Operators and Expressions in Python

- An Operator is a Symbol, which will perform some Operation on Objects / Variables / values.
- An Expression is a Collection of Objects / variables / Values Connected with an Operator.
- In Python Programming, we have 7 Types of Operators . They are

```
1. Arithmetic Operators
2. Assigment Operator
3. Relational Operators ( Comparision Operator )
4. Logical Operators ( Comparision Operator )
5. Bitwise Operators---Most Imp
6. Membership Operators
        a) in
        b) not in
7. Identity Operators
       a) is
       b) is not
```

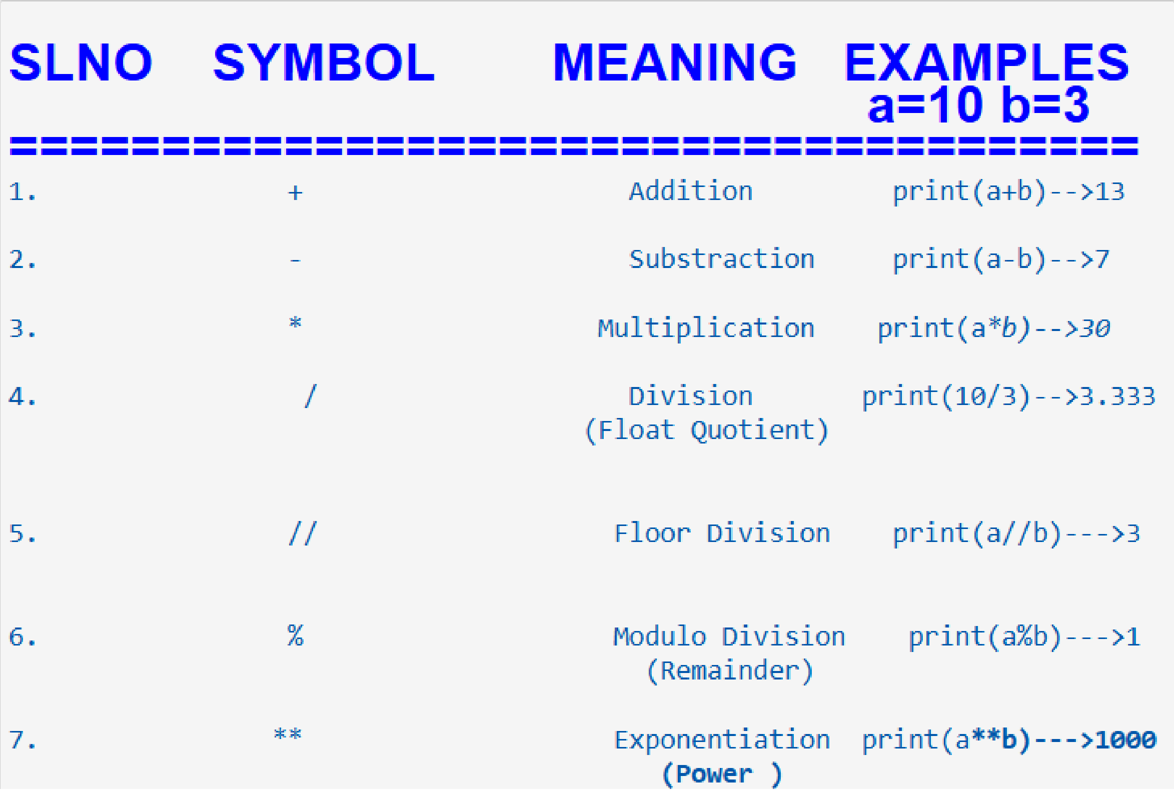
1. Arithmetic Operators

- The purpopse of Arithmetic Operators is that "To perform Various Arithmetic Operations such as Addition, Substraction, Multiplication..etc"

- If One or More Arithmetic Operators Connected with Object / Variables / Values then It is Called Arithmetic Expression.

- In Python Programming, we have 7 Types of Arithmetic Operators. They are given in the following Table.

```
1. Addition
2. Substraction
3. Multiplication
4. Division
5. Division
6. Modulo Division
7. Exponentiation
```



SLNO	SYMBOL	MEANING	EXAMPLES a=10 b=3
1.	+	Addition	print(a+b)-->13
2.	-	Substraction	print(a-b)-->7
3.	*	Multiplication	print(a*b)-->30
4.	/	Division (Float Quotient)	print(10/3)-->3.333
5.	//	Floor Division	print(a//b)--->3
6.	%	Modulo Division (Remainder)	print(a%b)--->1
7.	**	Exponentiation (Power)	print(a**b)--->1000

ADDITION

In [4]:

```
a = 10
b = 20
c = a + b
print(c)
```

```
30
```

Substraction

In [5]:

```
a = 20
b = 10
c = a - b
print(c)
```

```
10
```

Multiplication

In [6]:

```
a = 20
b = 3
c = a * b
print(c)
```

```
60
```

Division

In [7]:

```
a = 10
b = 3
```

```
c = 10/3
print(c)  # it wii be give float quotient
```

```
3.3333333333333335
```

Floor Division

In [10]:

```
a = 10
b = 3
c = 10//3
print(c)    # it will be give integer quotient
```

```
3
```

Modulo Division

In [11]:

```
a = 10
b = 3
c = a%b
print(c)   # it wii be give remainder
```

```
1
```

Exponentiation

In [12]:

```
a = 10
b = 3
c = a ** b
print(c) # it wii  GIVE power
```

```
1000
```

2. Assigment Operator

- The purpose of assignment operator is that "To assign or transfer Right Hand Side (RHS) Value / Expression Value to the Left Hand Side (LHS) Variable" Value / Expression Value to the Left Hand Side (LHS) Variable"
- The Symbol for Assigment Operator is single equal to (=).
- In Python Programming,we can use Assigment Operator in two ways.

```
                1. Single Line Assigment
                2. Multi Line Assigment
```

1. Single Line Assigment

- Syntax: LHS Varname= RHS Value

```
          LHS Varname= RHS Expression
```

- With Single Line Assigment at a time we can assign one RHS Value / Expression to the single LHS Variable Name.

In [13]:

```
a = 10
b = 20
c = a+b
print(a,b,c)
```

```
10 20 30
```

2. Multi Line Assigment:

- Syntax: Var1,Var2.....Var-n= Val1,Val2....Val-n

```
Var1,Var2.....Var-n= Expr1,Expr2...Expr-n
```

- Here The values of Val1, Val2...Val-n are assigned to Var1,Var2...Var-n Respectively.
- Here The values of Expr1, Expr2...Expr-n are assigned to Var1,Var2...Var-n Respectively.

In [14]:

```
a,b=10,20
print(a,b)
c,d,e=a+b,a-b,a*b
print(c,d,e)
```

```
10 20
30 -10 200
```

In [15]:

```
sno,sname,marks=10,"Rossum",34.56
print(sno,sname,marks)
```

```
10 Rossum 34.56
```

In [16]:

```
a,b=10,20
print(a,b)
```

```
10 20
```

In [17]:

```
a,b=b,a      # Swapping Logic
print(a,b)
```

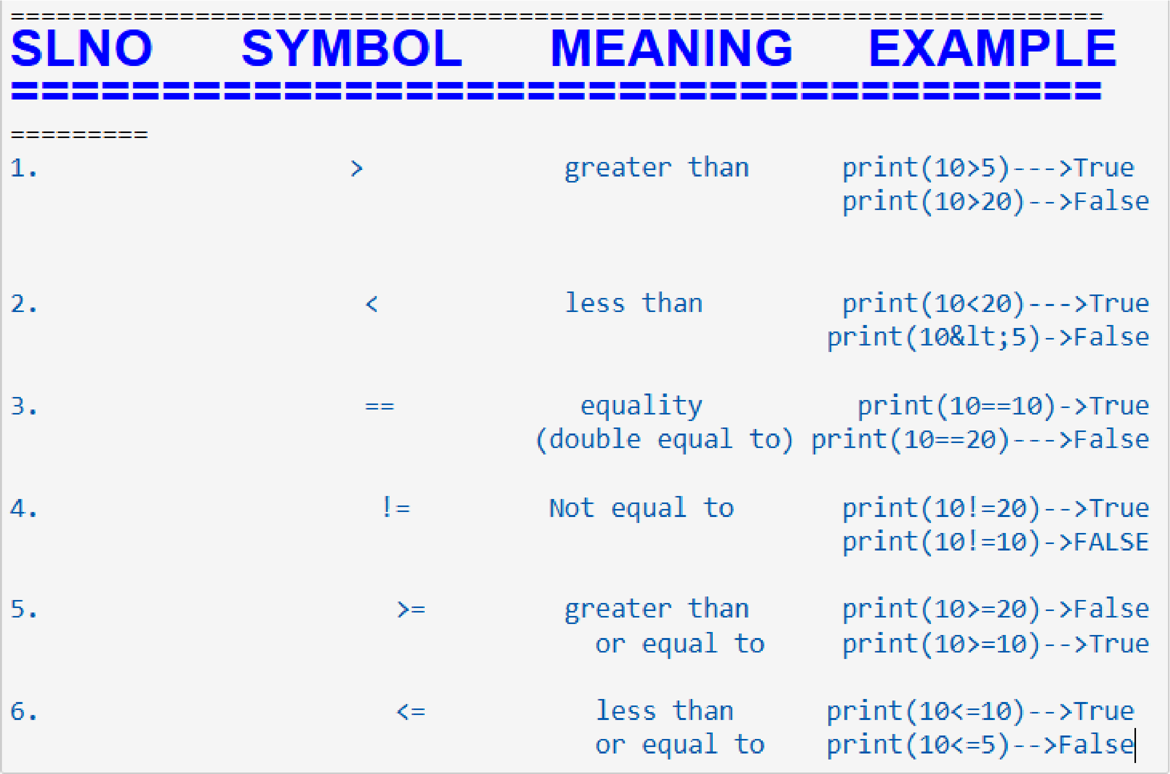
```
20 10
```

3. Relational Operators

- The purpose of Relational Operators is that "To Compare Two values."
- If Two or More Object / variables / Values Connected with Relational Operators then we call Relational Expression.
- The Result of Relational Expression is either True OR False
- The Relational Expression is also called Test Condition.
- In Python Programming, we have 6 Relational Operators. They are given in the following table.

```
1.greater than
2.less than
3.equality
4.Not equal to
```

```
5.greater than
6.less than
```



```
==================================================================
SLNO     SYMBOL     MEANING     EXAMPLE
==================================================================
=========
1.                      >            greater than      print(10>5)--->True
                                                       print(10>20)-->False


2.                       <           less than         print(10<20)--->True
                                                      print(10&lt;5)->False

3.                       ==           equality          print(10==10)->True
                                   (double equal to) print(10==20)--->False

4.                        !=         Not equal to      print(10!=20)-->True
                                                       print(10!=10)->FALSE

5.                         >=        greater than      print(10>=20)->False
                                       or equal to     print(10>=10)-->True

6.                         <=          less than      print(10<=10)-->True
                                       or equal to    print(10<=5)-->False
```

1. Greater than

In [18]:

```
a = 10
b = 5
c = a>b
print(c)
```

```
True
```

In [19]:

```
a = 10
b = 20
c = a>b
print(c)
```

```
False
```

2. less than

In [20]:

```
a = 10
b = 20
c = a<b
print(c)
```

```
True
```

In [21]:

```
a = 10
b = 5
c = a<b
print(c)
```

```
False
```

3.equality (double equal to)

In [1]:

```
a = 10
b = 10
c = a==b
print(c)
```

```
True
```

In [2]:

```
a = 10
b = 20
c = a==b
print(c)
```

```
False
```

4.Not equal to

In [3]:

```
a = 10
b = 20
c = a!=b
print(c)
```

```
True
```

In [5]:

```
a = 10
b = 10
c = a!=b
print(c)
```

```
False
```

5.greater than or equal to

In [6]:

```
a = 10
b = 20
c = a>=b
print(c)
```

```
False
```

In [7]:

```
a = 10
b = 10
c = a>=b
print(c)
```

```
True
```

6.lessthan or equal to

In [8]:

```
a = 10
b = 10
c = a<=b
print(c)
```

```
True
```

In [9]:

```
a = 10
b = 5
c = a<=b
print(c)
```

```
False
```

4. Logical Operators

- The purpose of Logical Operators is that "To combine two or More Relational Expressions".
- If Two Or More Relational Expressions are connected with Logical Operators then we call it as Logical Expression.
- The Result of Logical Expression is either True or False.
- The Logical Expression is also called Compund Test Condition.
- In Python programming, we have 3 types of Logical Operators. They are

```
1.and
2.or
3.not
```

1. and operator

- Syntax: RelExpr1 and RelExpr2
- The Functionality of "and" operator is expressed with Following Truth Table

RelExpr1	RelExpr2	RelExpr1 and RelExpr2
False	True	False
True	False	False
False	False	False
True	True	True

In [10]:

```
False and True
```

Out[10]:

```
False
```

In [11]:

```
True and False
```

Out[11]:

```
False
```

In [12]:

```
False and False
```

Out[12]:

```
False
```

In [13]:
```
True and True
```
Out[13]:
```
True
```

In [14]:
```
10>20 and 20>30  # Short Circuit Evaluation
```
Out[14]:
```
False
```

In [15]:
```
10>20 and 30>20 and 20>10   #Short Circuit Evaluation
```
Out[15]:
```
False
```

In [16]:
```
10<20 and 3>20 and 20>10    #Short Circuit Evaluation
```
Out[16]:
```
False
```

In [17]:
```
10>2 and 30>20 and 10>2   #Full length Evaluation
```
Out[17]:
```
True
```

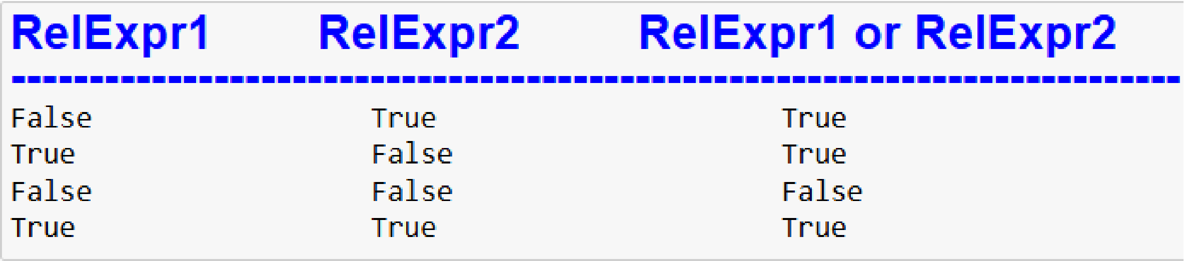
In [18]:
```
100>20 and 400>30 and 500>20   #Full length Evaluation
```
Out[18]:
```
True
```

Definition of Short Circuit Evaluation--in the case of "and" operator

- if an 'and' operator Connected with Multiple Relational Expressions and If Initial Relational Expression Evaluates to False then PVM will not Evaluate Rest of relational expressions and total result of Logical Expression is Considered as False. This Process of E valuation is called "Short Circuit Evaluation"

2. or operator

- Syntax: RelExpr1 or RelExpr2
- The Functionality of "or" operator is expressed with Following Truth Table



RelExpr1	RelExpr2	RelExpr1 or RelExpr2
False	True	True
True	False	True
False	False	False
True	True	True

False or True

In [20]:
```
True or False
```
Out[20]:
```
True
```

```
In [21]: False or False
Out[21]: False
```

```
In [22]: True or True
Out[22]: True
```

```
In [23]: 10>2 or 20>30 or 50>60    #Short Circuit Evaluation
Out[23]: True
```

```
In [24]: 10>20 or 30>20 or 50>30 or 50>60     #Short Circuit Evaluation
Out[24]: True
```

```
In [25]: 10>20 or 20>30 or 40>50   #Full length Evaluation
Out[25]: False
```

```
In [27]: 10>20 or 40>50 or 40>30     #Full length Evaluation
Out[27]: True
```

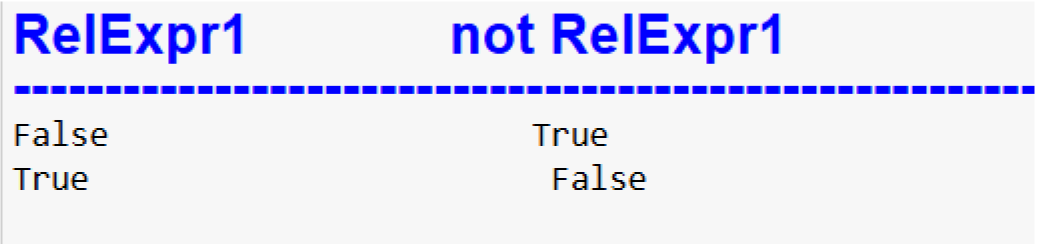
3. not operator

- Syntax: not Relational Expression

  (OR)

  not Logical Expression

- The Functionality of "not" operator is expressed with Following Truth Table



RelExpr1	not RelExpr1
False	True
True	False

```
In [28]: not True
Out[28]: False
```

```
In [29]: not False
Out[29]: True
```

```
In [30]: not True

Out[30]: False

In [31]: not False

Out[31]: True

In [32]: not 10

Out[32]: False

In [33]: not -10

Out[33]: False

In [34]: not 0

Out[34]: True

In [35]: not 10-10

Out[35]: True

In [36]: not "PYTHON"

Out[36]: False

In [37]: not""

Out[37]: True

In [38]: not "$"

Out[38]: False

In [39]: not "Python-python"

Out[39]: False

In [40]: not "10-10"

Out[40]: False

In [41]: 100 and 200

Out[41]: 200

In [42]: 100 and -120

Out[42]: -120
```

```
In [43]: 100 and 0

Out[43]: 0
```

```
In [48]: 0 and -123

Out[48]: 0
```

```
In [49]: 123-122 and 122-122

Out[49]: 0
```

```
In [47]: 123 and 345 and 100

Out[47]: 100
```

```
In [50]: "python" and "java" or "HTML"

Out[50]: 'java'
```

```
In [51]: "python" and False or "HTML"

Out[51]: 'HTML'
```

```
In [52]: "python" and False and "HTML"

Out[52]: False
```

```
In [53]: "python" and "False" and "HTML"

Out[53]: 'HTML'
```

```
In [54]: "#" and "$&" and "!" or 0

Out[54]: '!'
```

5. Bitwise Operators

- The purpose of Bitwise Operators is that "To perform the operations on Integer data in the form Bit by "Bit"
- Bitwise Operators are those which are applicable on Integer Data only But not on FloatingPnt Values bcoz Integer data provides Certainity where as floating point point data doeois not provide Certainity.
- The Execution Process of Bitwise Operators is that " Bitwise Operators First Coverts Integer data into Binary Format, Apply the Type of Bitwise Operator , get the result and Gives the Final Result data into Binary Format, Apply the Type of Bitwise Operator , get the result and Gives the Final Result in the form of Integer Data (Decimal Number System)"

- Since the Bitwise Operators Perform Operations on the basis of Bit by Bit and hence named as Bitwise Operators.
- In Python Programming, we have 6 types of Bitwise Operators. They are

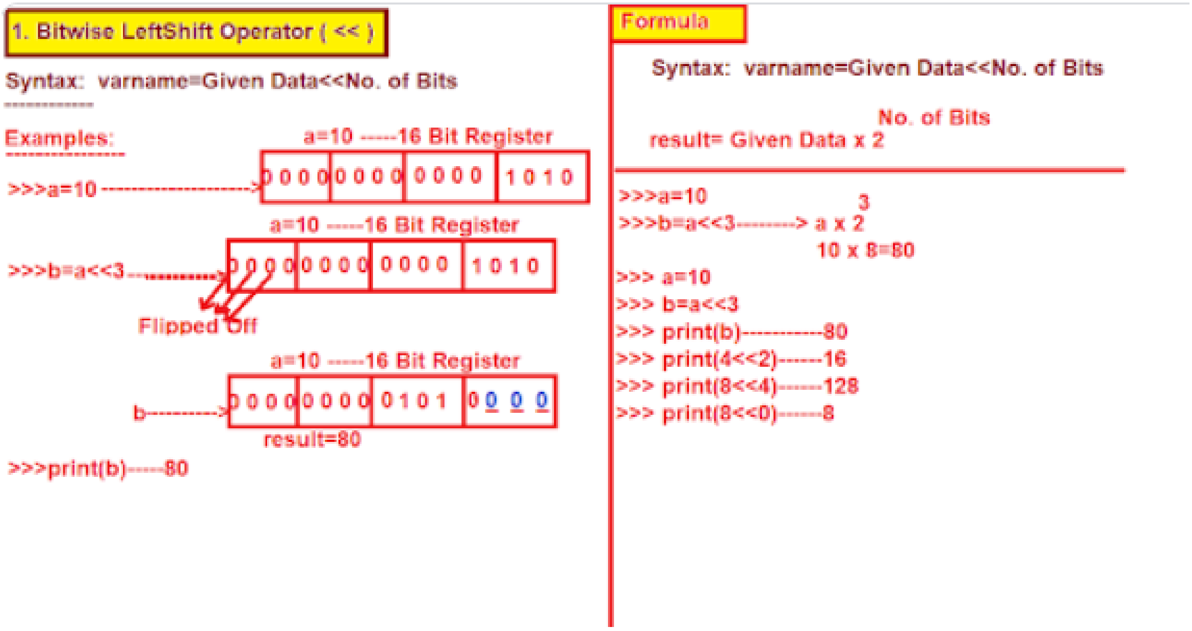
```
1. Bitwise LeftShift Operator ( << )
2. Bitwise Right Operator ( >> )
3. Bitwise AND Operator ( & )
4. Bitwise OR Operator ( | )
5. Bitwise Complement Operator ( ~ Tilde )
6. Bitwise XOR Operator ( ^ )
```

1. Bitwise LeftShift Operator (<<)

- Syntax: varname=Given Data << No. of Bits

Explanation:

The Execution Process of Bitwise LeftShift Operator (<<) is that "It Moves Number of Bits Towards Left Side By Adding Number of Zeros (Number of Zeros=Depending No. Of bits we Flipped-off) at Right Side.


1. Bitwise LeftShift Operator (<<)
Syntax: varname=Given Data<<No. of Bits
Examples:
a=10 -----16 Bit Register
>>>a=10 --------------------->
0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 1 0 1 0
a=10 -----16 Bit Register
>>>b=a<<3---------
0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 1 0 1 0
Flipped Off
a=10 -----16 Bit Register
b----------->
0 0 0 0 | 0 0 0 0 | 0 1 0 1 | 0 0 0 0
result=80
>>>print(b)-----80
Formula
Syntax: varname=Given Data<<No. of Bits
result= Given Data x 2^No. of Bits
>>>a=10
>>>b=a<<3--------> a x 2^3
10 x 8=80
>>> a=10
>>> b=a<<3
>>> print(b)-----------80
>>> print(4<<2)-------16
>>> print(8<<4)-------128
>>> print(8<<0)-------8

In [56]:
```
a =10
```

In [57]:
```
b = a<<3
```

In [58]:
```
print(b)
```

```
80
```

In [59]:
```
print(4<<2)
```

```
16
```

In [61]:
```
print(4<<2)
```

```
16
```

In [62]:
```
 print(8<<0)
```

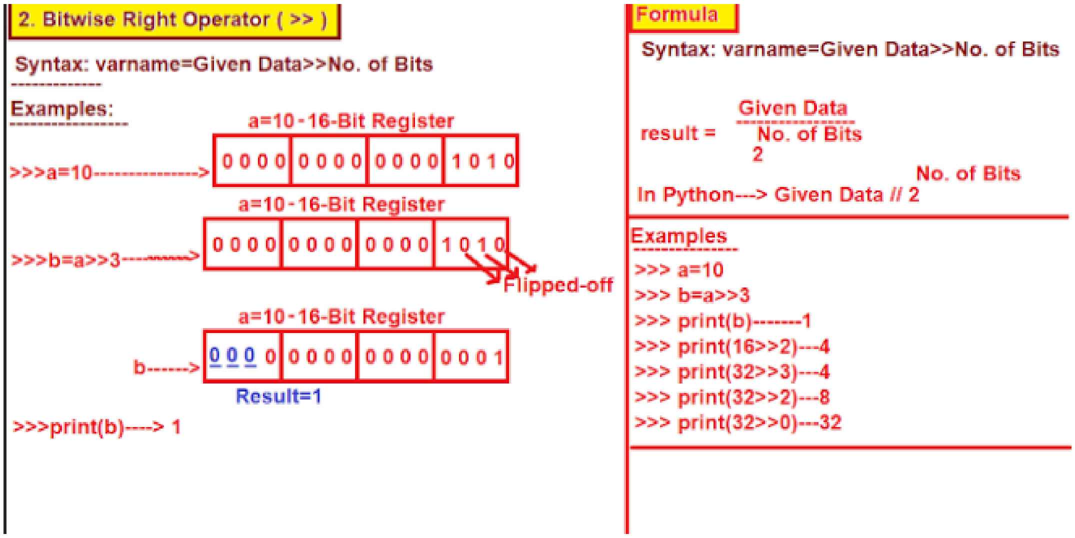
```
8
```

2. Bitwise RightShift Operator (>>)

- Syntax: varname=Given Data >> No. of Bits

Explanation:

The Execution Process of Bitwise Right Shift Operator ( ) is that "It Moves Number of Bitsowards Right Side By Adding Number of Zeros (Number of Zeros=Depending No. Of bits we Flipped-off) at Left Side.



2. Bitwise Right Operator (>>)

Syntax: varname=Given Data>>No. of Bits

Examples:

```
>>>a=10
>>>b=a>>3
b
>>>print(b)----> 1
```

Formula

Syntax: varname=Given Data>>No. of Bits

$$result = \frac{Given\ Data}{2^{No.\ of\ Bits}}$$

In Python---> Given Data // $2^{No.\ of\ Bits}$

Examples

```
>>> a=10
>>> b=a>>3
>>> print(b)-------1
>>> print(16>>2)---4
>>> print(32>>3)---4
>>> print(32>>2)---8
>>> print(32>>0)---32
```

In [63]:
```
a=10
b=a<<3
```

In [64]:
```
print(b)
```

```
80
```

In [65]:
```
print(16>>2)
```

```
4
```

In [67]:
```
print(32>>3)
```

```
4
```

In [68]:
```
print(32>>2)
```

```
8
```

In [69]:
```
print(32>>0)
```

```
32
```

3. Bitwise AND Operator (&)

- Syntax: varname = Value1 & Value2
- The Functionality of Bitwise AND Operator (&) is Expressed by using the Following

```
==========================================
Value1          Value2       Value1 & Value2
==========================================
0                  1              0
1                  0              0
0                  0              0
1                  1              1
==========================================
```

```
In [70]: 1 & 0
Out[70]: 0
```

```
In [71]: 0 & 1
Out[71]: 0
```

```
In [72]: 0 & 0
Out[72]: 0
```

```
In [73]: 1 & 1
Out[73]: 1
```

```
In [74]: a = 10
```

```
In [75]: s1={10,20,30}
```

```
In [76]: s2={15,20,35}
```

```
In [77]: s3=s1.intersection(s2)
```

```
In [78]: print(s3,type(s3))
         {20} <class 'set'>
```

```
In [79]: s1={10,20,30}
```

```
In [80]: s2={15,20,35}
```

```
In [81]: s3 =  s1 & s2   # Biwise AND ( & ) Operator
```

In [82]:

```
print(s3,type(s3))
```

```
{20} <class 'set'>
```

In [83]:

```
s1={"Apple","Mango","Kiwi"}
```

In [84]:

```
s2={"Mango","Kiwi","Guava"}
```

In [85]:

```
s3=s1 & s2    # Biwise AND ( &amp; ) Operator
```

4. Bitwise OR Operator (|)

- Syntax: varname = Value1 | Value2
- The Functionality of Bitwise OR Operator (|) is Expressed by using the Following Truth table.

Value1	Value2	Value1 | Value2
0	1	1
1	0	1
0	0	0
1	1	1

In [86]:

```
 0 | 1
```

Out[86]:

```
1
```

In [1]:

```
1 | 0
```

Out[1]:

```
1
```

In [88]:

```
0 | 0
```

Out[88]:

```
0
```

In [89]:

```
1 | 1
```

Out[89]:

```
1
```

In [90]:

```
a=10
b=15
c=a|b
print(c)
```

```
15
```

In [91]:

```
print(4|4)
```

```
4
```

In [92]:
```
print(4|15)
```
```
15
```

In [93]:
```
s1={10,20,30}
```

In [94]:
```
s2={30,40,50}
```

In [95]:
```
s3=s1.union(s2)
```

In [96]:
```
print(s3,type(s3))
```
```
{50, 20, 40, 10, 30} <class 'set'>
```

In [97]:
```
s1={10,20,30}
```

In [98]:
```
s2={30,40,50}
```

In [99]:
```
s3=s1|s2 # Bitwise OR Operator
```

In [100...
```
print(s3,type(s3))
```
```
{50, 20, 40, 10, 30} <class 'set'>
```

In [101...
```
s1={"apple","mango","kiwi"}
```

In [102...
```
s2={"sberry","mango","guava"}
```

In [103...
```
s3=s1|s2 # Bitwise OR Operator
```

In [104...
```
print(s3,type(s3))
```
```
{'kiwi', 'sberry', 'mango', 'guava', 'apple'} <class 'set'>
```

5. Bitwise Complement Operator(~ Tilde)

- Bitwise Complement Operator(~) is used for Complementing the Given Integer Data.
- Bitwise Complement Operator(~) Internally It will Invert the Bits and Becomes the Result of
- Bitwise Complement Operator(~).
- Inverting the bits is nothing But 1 becomes 0 and 0 becomes 1.
- The Formula for Bitwise Complement Operator= - (Value+1)

In [106...
```
a=10
```

In [107...
```
~a
```

Out[107]:
```
-11
```

- Let a and whose Binary = 1010
- =Bitwise Complement of a= 0101 (Inverting the Bits)

Proof: How ~10 becomes -11

- Let Given Number : 11
- Binary Format of 11 = 1011
- 1's Complement of 11=0100
- 2's Complement of 11= 1 's Complement of 11 +1

```
= 0100+1
 =0100
 +0001
 ----------
0101---which is 2's Complement of 11
```

In [108…

```
a = 16
```

In [109…

```
~a
```

Out[109]:

```
-17
```

- Let a and whose Binary = 1 0000
- Bitwise Complement of a= 0 1111 (Inverting the Bits)

Proof: How ~16 becomes -17

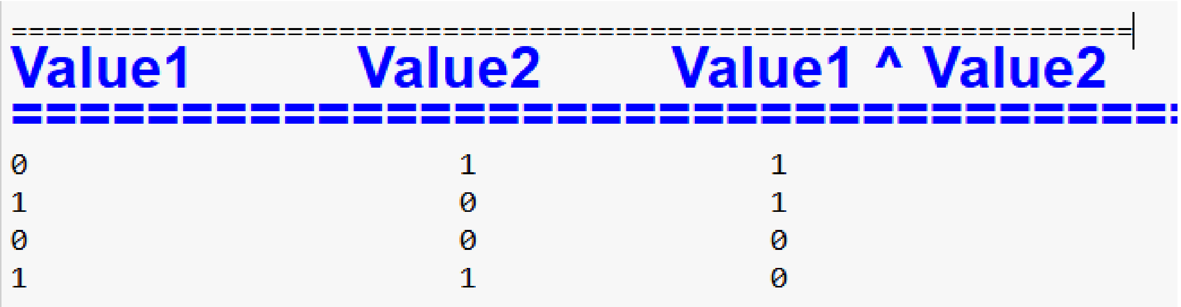
- Let Given Number : 17
- Binary Format of 17 = 1 0001
- 1's Complement of 17=0 1110
- 2's Complement of 17= 1 's Complement of 17 +1

```
=0 1110+1
=0 1110
+0 0001

 0 1111---which is 2's Complement of 17
```

6. Bitwise XOR Operator (^)

- Syntax: varname = Value1 ^ Value2
- The Functionality of Bitwise XOR Operator (^) is Expressed by using the Following Truth table.



```
=================================================================
Value1          Value2          Value1 ^ Value2
=================================================================
0                        1                 1
1                        0                 1
0                        0                 0
1                        1                 0
```

In [110...
```
0 ^ 1
```
Out[110]:
```
1
```

In [111...
```
1 ^ 0
```
Out[111]:
```
1
```

In [112...
```
 0 ^ 0
```
Out[112]:
```
0
```

In [113...
```
1 ^ 1
```
Out[113]:
```
0
```

In [114...
```
a=2
b=3
```

In [115...
```
c=a^b
print(c)
```
```
1
```

In [116...
```
print(15^10)
```
```
5
```

In [117...
```
print(10^15)
```
```
5
```

In [118...
```
print(7^4)
```
```
3
```

In [119...
```
s1={10,20,30}
```

In [120...
```
s2={30,40,50}
```

In [121...
```
s3=s1.symmetric_difference(s2)
```

In [122...
```
print(s3,type(s3))
```
```
{40, 10, 50, 20} <class 'set'>
```

In [123...

```
s1={10,20,30}
```

In [124...

```
s2={30,40,50}
```

In [125...

```
s3=s1^s2 # Bitwise XOR Operator ( ^ )
```

In [126...

```
print(s3,type(s3))
```

```
{40, 10, 50, 20} <class 'set'>
```

6. Membership Operators--Most Imp

- The purpose of Membership Operators is that "To check the whether the Specified Value Present in terable object or not"
- An Iterable object is one which contains More than One Value (str,bytes,bytearray,range,list,tuple,set,frozenset,dict). where as a Non-An Iterable object is one which contains Only One Value.
- In Python Programming, we have Two Types of Membership Operators. They are

```
1. in
2. not in
```

1. In

- Syntax: Value in Iterable-Object
- The "in" Operator Returns True provided "Value" Present iIterable-Object
- The "in" Operator Returns False provided "Value" not Present iIterable-Object

not in

- Syntax: Value not in Iterable-Object
- The "not in" Operator Returns True provided "Value" Not Present iIterable-Object.
- The "not in" Operator Returns False provided "Value" Present iIterable-Object.

In [127...

```
s="PYTHON"
```

In [129...

```
"P" in s
```

Out[129]:

```
True
```

In [130...

```
"p" in s
```

Out[130]:

```
False
```

```
In [131…  "p " not in s

Out[131]: True

In [132…  "P" not in s

Out[132]: False
```

7. Identity Operators

- The purpose of Identity Operators is that "To Compare the memory address of Two Objects".
- In Python Programming, we have Two Types of Identity Operators. They are

```
1. is
2. is not
```

1. is

- Syntax: Object1 is Object2
- The "is" Operator Returns True provided Both Object1 and Object2 Memory Address Must be same
- The "is" Operator Returns False provided Both Object1 and Object2 Memory Addresses are Different.

2. is not

- Syntax: Object1 is not Object2
- The "is not" Operator Returns True provided Both Object1 and Object2 Memory Addresses are different
- The "is not" Operator Returns False provided Both Object1 and Object2 Memory Addresses

```
In [134…  a=200
          b=a # Deep Copy

In [135…  print(a,id(a))

          200 1981736051344

In [136…  print(b,id(b))

          200 1981736051344

In [137…   a is b

Out[137]: True
```

```
In [138…  a is not b

Out[138]: False

In [139…  l1=[10,"RS"]

In [140…  l2=l1.copy() # Shallow Copy

In [141…  print(l1,id(l1))

          [10, 'RS'] 1981851060992

In [142…  print(l2,id(l2))

          [10, 'RS'] 1981851149568

In [143…  l1 is l2

Out[143]: False

In [144…  l1 is not l2

Out[144]: True

In [145…  a=None
          b=None

In [146…  print(a,id(a))

          None 140735247916248

In [147…  print(b,id(b))

          None 140735247916248

In [148…  a is b

Out[148]: True

In [149…  a is not b

Out[149]: False

In [150…  d1={10:"Apple",20:"Mango"}

In [151…  d2={10:"Apple",20:"Mango"}

In [152…  print(d1,id(d1))

          {10: 'Apple', 20: 'Mango'} 1981851072704

In [153…  print(d2,id(d2))

          {10: 'Apple', 20: 'Mango'} 1981851073792

In [154…  d1 is d2
```

Out[154]:

```
False
```

In [155…

```
d1 is not d2
```

Out[155]:

```
True
```

In [156…

```
a,b=2+3.5j,2+3.5j
```

In [157…

```
print(a,id(a))
```

```
(2+3.5j) 1981851283920
```

In [158…

```
print(b,id(b))
```

```
(2+3.5j) 1981851283920
```

In [159…

```
a is b
```

Out[159]:

```
True
```

In [160…

```
a is not b
```

Out[160]:

```
False
```

In [ ]: