```
True
2.3 >= 1.9
True
False >= 0.0
True
False >= 0.000000008
False
'great'>='great'
True
{1,2,3}>{1,2,3}
False
{1,2,3}>={1,2,3}
True
[10,20,30]>=[30,10,20]
False
(1,2,7)>=(1,2,2*4-True)
True
```

- **Lesser than or Equal to (<=):** It will not support for complex and dictionary.
  --- Used to check whether the operand1 is lesser than or equal to operand2 or not.

  For SVDT - It compares values,
  For string, ASCII values will be compared.
  For Set : All the values of operand1 should be present in operand2, Then it will give True or it will give False.

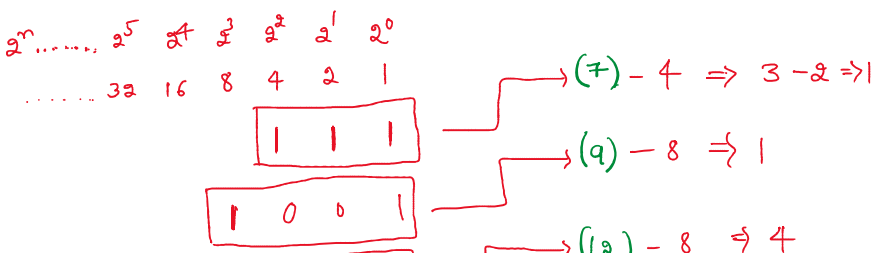
  Proof:

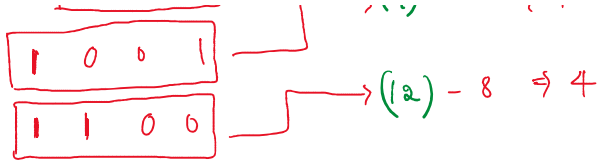
```
0.0 <= False
True
1 <=  4
True
True <= False
False
'python' <='java'
False
[5,4,3,2]<=[5,4,True+2,10%4]
True
(1,2,3) <= (False**0,True**2+1,123%10)
True
{3,'hello',8,5}<={24,56,'hello',5,1+2,8,True}
True
```

Day-8

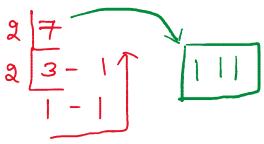
4) **<u>Bitwise Operator</u> : -- it is applicable only for integer**
   --- It will consider binary values perform operation bit by bit .

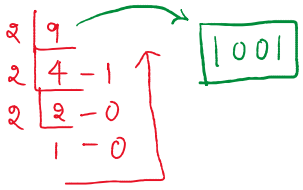
   We can get binary values using 3 ways,
   - **Binary scale method.**

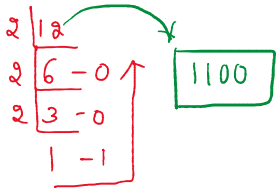




- **Divide by 2 method.**



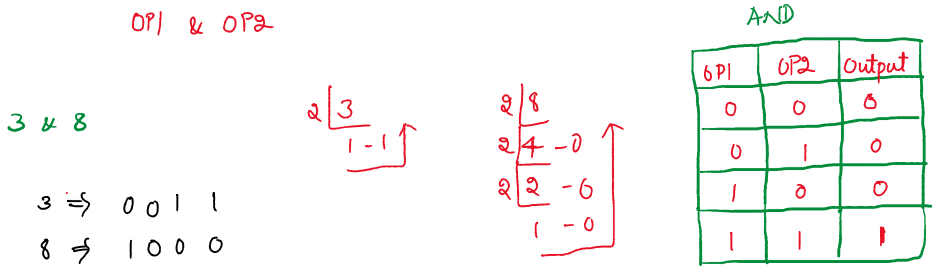


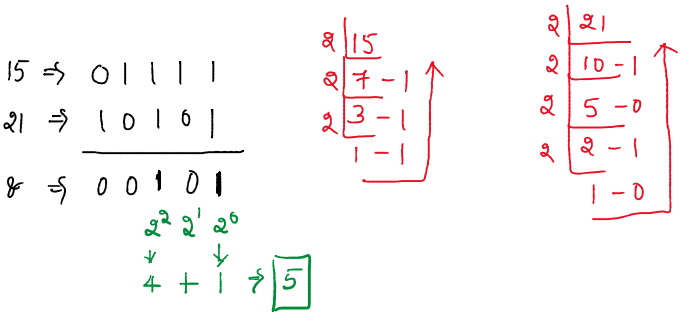


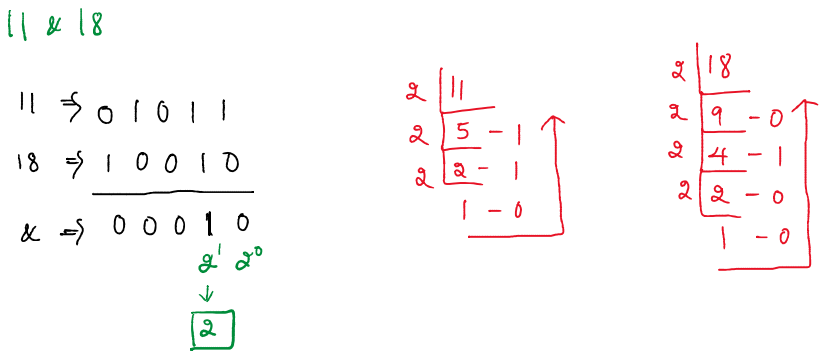
- **Using bin() function.**

```
bin(7)
'0b111'
bin(9)
'0b1001'
bin(12)
'0b1100'
```

- **Bitwise AND(&):**







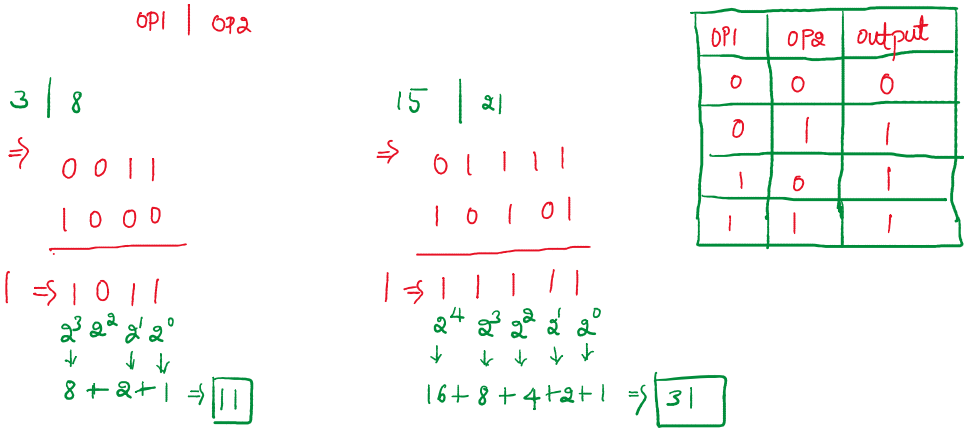
Proof:

3 & 8
0
15 & 21
5
11 & 18
2

Assignment : Get the bitwise and values of

- 17 & 19
- 32 & 27
- 14 & 21
- 13 & 24

- **Bitwise OR(|):**

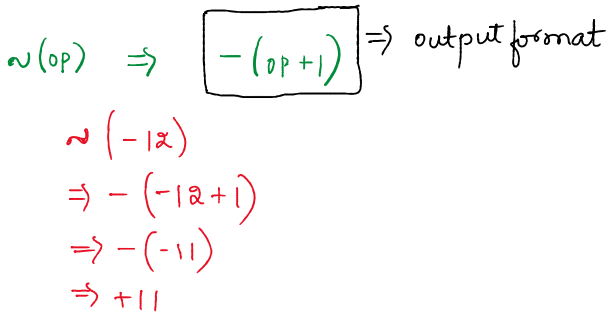


Program:

3 | 8
11
15 | 21
31

Assignment : Take the rest of the previous bitwise and questions and do it for bitwise or.

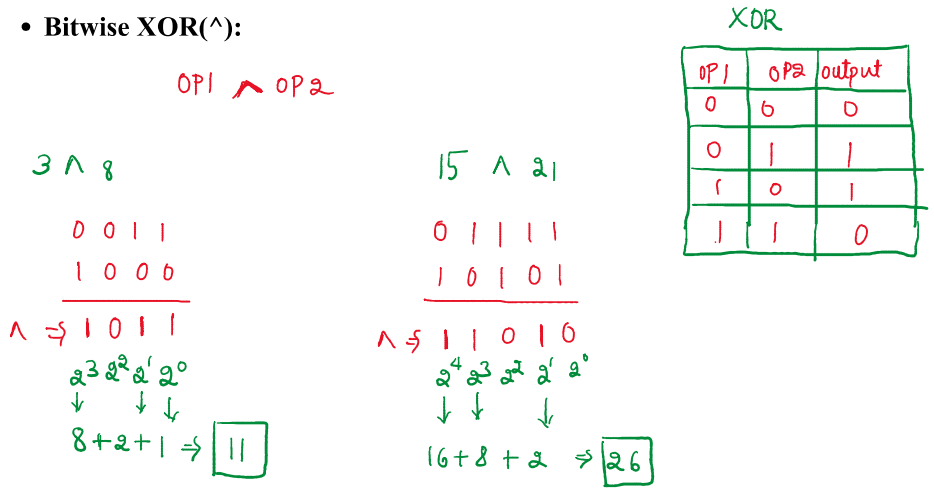
- **Bitwise NOT(~):**



Proof:

~3
-4
~-12
11

- **Bitwise XOR(^):**

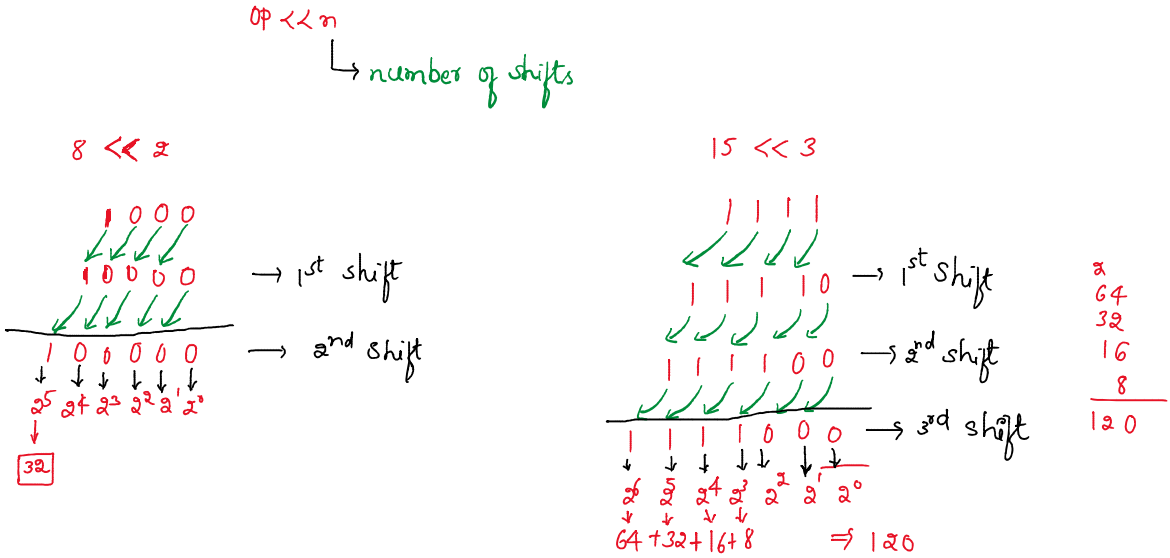


Program:

```
3 ^ 8
11
15 ^ 21
26
```

Assignment : Take the rest of the previous bitwise and questions and do it for bitwise XOR.

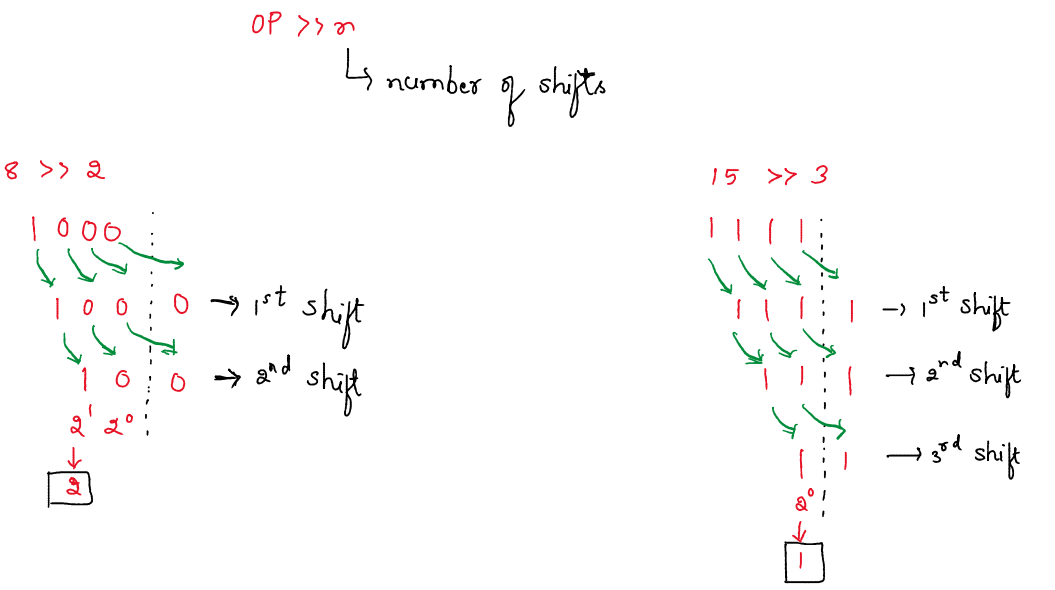
- **Bitwise left shift operator(<<):**



Proof:

```
8 << 2
32
15 << 3
120
```

- **Bitwise right shift operator(>>):**



Proof:

```
8>>2
2
15 >> 3
1
```

15 << 2

1 1 1 1 0 0

15 >>2

1 1 (1 1)

5) **Assignment Operator(=):**
---- It is used to assign the value to a variable.

a=a+b ----- a+=b
a=a-b ----- a-=b
a=a*b ----- a*=b
a=a/b ----- a/=b
a=a//b ----- a//=b
a=a%b ----- a%=b
a=ab ----- a**=b**
a=a&b ----- a&=b
a=a|b ----- a|=b
a=a^b ----- a^=b
a=a<<b ----- a<<=b
a=a>>b ----- a>>=b

Proof:

```
a = 10
a = a + 3
a
13
a = a - 3
a
10
a += 3
a
13
a -= 3
a
10
```

7) **Membership operator:**
--- It is used to check if the value present inside the collection or not.

- **in** --- values present in collection. It returns True if the value is present inside the collection, else return False.

```
'hi' in  'hi hello'
True
10 in {'a':10,'b':20}
False
[1] in [1,2,3]
False
1 in [1,2,3]
True
[1] in [[1],2,3]
True
True in {1,2,3,4}
True
(1) in (1,2,3)
True
(1)
1
(1,) in (1,2,3)
False
```

- **not in ---** values not present in collection. It returns True if the value is not present inside the collection, else return False.

```
45 not in [12,34,56]
True
p not in 'python'
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    p not in 'python'
NameError: name 'p' is not defined
'p' not in 'python'
False
'hl' not in 'hello'
True
'' not in 'python'
False
[] not in [10,20]
True
() not in (10,20)
True
[1] not in [1,2,3]
True
```

8) **Identity Operator:**
--- It is used to check whether both the operands are sharing the same address or not.

Types:

- is --- It will return True if both the operands are sharing the same address or else return False.

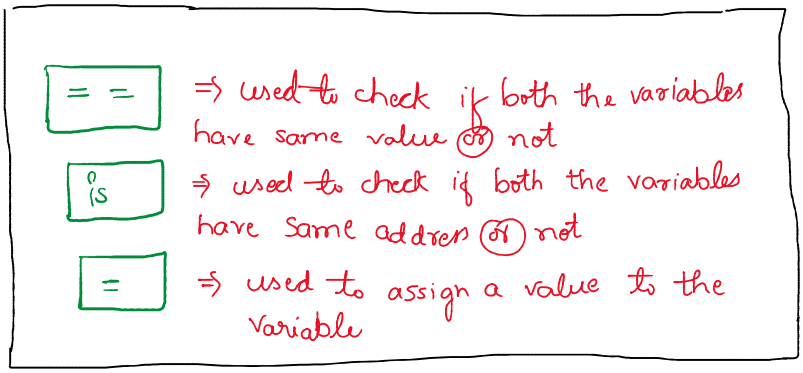
Syntax: OP1 is OP2

- is not --- It will return True if both the operands are not sharing the same address or else return False.

  Syntax: OP1 is not OP2

  Proof:

```
a=10
b=20
c=10
a is b
False
a is c
True
b is c
False
a is not b
True
a is not c
False
b is not c
True
```



↳ Important Questions on Operators

Day-9

Input & Output Statements:

- **Input Statements:**
  --- We should never allow the user to modify the code instead just allow them to access the code.

  Syntax:

```
var = input(' message ')
a = input('Enter the value: ')
print(type(a))
```

Note: input() function will take input by default in the form of string.

If we want other datatype values like int, float, complex then we have to use type casting.

```
a = dest_type(input('message'))
a = int(input('Enter the value: '))
print(type(a))


a = list(input('Enter the value: '))
print(a)
print(type(a))
'''
Enter the value: sakshi
['s', 'a', 'k', 's', 'h', 'i']
<class 'list'>'''
```