %s" %'language')
```

Output:

```
Welcome to %Python language
```

Example-2:

\n is used for Line Break.

```
print("Sunday\nMonday\nTuesday\nWednesday\nThursday\nFriday\nSaturday"
)
```

Output:

```
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```

Example-3:

Any word print multiple times.

```
print('-gog'*5)
```

Output:

```
-gog-gog-gog-gog-gog
```

Example-4:

\t is used for tab.

```
print("""

Language:

\t1 Python

\t2 Java\n\t3 JavaScript

""")
```

Output:

```
Language:
        1 Python
        2 Java
        3 JavaScript
```

Precision Width and Field Width:

Field width 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.

Example-1:

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

```
print("%f" % 5.1234567890)
```

Output:

```
5.123457
```

Example-2:

Notice upto 5 decimal points are returned

```
print("%.5f" % 5.1234567890)
```

Output:

```
5.12346
```

Example-3:

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

```
print("%9.5f" % 5.1234567890)
```

Output:

```
  5.12346
```

Example-4:

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

```
print("%015.5f" % 5.1234567890)
```

Output:

```
000000005.12346
```

Example-5:

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.

```
print("% 9f" % 5.1234567890)

print("% 9f" % -5.1234567890)
```

Output:

```
 5.123457
-5.123457
```

Example-6:

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

```
print("%+9f" % 5.1234567890)

print("% 9f" % -5.1234567890)
```

Output:

```
+5.123457
-5.123457
```

Example-7:

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

```
print("%-9.4f" % 5.1234567890)
```

Output:

```
5.1235
```