Arithmetic operator

```
In [1]: x1,y1=10,5
```

```
In [2]: x1+y1
```

```
Out[2]: 15
```

```
In [3]: x1-y1
```

```
Out[3]: 5
```

```
In [4]: x1*y1
```

```
Out[4]: 50
```

```
In [5]: x1/y1
```

```
Out[5]: 2.0
```

```
In [6]: x1//y1
```

```
Out[6]: 2
```

```
In [7]: x1%y1
```

```
Out[7]: 0
```

```
In [8]: x1**y1
```

```
Out[8]: 100000
```

```
In [9]: 2**3
```

```
Out[9]: 8
```

Assignment operator

```
In [11]: x = 2
```

```
In [12]: x=x+2
```

```
In [13]: x
```

```
Out[13]: 4
```

```
In [14]: x+=2
```

```
In [15]: x
```

```
Out[15]: 6

In [16]: x+=2

In [17]: x

Out[17]: 8

In [18]: x*=2

In [19]: x

Out[19]: 16

In [21]: x-=2

In [22]: x

Out[22]: 14

In [23]: x/=2

In [24]: x

Out[24]: 7.0

In [25]: a,b=4,5

In [26]: a

Out[26]: 4

In [27]: b

Out[27]: 5
```

Unary Operator

```
In [28]: n = 7

In [29]: m=-(n)

In [30]: m

Out[30]: -7

In [31]: n

Out[31]: 7
```

```
In [32]: -n
```

```
Out[32]: -7
```

Relational Operator

```
In [33]: a=5
         b=7
```

```
In [34]: a==b
```

```
Out[34]: False
```

```
In [35]: a<b
```

```
Out[35]: True
```

```
In [36]: a>b
```

```
Out[36]: False
```

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

```
In [37]: a==b
```

```
Out[37]: False
```

```
In [38]: a=10
```

```
In [39]: a!=b
```

```
Out[39]: True
```

```
In [40]: # hear if i change b = 6
         b = 10
```

```
In [41]: a==b
```

```
Out[41]: True
```

```
In [42]: a>=b
```

```
Out[42]: True
```

```
In [43]: a<=b
```

```
Out[43]: True
```

```
In [44]: a<b
```

```
Out[44]: False
```

```
In [45]: a>b

Out[45]: False

In [46]: b=7

In [47]: a!=b

Out[47]: True
```

Logical operator

```
In [2]: a=5
        b=4

In [3]: a < 8 and b < 5 #refer to the truth table

Out[3]: True

In [4]: a < 8 and b < 2

Out[4]: False

In [5]: a < 8 or b < 2

Out[5]: True

In [6]: a>8 or b<2

Out[6]: False

In [7]: x = False
        x

Out[7]: False

In [8]: not x # you can reverse the operation

Out[8]: True

In [9]: x

Out[9]: False

In [10]: not x

Out[10]: True
```

Number system coverstion (bit-binary digit)

```
In [12]: 25
```

```
Out[12]:  25

In [13]:  bin(25)

Out[13]:  '0b11001'

In [14]:  int(0b11001)

Out[14]:  25

In [15]:  bin(30)

Out[15]:  '0b11110'

In [16]:  int(0b11110)

Out[16]:  30

In [17]:  int(0b11001)

Out[17]:  25

In [18]:  oct(25)

Out[18]:  '0o31'

In [19]:  int(0o31)

Out[19]:  25

In [20]:  int(0b11110)

Out[20]:  30

In [21]:  0o31

Out[21]:  25

In [22]:  0b11001

Out[22]:  25

In [23]:  int(0b11001)

Out[23]:  25

In [24]:  bin(7)

Out[24]:  '0b111'

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

```
Out[25]: '0o31'

In [26]: 0o31

Out[26]: 25

In [27]: int(0o31)

Out[27]: 25

In [28]: hex(25)

Out[28]: '0x19'

In [30]: 0x19

Out[30]: 25

In [31]: hex(16)

Out[31]: '0x10'

In [32]: 0xa

Out[32]: 10

In [33]: 0xb

Out[33]: 11

In [34]: hex(1)

Out[34]: '0x1'

In [35]: hex(25)

Out[35]: '0x19'

In [36]: 0x19

Out[36]: 25

In [37]: 0x15

Out[37]: 21
```

swap 2 - variable in python

```
In [1]: a = 5
        b = 6
```

In [2]:
```
a = b
b = a
```

In [3]:
```
print(a)
print(b)
```

```
6
6
```

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

In [6]:
```
temp = a1
a1 = b1
b1 = temp
```

In [7]:
```
print(a1)
print(b1)
```

```
8
7
```

In [8]:
```
a2 = 5
b2 = 6
```

In [9]:
```
#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 [10]:
```
print(a2)
print(b2)
```

```
6
5
```

In [11]:
```
0b110
```

Out[11]:
```
6
```

In [12]:
```
0b101
```

Out[12]:
```
5
```

In [13]:
```
print(0b110)
print(0b101)
```

```
6
5
```

In [14]:
```
print(0b101)
print(0b110)
```

```
5
6
```

In [15]:

```
#but when we use a2 + b2 then we get 11 that means we will get 4 bit which is 1 bit
print(bin(11))
print(0b1011)
```

```
0b1011
11
```

In [16]:

```
print(a2)
print(b2)
```

```
6
5
```

In [17]:

```
#there is other way to work using swap variable also which is XOR because it will n
a2 = a2 ^ b2
b2 = a2 ^ b2
a2 = a2 ^ b2
```

In [18]:

```
print(a2)
print(b2)
```

```
5
6
```

In [19]:

```
a2, b2
```

Out[19]:

```
(5, 6)
```

In [20]:

```
a2 , b2 = b2, a2  # how it work is b2 6 a2 is 5 first it goes into stack & then it
```

In [21]:

```
print(a2)
print(b2)
```

```
6
5
```

In [22]:

```
print(bin(12))
print(bin(13))
```

```
0b1100
0b1101
```

In [23]:

```
0b1101
```

Out[23]:

```
13
```

In [24]:

```
0b1100
```

Out[24]:

```
12
```

Bitwise operator

complement(~)

```
In [25]: ~12

Out[25]: -13

In [26]: ~45

Out[26]: -46

In [27]: ~90

Out[27]: -91

In [28]: ~10

Out[28]: -11
```

bitwise and operator

```
In [1]: 12&13

Out[1]: 12

In [2]: 12|13

Out[2]: 13

In [3]: 1&0

Out[3]: 0

In [4]: 1|0

Out[4]: 1

In [5]: bin(13)

Out[5]: '0b1101'

In [6]: print(bin(35))
        print(bin(40))

0b100011
0b101000

In [7]: 35 & 40

Out[7]: 32

In [8]: # in XOR if the both number are different then we will get 1 or else we will get 0
        12 ^ 13

Out[8]: 1
```

In [9]:
```
print(bin(25))
print(bin(30))
```

```
0b11001
0b11110
```

In [10]:
```
25^30
```

Out[10]:
```
7
```

In [11]:
```
bin(7)
```

Out[11]:
```
'0b111'
```

In [12]:
```
bin(25)
```

Out[12]:
```
'0b11001'
```

In [13]:
```
bin(30)
```

Out[13]:
```
'0b11110'
```

In [14]:
```
bin(10)
```

Out[14]:
```
'0b1010'
```

In [15]:
```
10<<1
```

Out[15]:
```
20
```

In [17]:
```
10>>2
```

Out[17]:
```
2
```

In [18]:
```
bin(10)
```

Out[18]:
```
'0b1010'
```

In [19]:
```
10<<1
```

Out[19]:
```
20
```

In [20]:
```
10<<2
```

Out[20]:
```
40
```

In [21]:
```
# BIT WISE LEFT SHIFT OPERATOR
# in left shift what we need to to we need shift in left hand side & need to shift
#bit wise left operator bydefault you will take 2 zeros ( )
#10 binary operator is 1010 | also i can say 1010
10<<2
```

```
Out[21]:  40
```

```
In [22]:  10<<3
```

```
Out[22]:  80
```

```
In [23]:  bin(20)
```

```
Out[23]:  '0b10100'
```

```
In [24]:  20<<4 #Can we do this
```

```
Out[24]:  320
```

right operator

```
In [25]:  bin(10)
```

```
Out[25]:  '0b1010'
```

```
In [26]:  10>>1
```

```
Out[26]:  5
```

```
In [27]:  10>>2
```

```
Out[27]:  2
```

```
In [28]:  10>>3
```

```
Out[28]:  1
```

```
In [29]:  bin(20)
```

```
Out[29]:  '0b10100'
```

```
In [ ]:
```