Membership Operator

Membership operator is binary operator and required 2 operands
Membership operator is used for searching or finding a given value into collection of values.
“in” keyword represents membership operator

1. in
2. not in

Membership operator returns boolean value (True/False)
If given value found within collection of values, it returns True else False

Syntax: value in collection-type

Operand1 can be of any type
Operand2 must be collection-type

Example:

```
>>> 10 in [10,20,30,40,50]
True
>>> 100 in [10,20,30,40,50]
False
>>> "naresh" in ["suresh","ramesh","kishore"]
False
>>> "naresh" in ["suresh","ramesh","kishore","naresh"]
True
>>> "a" in "naresh"
True
>>> "x" in "naresh"
False
```

Example:

```
# Write a program to find input character is vowel or not

ch=input("Enter any character ")
if ch in "aeiouAEIOU":
   print("Vowel")
else:
   print("Not Vowel")
```

Output

```
Enter any character a
Vowel

Enter any character E
Vowel

Enter any character x
Not Vowel
```

Example:

```
>>> 10 in (10,20,30,40)
True
>>> "py" in "python"
True
>>> 10 in (10,20,30,40)
True
>>> "py" in "python"
True
>>> 10 not in (10,20,30,40)
False
>>> 100 not in (10,20,30,40)
```

True

Bitwise Operators
Bitwise operators are binary operators and required 2 operands to perform operations.
Bitwise operators are applied in python on integer data type/values.

1. >> (right shift operator)
2. << (left shift operator)
3. & (bitwise and operator)
4. | (bitwise or operator)
5. ^ (bitwise XOR operator)
6. ~ (bitwise NOT operator)

Applications of bitwise Operators
1. Embedded Applications
2. Encryption and Decryption
3. Memory Management
4. Image/audio/video processing

Binary codes are two 1 and 0

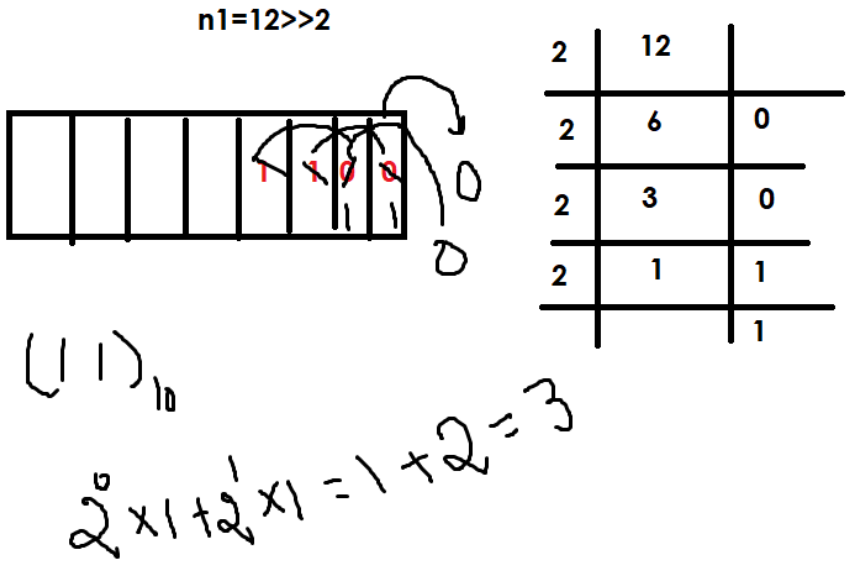
1 is called set bit
0 is called unset bit

>> Right shift operator
This operator is used to shift given n bits towards right side.

Syntax: operand>>n

This operator is used to delete bits
Right shift operator by deleting bits, the value is decremented.

Python runtime or operating system fills the data from right to left



$(11)_{10}$

$2^0 \times 1 + 2^1 \times 1 = 1 + 2 = 3$

Example:

```
>>> n1=12>>2
>>> print(n1)
3
>>> print(bin(12))
0b1100
>>> print(bin(n1))
0b11
>>> n2=15
>>> n3=n2>>3
>>> print(bin(n2))
0b1111
>>> print(bin(n3))
0b1
>>> print(n2,n3)
15 1
>>> n4=0b1010
>>> n5=n4>>1
```

```
>>> print(bin(n4),bin(n5))
0b1010 0b101
>>> print(n4,n5)
10 5
```

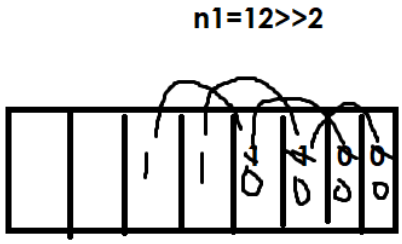
Formula : operand//2 pow n

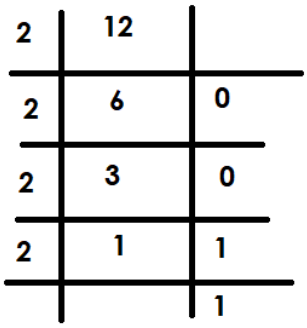
```
>>> a=256
>>> b=a>>3
>>> print(a,b)
256 32
>>> print(bin(a),bin(b))
>>> n1=0xa
>>> n2=n1>>2
>>> print(n1,n2)
10 2
>>> print(bin(n1),bin(n2))
0b1010 0b10
```

<< (Left shift operator)

This operator is used for shifting **n** bits towards left side.
This operator increment value by adding **n** bits at right side

Syntax: operand<<n






```
>>> a=12
>>> b=a<<2
>>> print(a,b)
12 48
>>> print(bin(a),bin(b))
0b1100 0b110000
```

Formula: operand*2 pow n

```
>>> n1=0xc
>>> n2=n1<<2
>>> print(n1,n2)
12 48
```

What is logic gate?

Logic gates are fundamental building blocks of digital circuits that perform basic logical operations on binary inputs, resulting in a single binary output. They are essential for processing data and making decisions within electronic devices like computers and calculators

Bitwise & operator

This operator is used for applying AND gate

It is a binary operator and required two operator

Truth table & operator

Opr1	Opr2	Opr1 & Opr2
1	1	1
0	0	0
1	0	0
0	1	0

a=12 ---> 1100
b=10 ---> 1010
c=a&b ---> 1000

```
>>> a=12
>>> b=10
>>> c=a&b
>>> print(a,b,c)
12 10 8
>>> print(bin(a),bin(b),bin(c))
0b1100 0b1010 0b1000
>>> n1=0b1100
>>> n2=0b1010
>>> n3=n1&n2
>>> print(bin(n1),bin(n2),bin(n3))
0b1100 0b1010 0b1000
>>> print(n1,n2,n3)
12 10 8
>>> n4=0xc
>>> n5=0xa
>>> n6=n4&n5
```

```
>>> print(bin(n4),bin(n5),bin(n6))
0b1100 0b1010 0b1000
>>> print(n4,n5,n6)
12 10 8
>>> n7=1.5&1.6
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    n7=1.5&1.6
TypeError: unsupported operand type(s) for &: 'float' and 'float'
```

Bitwise | (OR) operator

This operator is used for applying OR gate

Truth table of bitwise | operator

Opr1	Opr2	Opr1 | Opr2
1	0	1
0	1	1
0	0	0
1	1	1

a=12 --> 1100
b=10 --> 1010
c=a|b --> 1110

```
>>> a=12
>>> b=10
>>> c=a|b
>>> print(a,b,c)
12 10 14
>>> print(bin(a),bin(b),bin(c))
```

Bitwise ^ (XOR) operator

This operator represents XOR Gate

Truth table of XOR operator

Opr1	Opr2	Opr1 ^ Opr2
1	0	1
0	1	1
0	0	0
1	1	0

```
a=12     --> 1100
b=10     --> 1010
c=a^b    --> 0110
```

```
>>> a=12
>>> b=10
>>> c=a^b
>>> print(a,b,c)
12 10 6
>>> print(bin(a),bin(b),bin(c))
0b1100 0b1010 0b110
```

Example:

```
# Write a program to swap two numbers without using
# arithmetic operators and third variable

a=int(input("Enter First Number "))
b=int(input("Enter Second Number "))
print('before swaping ',a,b)
```

```
a=a^b
b=a^b
a=a^b
print('after swaping ',a,b)
```

Output

```
Enter First Number 12
Enter Second Number 10
before swaping  12 10
after swaping  10 12
```

Bitwise ~ not operator

This operator represents not gate

It is a unary operator and required one operand

Opr1	**~opr1**
0	1
1	0

Formula : -(opr+1)

```
>>> a=12
>>> b=~a
>>> print(a,b)
12 -13
>>> print(bin(a),bin(b))
0b1100 -0b1101
>>> a=-13
>>> b=~a
>>> print(a,b)
-13 12
>>> print(bin(a),bin(b))
-0b1101 0b1100
```

Walrus operator (:=) (OR) Assignment Expression operator

Walrus operator is introduced in python 3.8 version