Arithmetic operator

In [ ]:
```
x1, y1 = 10, 5
```

In [ ]:
```
#x1 ^ y1
```

In [ ]:
```
x1  + y1
```

In [ ]:
```
x1 - y1
```

In [ ]:
```
x1 * y1
```

In [ ]:
```
x1 / y1 # float division
```

In [ ]:
```
x1 // y1 #int division
```

In [ ]:
```
x1 % y1
```

In [ ]:
```
x1 ** y1
```

In [ ]:
```
3 ** 2
```

In [ ]:
```
x2 = 3
y2 = 3

x2 ** y2
```

Assignment operator

In [ ]:
```
x = 2
```

In [ ]:
```
x = x + 2 # if you want to increment by 2
```

In [ ]:
```
x
```

In [ ]:
```
x += 2
x
```

In [ ]:
```
x += 2
x
```

In [ ]:
```
x *= 2
```

In [ ]:
```
x
```

In [ ]:
```
x -= 2
```

In [ ]:
```
x
```

In [ ]:
```
x /= 2
x
```

In [ ]:
```
x //= 2
x
```

In [ ]:
```
a, b = 5,6 # you can assigned variable in one line as well
print(a)
print(b)
```

In [ ]:
```
a = 5
b = 6
print(a)
print(b)
```

In [ ]:
```
a
```

In [ ]:
```
b
```

unary operator

- unary means 1 || binary means 2
- Here we are applying unary minus operator(-) on the operand n; the value of m becomes -7, which indicates it as a negative value.

In [ ]:
```
n = 7 #negattion
n
```

In [ ]:
```
m = -(n)
m
```

In [ ]:
```
n
```

In [ ]:
```
-n
```

Relational operator

we are using this operator for comparing

In [ ]:
```
a = 5
b = 6
```

In [ ]:
```
a<b
```

In [ ]:
```
a>b
```

In [ ]:
```
# a = b # we cannot use = operatro that means it is assigning
```

In [ ]:
```
a == b
```

In [ ]:
```
a != b
```

In [ ]:
```
# hear if i change b = 6
b = 5
```

In [ ]:
```
a == b
```

In [ ]:
```
a
```

In [ ]:
```
b
```

In [ ]:
```
a > b
```

In [ ]:
```
a >= b
```

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

In [ ]:
```
a < b
```

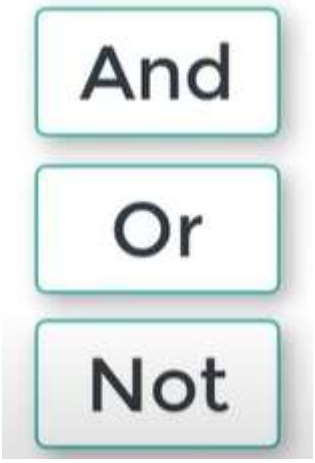
In [ ]:
```
a>b
```

In [ ]:
```
b = 7
```

In [ ]:
```
a != b
```

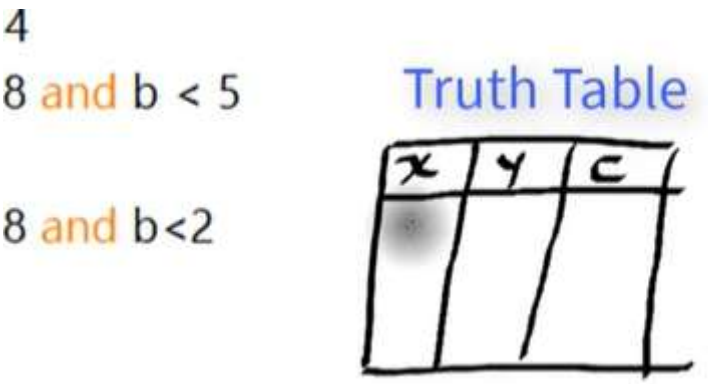
LOGICAL OPERATOR

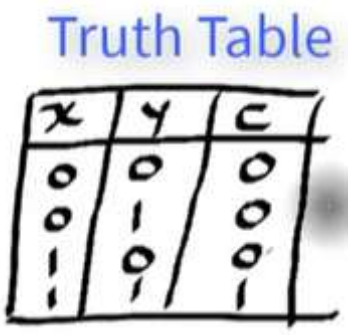
- logical operator you need to understand about true & false table

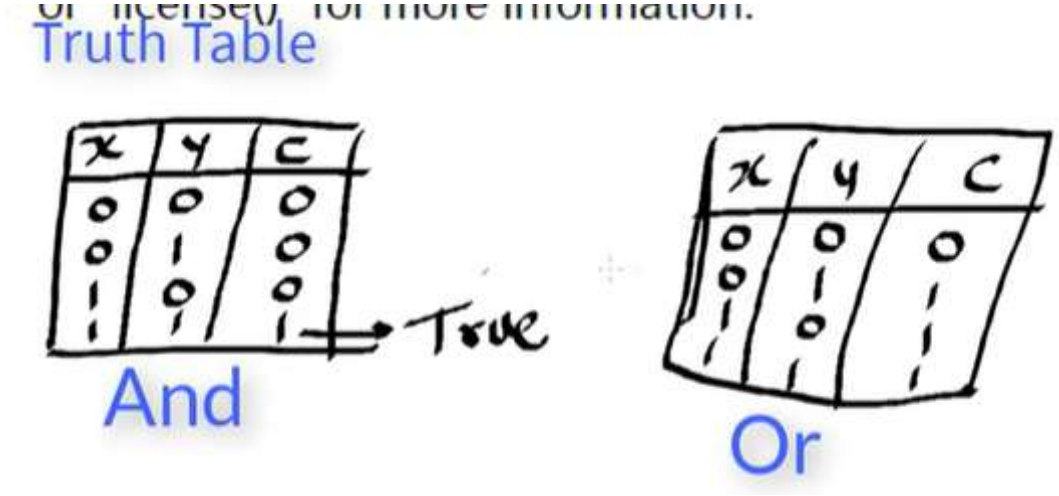


- 3 importand part of logical operator is --> AND, OR, NOT

- lets understand the truth table:- in truth table you can represent (true-1 & false means- 0)







In [ ]:
```
a = 5
b = 4
```

In [ ]:
```
a < 8 and b < 5 #refer to the truth table
```

In [ ]:
```
a < 8 and b < 2
```

In [ ]:
```
a < 8 or b < 2
```

In [ ]:
```
a>8 or b<2
```

In [ ]:
```
x = False
x
```

In [ ]:
```
not x  # you can reverse the operation
```

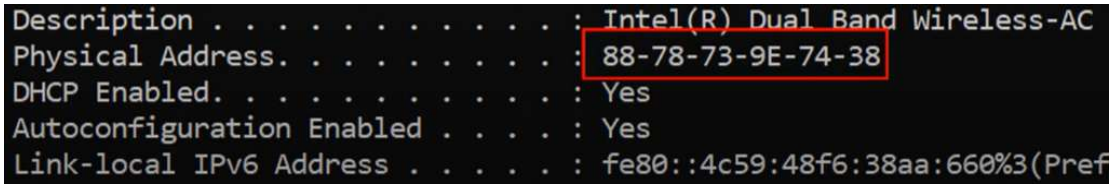
In [ ]:
```
x = not x
x
```

In [ ]:
```
x
```

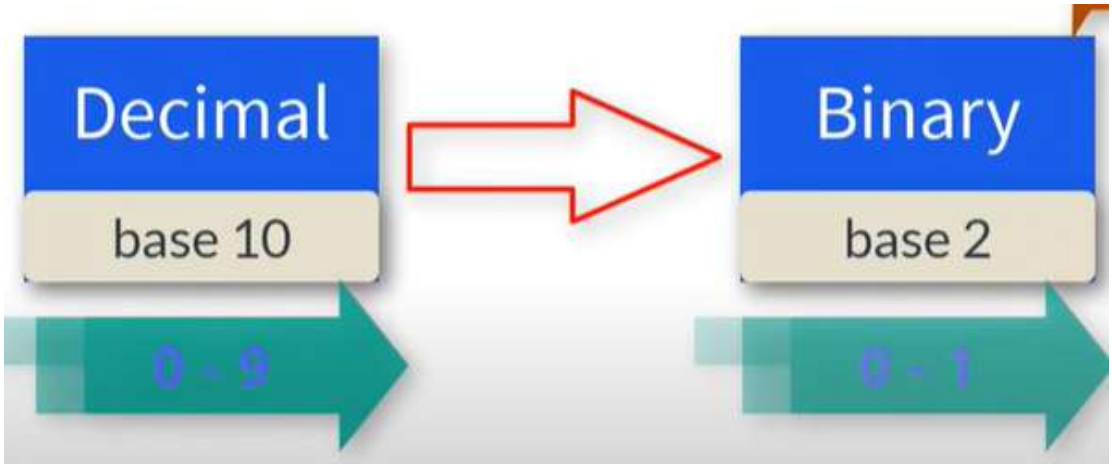
In [ ]:
```
not x
```

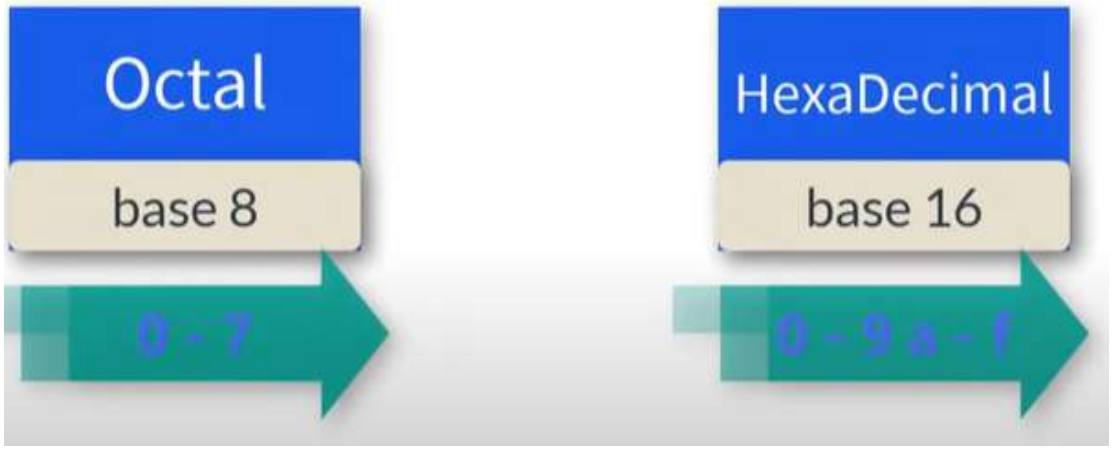
Number system coverstion (bit-binary digit)

- In the programing we are using binary system, octal system, decimal system & hexadecimal system
- but where do we use this in cmd - you can check your ip address & lets understand how to convert from one system to other system
- when you check ipaddress you will these format --> cmd - ipconfig



binary : base (0-1) --> please divide 15/2 & count in reverse order ocatl : base (0-7)
hexadecimal : base (0-9 & then a-f)





In [ ]:
```
25
```

In [ ]:
```
bin(25)
```

In [ ]:
```
int(0b11001)
```

In [ ]:
```
bin(30)
```

In [ ]:
```
int(0b11110)
```

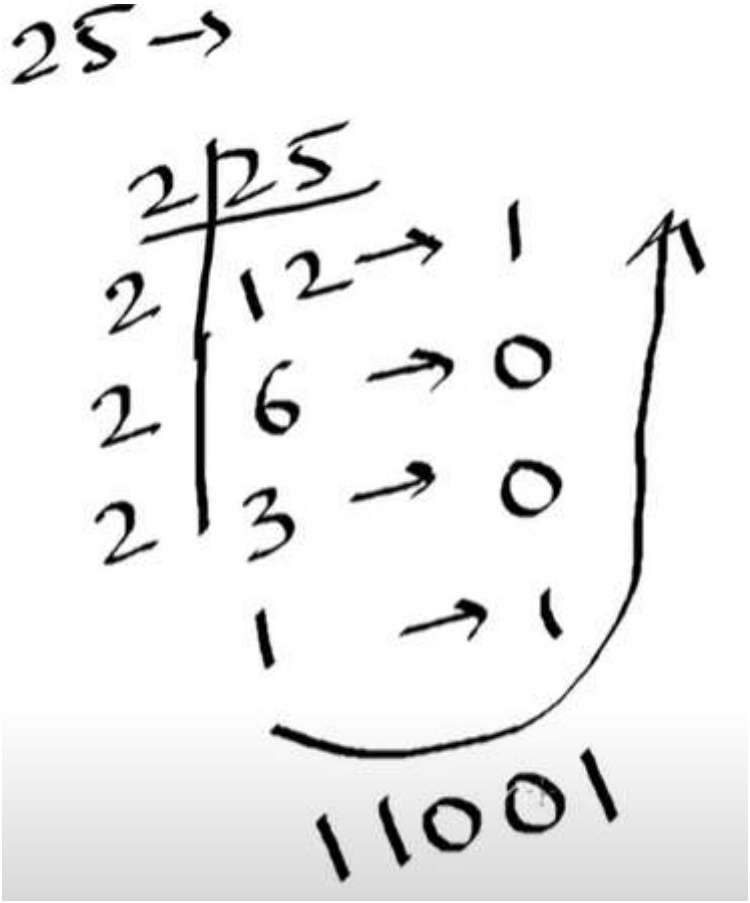
In [ ]:
```
int(0b11001)
```

In [ ]:
```
oct(25)
```

In [ ]:
```
int(0o31)
```

In [ ]:
```
int(0b11110)
```

In [ ]:
```
0o31
```



In [ ]:
```
0b11001
```

In [ ]:
```
int(0b11001)
```

In [ ]:
```
bin(7)
```

In [ ]:
```
oct(25)
```

In [ ]:
```
0o31
```

In [ ]:
```
int(0o31)
```

In [ ]:
```
hex(25)
```

In [ ]:
```
0x19
```

In [ ]:
```
hex(16)
```

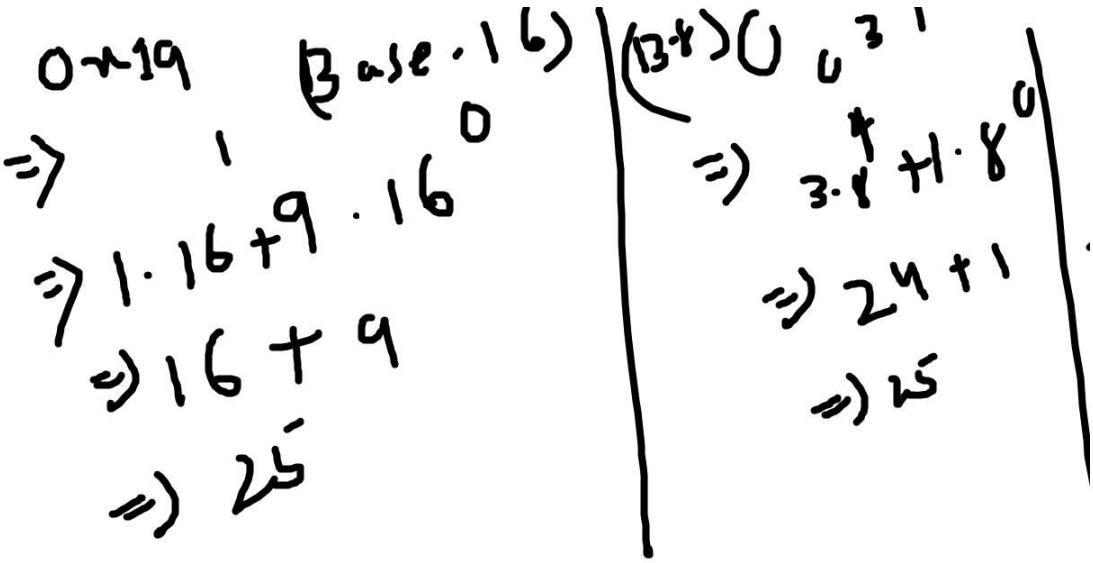
In [ ]:
```
0xa
```

In [ ]:
```
0xb
```

In [ ]:
```
hex(1)
```

```
>>> hex(1)
'0x1'
>>> hex(2)
'0x2'
>>> hex(8)
'0x8'
>>> hex(10)
'0xa'
>>> hex(11)
'0xb'
>>> hex(256)
'0x100'
```

In [ ]:
```
hex(25)
```



In [ ]:
```
0x19
```

In [ ]:
```
0x15
```

swap 2-variable in python

(a,b = 5,6) After swap we should get ==> (a, b = 6,5)

In [ ]:
```
a = 5
b = 6
```

In [ ]:
```
a = b
b = a
```

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

In [ ]:
```
# in above scenario we lost the value 5
a1 = 7
b1 = 8
```

In [ ]:
```
temp = a1
a1 = b1
b1 = temp
```

In [ ]:
```
print(a1)
print(b1)
```

- in the above code we are using third variable
- in interview they might ask can we swap better way without using 3rd variable

In [ ]:
```
a2 = 5
b2 = 6
```

In [ ]:
```
#swap variable formulas without using 3rd formul
a2 = a2 + b2 # 5+6 = 11
b2 = a2 - b2 # 11-6 = 5
a2 = a2 - b2 # 11-5 = 6
```

In [ ]:
```
print(a2)
print(b2)
```

In [ ]:
```
0b110
```

In [ ]:
```
0b101
```

In [ ]:
```
print(0b110)
print(0b101)
```

In [ ]:
```
print(0b101)
print(0b110)
```

In [ ]:
```
#but when we use a2 + b2 then we get 11 that means we will get 4 bit which is 1
print(bin(11))
```

```
print(0b1011)
```

-there is other way to work using swap variable also which is XOR because it will not waste extra bit

XOR Basics

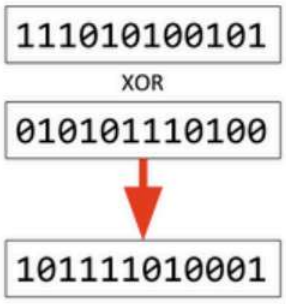
An XOR or *eXclusive OR* is a bitwise operation indicated by ^ and shown by the

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Encryption: XOR

Take data represented in binary and perform an operation against another set of bits where you get a 1 only if exactly one of the bits is 1

First Bit	Second Bit	Resulting Bit
0	0	0
0	1	1
1	0	1
1	1	0



In [ ]:
```
print(a2)
print(b2)
```

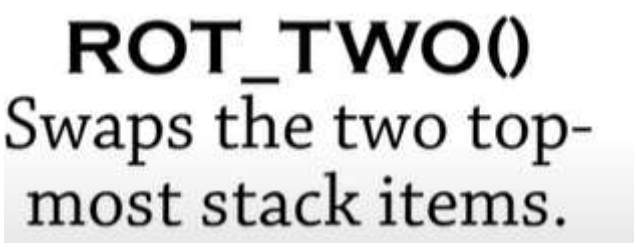
In [ ]:
```
#there is other way to work using swap variable also which is XOR because it wil
a2 = a2 ^ b2
b2 = a2 ^ b2
a2 = a2 ^ b2
```

In [ ]:
```
print(a2)
print(b2)
```

In [ ]:
```
a2, b2
```

In [ ]:
```
a2 , b2 = b2, a2 # how it work is b2 6 a2 is 5 first it goes into stack & then i
```

In [ ]:
```
print(a2)
print(b2)
```

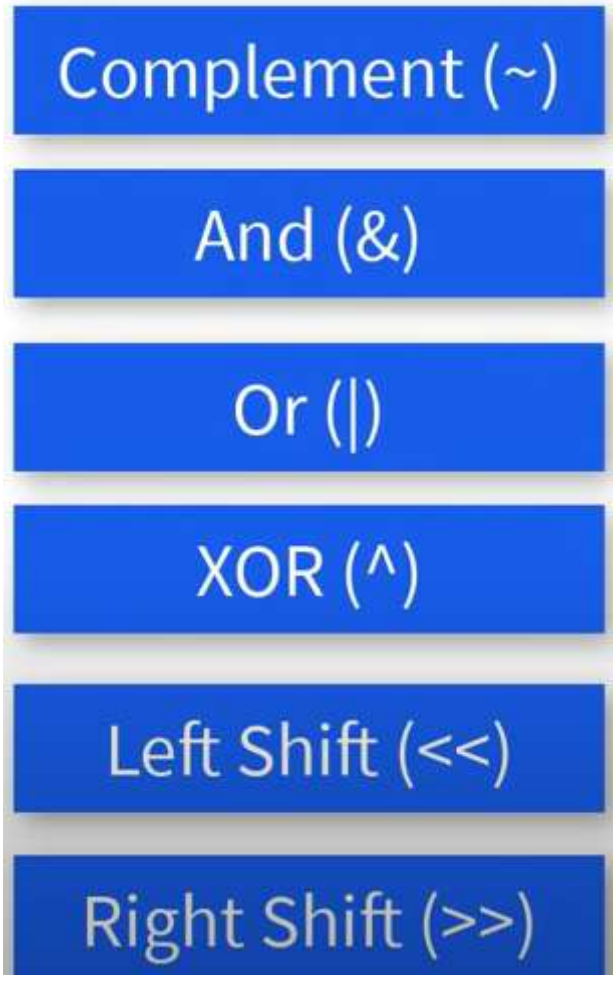


- internally it uses the rotational concept

BITWISE OPERATOR

- WE HAVE 6 OPERATORS

COMPLEMENT (~) || AND (&) || OR (|) || XOR (^) || LEFT SHIFT (<<) || RIGHT SHIFT (>>)



In [ ]:
```
print(bin(12))
print(bin(13))
```

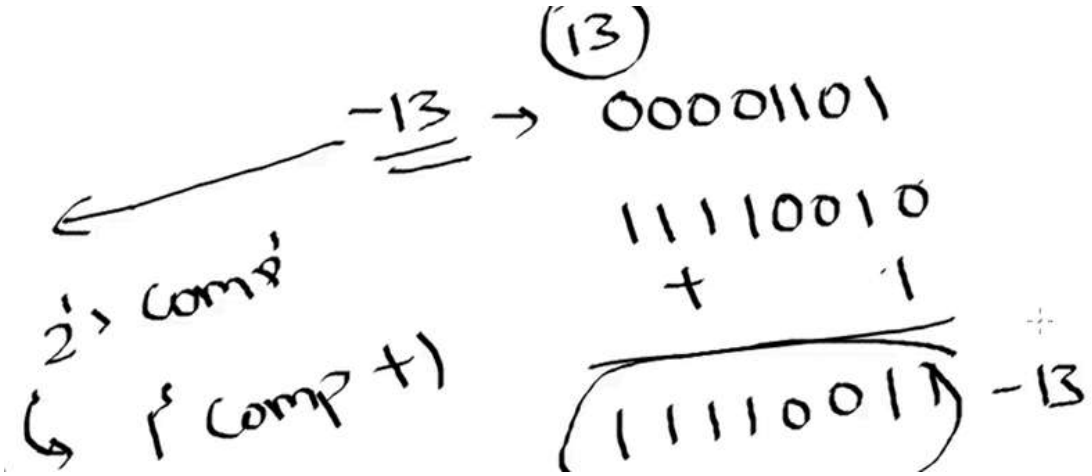
In [ ]:
```
0b1101
```

In [ ]:
```
0b1100
```

complement --> you will get this key below esc character

12 ==> 1100 ||

- first thing we need to understand what is mean by complement.
- complement means it will do reverse of the binary format i.e. - ~0 it will give you 1 ~1 it will give 0
- 12 binary format is 00001100 (complement of ~00001100 reverse the number - 11110011 which is (-13)
- in the virtual memory we cant store -ve number & the only way to store the -ve value by using complimentory
- but the question is why we got -13
- to understand this concept (we have concept of 2's complement
- 2's complement mean (1's complement + 1)
- in the system we can store +Ve number but how to store -ve number
- lets understand binary form of 13 - 00001101 + 1



In [ ]:
```
# COMPLEMENT (~) (TILDE  OR TILD)
~12 # why we get -13 . first we understand what is complment means (reversr of b
```

In [ ]:
```
~46
```

In [ ]:
```
~54
```

In [ ]:
```
~10
```

bit wise and operator

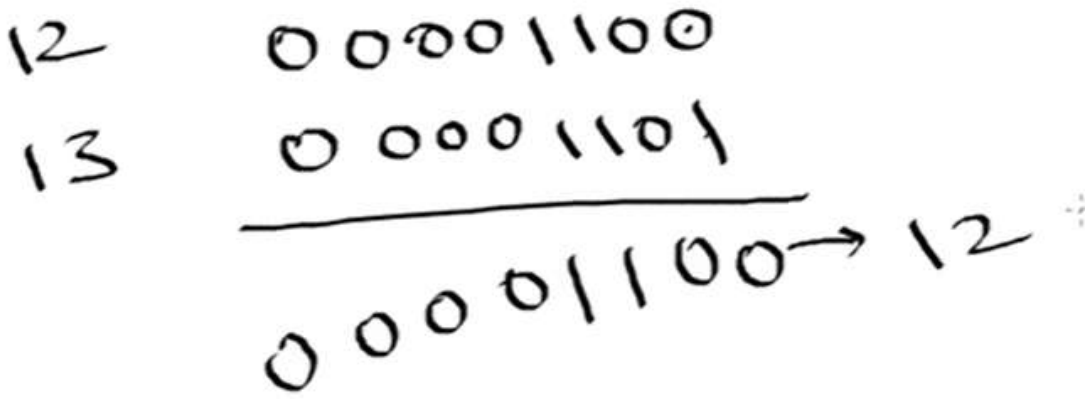
AND - LOGICAL OPERATOR ||| & - BITWISE AND OPERATOR
(we know that 1 & 1 is 1) 12 - 00001100 13 - 00001101 when we are add both then then outut we will get as 12

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

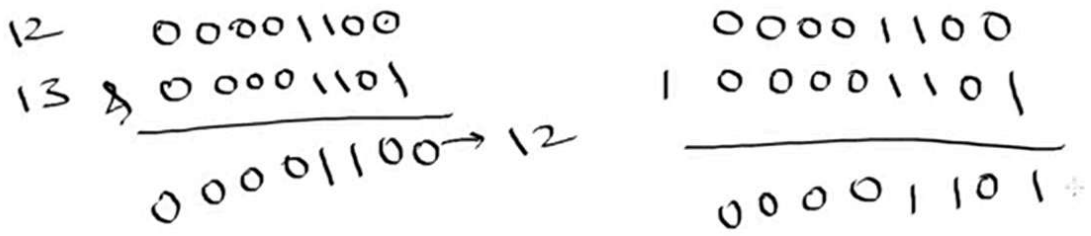


In [ ]:
```
12 & 13
```

In [ ]:
```
12 | 13
```

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

In [ ]:
```
1 | 0
```

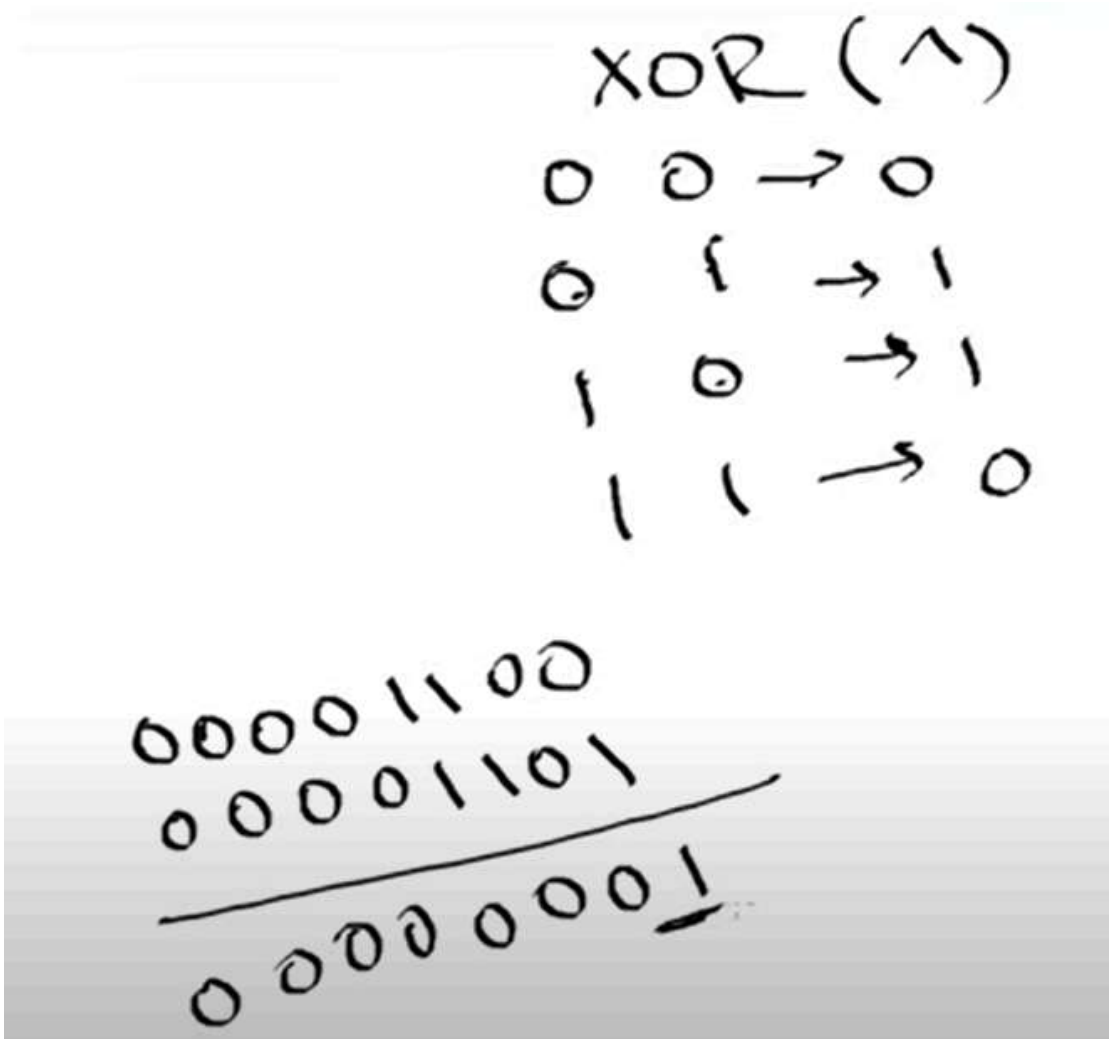


In [ ]:
```
bin(13)
```

In [ ]:
```
print(bin(35))
print(bin(40))
```

In [ ]:
```
35 & 40 #please do the homework conververt 35,40 to binary format
```

In [ ]:
```
35 | 40
```



In [ ]:
```
# in XOR if the both number are different then we will get 1 or else we will get
12 ^ 13
```

In [ ]:
```
print(bin(25))
print(bin(30))
```

In [ ]:
```
25^30
```

In [ ]:
```
bin(7)
```

In [ ]:
```
bin(25)
```

In [ ]:
```
bin(30)
```

In [ ]:
```
0b00111
```

In [ ]:
```
bin(10)
```

In [ ]:
```
10<<1
```

In [ ]:
```
10<<2
```

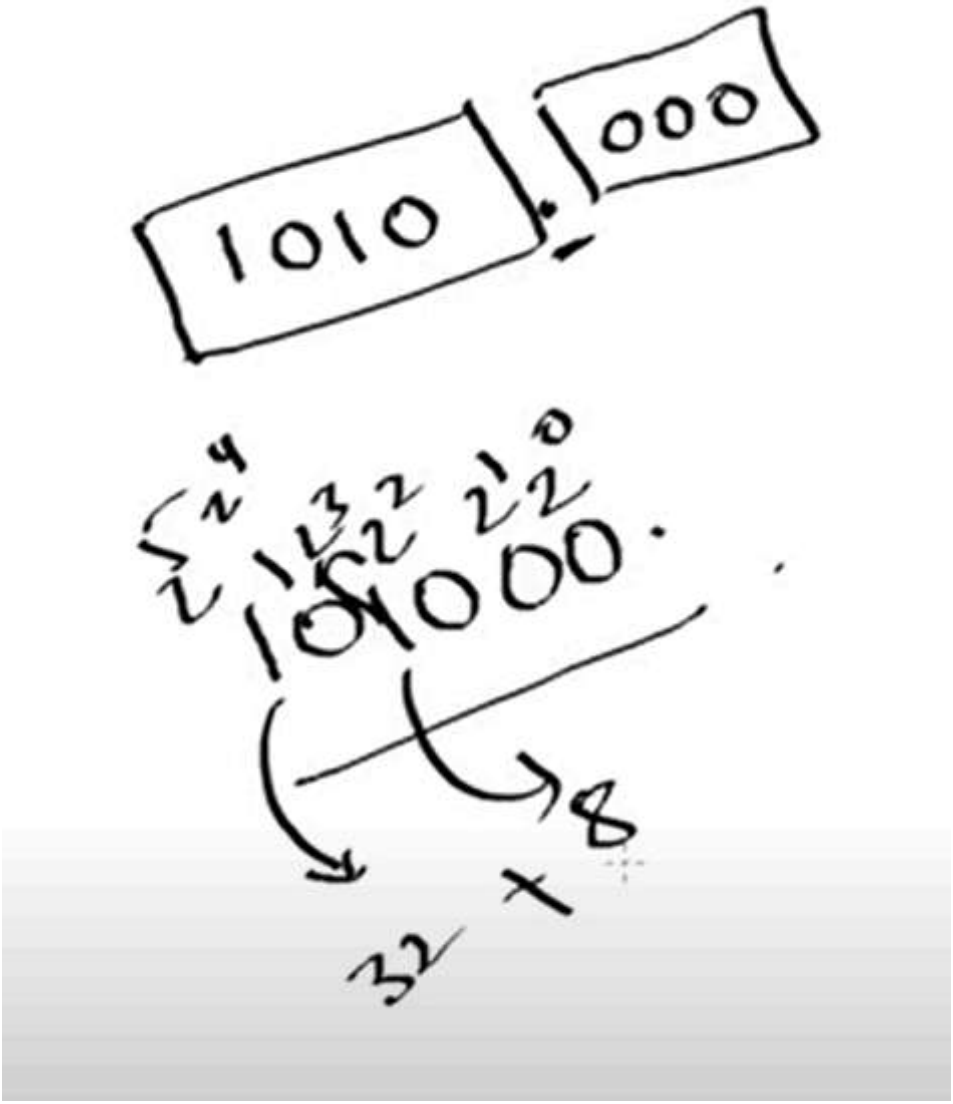
In [ ]:
```
bin(10)
```

In [ ]:
```
10<<1
```

In [ ]:
```
10<<2
```

In [ ]:
```
# BIT WISE LEFT SHIFT OPERATOR
# in left shift what we need to to we need shift in left hand side & need to shi
#bit wise left operator bydefault you will take 2 zeros ( )
#10 binary operator is 1010 | also i can say 1010
10<<2
```

In [ ]:
```
10<<3
```



In [ ]:
```
bin(20)
```

In [ ]:
```
20<<4 #can we do this
```

BITWISE RIGHTSHIFT OPERATOR

- left side we are gaining the bits
- right side we are lossing bits