Base conversion functions

1. bin()
2. hex()
3. oct()

bin() → This function is used to perform the following conversions

1. decimal to bin
2. hexa decimal to bin
3. octal to bin

```
>>> bin(20)
'0b10100'
>>> bin(0xb)
'0b1011'
>>> bin(0o12)
'0b1010'
```

hex() → This function is used to perform the following conversions

1. decimal to hexadecimal
2. octal to hexadecimal
3. binary to hexadecimal

```
>>> hex(10)
'0xa'
>>> hex(11)
'0xb'
>>> hex(255)
'0xff'
>>> hex(0o12)
'0xa'
>>> hex(0b1010)
'0xa'
```

oct(): This function is used to perform the following conversion

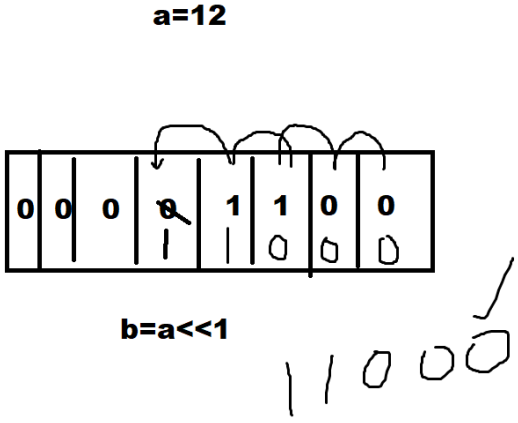
1. decimal to octal
2. hexadecimal to octal
3. binary to octal

```
>>> oct(10)
'0o12'
>>> oct(0xa)
'0o12'
>>> oct(0b1010)
'0o12'
```

Left Shift Operator

This operator is used to shift number of bits towards left side.
By shifting number of bits towards left side the value get incremented by adding those many number bits at right side.

Syntax: opr<<n




```
>>> a=12
>>> b=a<<1
>>> print(a,b)
12 24
>>> print(bin(a),bin(b))
0b1100 0b11000
>>> x=15
>>> y=x<<2
>>> print(bin(x),bin(y))
0b1111 0b111100
>>> print(x,y)
```

15 60

Bitwise (&) and operator

This bitwise operator is used to apply and gate

Syntax: Opr1 & Opr2

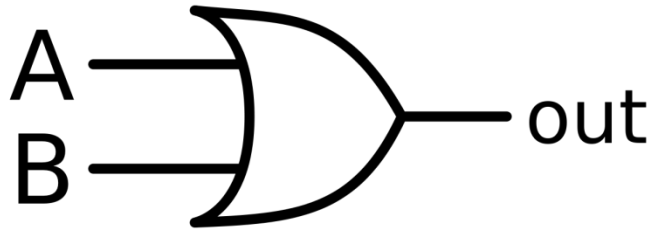
Truth table of (&) operator

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

```
>>> A=0b1
>>> B=0b1
>>> A&B
1
>>> X=0b1001
>>> Y=0b1100
>>> Z=X&Y
>>> print(bin(X),bin(Y),bin(Z))
```

Bitwise (|) or operator

This bitwise operator is used to apply or gate
Syntax: Opr1 | Opr2



Truth table of or (|) operator

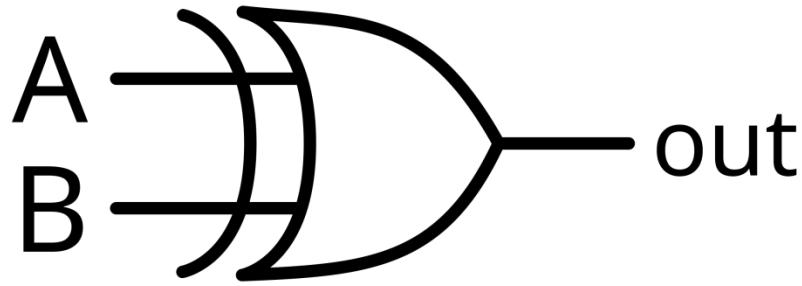
Opr1	**Opr2**	**Opr1 | Opr2**
1	0	1
0	1	1
1	1	1
0	0	0

```
>>> A=0b1
>>> B=0b0
>>> A|B
1
>>> X=0b1010
>>> Y=0b1110
>>> Z=X|Y
>>> print(X,Y,Z)
10 14 14
>>> print(bin(X),bin(Y),bin(Z))
```

Bitwise (^) XOR Operator

This bitwise operator is used to apply XOR gate

Syntax: Opr1 ^ Opr2



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

```
>>> A=2
>>> B=3
>>> C=A^B
>>> print(A,B,C)
2 3 1
>>> print(bin(A),bin(B),bin(C))
0b10 0b11 0b1
>>> X=1.5
>>> Y=2.5
>>> Z=X&Y
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    Z=X&Y
TypeError: unsupported operand type(s) for &: 'float' and 'float'
```

~ Bitwise not operator

This operator is used inverse bits
It is a unary operator, it required one operand
Truth table of not operator

Opr1	~Opr1
1	0
0	1

```
>>> a=0b1
>>> b=~a
>>> print(bin(a),bin(b))
0b1 -0b10
>>> print(a,b)
1 -2
>>> x=0b101
```

```
>>> y=~x
>>> print(bin(x),bin(y))
0b101 -0b110
>>> print(x,y)
5 -6
>>> a=7
>>> b=~a
>>> print(a,b)
7 -8
```

Formula : -(opr+1)

Example of no operators in python like ++ and --

```
>>> a=10
>>> a++
SyntaxError: invalid syntax
>>> ++a
10
>>> +-10
-10
>>> --10
10
```

Assignment Operators OR update operators OR Compound Assignment Operators

A single operator which perform two operations

1. Binary Operation
2. Assignment

1. +=
2. -=
3. *=
4. /=

5. //=
6. %=
7. **=
8. >>=
9. <<=
10. &=
11. |=
12. ^=

Example:

```
a=10
print(a)
a+=5 # a=a+5
print(a)
a-=1 # a=a-1
print(a)
a*=5  # a=a*5
print(a)
a/=2 # a=a/2
print(a)
a//=3 # a=a//2
print(a)
a**=2 # a=a**2
print(a)
```

Output

```
10
15
14
70
35.0
11.0
121.0
```

Example:

```
>>> a=5
```

```
>>> b=6
>>> a+=b
>>> print(a,b)
11 6
>>> x=5
>>> x+=1+5
>>> print(x)
11
>>> y=6
>>> y+=6-3
>>> print(y)
9
```

Walrus Operator (:=)

Walrus operator is introduced in python 3.8 version.
Walrus operator is also called assignment expression operator
It is a binary operator and required 2 operands.

```
>>> a=5
>>> b=2
>>> c=(x:=a+b)*(y:=a-b)
>>> print(a,b,c,x,y)
5 2 21 7 3
>>> p:=100
SyntaxError: invalid syntax
```