String Formatting in Python:
In Python a string of required formatting can be achieved by different methods.
1) Using %
2) Using {}

Using %: The formatting using % is similar to that of ‘printf’ in C programming language.

```
%d ==> integer
%f ==> float
%s  ==> string
%x ==> hexadecimal
%o ==> octal
```

Syntax:

```
print("formatted string" %(variable list))
```

Example:

```
print("%s having %s Years Experience in IT." %("KSRAJU","15+"))
print("%s having %d Years Experience in IT." %("KSRAJU",15))
```

Example:

```
name = "RaaJ"

print("Hello, %s!" % name)
```

Example:

```
name = "RaaJ"

age = 23

print("%s is %d years old." % (name, age))
```

Example:

```
a=10; b=20; c=30
print("a value is %i" %a)
print("b value is %d and c value is %d" %(b,c))
```

Formatting Strings:
We can format strings with variable values by using replacement operator {} and format() method.

Syntax:

```
{ } .format(value)
```

value :
Can be an integer, floating point numeric constant, string, characters or even variables.

Example:

```
name='python'
print("Hello {}".format(name))
```

Syntax :

```
{ } { } .format(value1, value2)
```

Example:
```
name='ksraju';dept='Software'
print("Hello {} You are {} department".format(name,dept))
```

Syntax :
```
{ } { } { } .format(value1, value2,value3)
```

Example:
```
PyName='SMITH';PySal="$10000";PyLoc="HYDERABAD"
print("{}'s Sal is: {} & His Loc is: {}".format(PyName,PySal,PyLoc))
print("{0}'s Sal is: {1} & His Loc is:
{2}".format(PyName,PySal,PyLoc))
```

Printing multiple variables
There are following methods to print multiple variables,
Method 1: Passing multiple variables as arguments separating them by commas
Method 2: Using format() method with curly braces ({})
Method 3: Using format() method with numbers in curly braces ({0})
Method 4: Using format() method with explicit name in curly braces ({v})
Method 5: Using string concatenation

Method 1:
To print multiple variables using the print() function, we need to provide the variable names as arguments separated by the commas.

Syntax:
```
print(variable1, varaible2, variable3, ...)
```

Example:
```
# Python program to print multiple variables
name = "RajuSir"; age = 45; country = "INDIA"
#Printing variables one by one
print("Printing Normally...")
print(name, age, country)
print() # prints a new line
# Printing with comma seprator
print("Printing with comma seprator...")
print(name, age, country, sep=',')
print() # prints a new line
# printing variables with messages
print("Printing with messages...")
print("Name:", name, "Age:", age, "Country:", country)
```

Method 2:
By using the new-style string formatting (format() method), we can also print the multiple variables. Here, we have to specify the curly braces ({}) where we have to print the values and in the format() method, provide the multiple variables separated by the commas.

Syntax:
```
print("{} {} {}".format(variable1, variable2, variable2)
```

Example:
```
# Python program to print multiple variables
# using format() method
```

```
name = "RajuSir"; age = 45; country = "INDIA"
print("{} {} {}".format(name, age, country))
print("Name: {}, Age: {}, Country: {}".format(name, age, country))
```

Method 3:
By using the new-style string formatting with numbers (format() method), we can also print the multiple variables. This is similar to method 2 but here we can use the numbers inside the curly braces ({0}), it will help for reordering the values.

NOTE
Number 0 represents the first variable in format() method, 1 represents the second, and so on.

Syntax:
```
print("{0} {1} {2}".format(variable1, variable2, variable2)
```

Example:
```
# Python program to print multiple variables
# using format() method with numbers
name = "RajuSir"; age = 45; country = "INDIA"
print("{0} {1} {2}".format(name, age, country))
print("Name: {0}, Age: {1}, Country: {2}".format(name, age, country))
print("Country: {2}, Name: {0}, Age: {1}".format(name, age, country))
# printing all values 2-2 times
print("{0} {0} {1} {1} {2} {2}".format(name, age, country))
```

Method 4:
By using the new-style string formatting with explicit names (format() method), we can also print the multiple variables. This is similar to method 3 but here we can use the explicit names inside the curly braces ({n}), it will help for remembering the order and variable names.

Syntax:
```
print("{v1} {v2} {v3}".format(v1=variable1, v2=variable2,
v3=variable2)
```

Example:
```
# Python program to print multiple variables
# using format() method with explicit names
name = "RajuSir"; age = 41; country = "INDIA"
print("{n} {a} {c}".format(n=name, a=age, c=country))
print("Name: {n}, Age: {a}, Country: {c}".format(n=name, a=age,
c=country))
print("Country: {c}, Name: {n}, Age: {a}".format(n=name, a=age,
c=country))
# printing all values 2-2 times
print("{n} {n} {a} {a} {c} {c}".format(n=name, a=age, c=country))
```

Method 5:
We can print multiple variables by concatenating them as a string.

Syntax:
```
print(str(variable1) + str(variable2) + str(variable3))
```

Note:

If we want to display any message or separator, we can also concatenate them with the variables.
If a variable is a string, then there is no need to use str().

Example:
```
# Python program to print multiple variables
# using string concatenation

name = "RajuSir"; age = 45; country = "INDIA"
print("Without separator...")
print(name + str(age) + country)
print("Separating by commas...")
print(name + "," + str(age) + "," + country)
print("Printing with messages...")
print("Name: " + name + " Age: " + str(age) + " Country: " + country)
```

What is Indentation:
Indentation is a way of telling the Python interpreter that a series of statements belong to a particular block of code. In languages like C, C++, Java, we use curly braces { } to indicate the start and end of a block of code. In Python, we use space/tab as indentation to indicate the same to the compiler.

1 Increase indent after an if statement or for statement (after :)
2 Maintain indent to indicate the scope of the block
3 Blank lines are ignored - they do not affect indentation
4 Comments on a line by themselves are ignored with regard to indentation

Example:
```
if True:
    print("True")
else:
    print("False")
```

WORKING WITH PYTHON OPERTAORS
An operator is a character that represents an action. Python Supports different types of Operators:

1 Arithmetical Operators
2 Comparison (Relational) Operators
3 Assignment Operators
4 Logical Operators
5 Membership Operators
6 Identity Operators
7 Bitwise Operators
8 Ternary Operator

Python Arithmetical Operators:
Arithmetic operators are used to perform various arithmetic / mathematical operations

Operator	Name
+	Addition Operator
-	Subtraction Operator
*	Multiplication Operator
**	Power Operator
/	Division Operator

```
%       Modulus Operator
//      Floor Division Operator
```

Syntax:

```
exp1  +  exp2
exp1  -  exp2
exp1  *  exp2
exp1 **  exp2
exp1  /  exp2
exp1 //  exp2
exp1  %  exp2
```

Example:

```
x = 15
y = 4
print('x + y =',x+y)
print('x - y =',x-y)
print('x * y =',x*y)
print('x / y =',x/y)
print('x // y =',x//y)
print('x ** y =',x**y)
```

Example:

```
x=int(input("Enter Any Number: "))
y=int(input("Enter Any Number: "))
z=x+y
print("Sum is: ",z)
z=x-y
print("Diff is: ",z)
z=x*y
print("Product is: ",z)
z=x/y
print("Div is: ",z)
z=x%y
print("Mod is: ",z)
z=x**y
print("Expo is: ",z)
z=x//y
print("Fdiv is: ",z)
```

Python Comparison Operators

These operators are used to compare values. They are also called Relational operators.

```
Operator                Meaning
Example
>       Greater that - True if left operand is greater than the right
x > y
<       Less that - True if left operand is less than the right
x < y
==      Equal to - True if both perands are equal
x == y
!=      Not equal to - True if operands are not equal
x != y
>=      True if left operand is greater than or equal to the right
x >= y
<=      True if left operand is less than or equal to the right
```

```
x <= y

Example:
A=1;B=2
print(A==B)#False
print(A!=B)#True
print(A<B)#True
print(A>B)#False
print(A<=B)#True
print(A>=B)#False

Example:
x = 10
y = 12
# Output: x > y is False
print('x > y  is',x>y)
# Output: x < y is True
print('x < y  is',x<y)
# Output: x == y is False
print('x == y is',x==y)
# Output: x != y is True
print('x != y is',x!=y)
# Output: x >= y is False
print('x >= y is',x>=y)
# Output: x <= y is True
print('x <= y is',x<=y)

Python Assignment Operators
Operator        Description
Example
=       Assigns values from right side operands to left side operand
c = a + b assigns value of a + b into c
+=      Add AND It adds right operand to the left operand and assign
the result to left operand                      c += a is equivalent
to c = c + a
-=      Subtract AND It subtracts right operand from the left operand
and assign the result to left operand           c -= a is equivalent
to c = c - a
*=      Multiply AND It multiplies right operand with the left operand
and assign the result to left operand           c = a is equivalent to
c = c a
/=      Divide AND It divides left operand with the right operand and
assign the result to left operand               c /= a is equivalent
to c = c / a
%=      Modulus AND It takes modulus using two operands and assign the
result to left operand                  c %= a is equivalent to c = c
% a
**=     Exponent AND Performs exponential (power) calculation on
operators and assign value to the left operand  c = a is equivalent to
c = c a
//=     Floor Division It performs floor division on operators and
assign value to the left operand                        c //= a is
equivalent to c = c // a


Example:
a = 21
```

```
b = 10
c = 0

c = a + b
print ("Line 1 - Value of c is ", c)

c += a
print ("Line 2 - Value of c is ", c )

c *= a
print ("Line 3 - Value of c is ", c )

c /= a
print ("Line 4 - Value of c is ", c )

c  = 2
c %= a
print ("Line 5 - Value of c is ", c)

c **= a
print ("Line 6 - Value of c is ", c)

c //= a
print ("Line 7 - Value of c is ", c)
```

Python Logical(Boolean) Operators
Python supports the following list of logical Operators:

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example:

```
x = True; y = False
# Output: x and y is False
print('x and y is',x and y)

# Output: x or y is True
print('x or y is',x or y)

# Output: not x is False
print('not x is',not x)
```

Example: With strings,
For "and" operator: If the first operand is True, it checks the second operand and returns the second operand.
For "or" operator: If the first operand is False, it checks the second operand and returns the second operand.
For "and" operator: If the operand is an empty string, it returns True; False, otherwise.

Example:

```
PyStr1 = "Hello"; PyStr2 = "World"
```

```
# and operator on string
print("PyStr1 and PyStr2: ", PyStr1 and PyStr2)
print("PyStr2 and PyStr1: ", PyStr2 and PyStr1)
print()

# or operator on string
print("PyStr1 or PyStr2: ", PyStr1 or PyStr2)
print("PyStr2 or PyStr1: ", PyStr2 or PyStr1)
print()

# not operator on string
print("not PyStr1: ", not PyStr1)
print("not PyStr2: ", not PyStr2)
print()
```

Example:
Python repr()
It returns a printable representation of the given object.

Syntax:

```
repr(obj)
```

Example:

```
PyStr = 'Hello Python'
print (repr(PyStr))
```

Example:

```
PyStr1 = "" # empty string
PyStr2 = "World" # non-empty string

# and operator on string
print("PyStr1 and PyStr2: ", repr(PyStr1 and PyStr2))
print("PyStr2 and PyStr1: ", repr(PyStr2 and PyStr1))
print()

# or operator on string
print("PyStr1 or PyStr2: ", repr(PyStr1 or PyStr2))
print("PyStr2 or PyStr1: ", repr(PyStr2 or PyStr1))
print()

# not operator on string
print("not PyStr1: ", not PyStr1)
print("not PyStr2: ", not PyStr2)
print()
```

Python Special Operators:
Python Scripting Language offers two types of Special Operators:
1. Membership operators 2. Identity operators

Python Membership Operators
They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

NOTE: In a dictionary we can only test for presence of key, not the value.

Operator   Meaning

```
in      True if value/variable is found in the sequence
not in  True if value/variable is not found in the sequence
```

Examle:

```
PyStr="PYTHON"
print('P' in PyStr)
print('P' not in PyStr)
print('p' not in PyStr)
print('N' not in PyStr)
```

Example:

```
x = 'Hello world'
y = {1:'a',2:'b'}
print('H' in x)
print('hello' not in x)
print(1 in y)
print('a' in y)
```