Operators

what is operator?

operator is a special charecter,which is used to perform the operation on the data of the operands/objects.

```
x=10
y=20

x + y   --> expression(both combination of operators and operands)

        --> x and y are the operands
        --> + is a Operator
        --> 10 and 20 are the data of the operands
```

types of operators:

1).Arithmetic Operators

2).Comparission/Relational Operators

3).Logical Operators

4).Bitwise Operators

5).Assignment Operators

6).Identity Operators

7).Membership Operators

Arithmetic Operators:

these operators are used to perform the arithmetic calculations.

arithmetic calculation	arithmetic operator	input x=10,y=3	result
addition	+	10+3	13
subtraction	-	10-3	7
multiplication	*	10*3	30
division (floating point division)	/	10/3	3.3333335
floor division	//	10//3	3

(integer division)

modulo % 10%3 1

exponent ** 10**3 1000

ex:
---

arith.py

```
x,y=10,3
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x//y)
print(x%y)
print(x**y)
```

output

```
13
7
30
3.3333333333335
3
1
1000
```

string concatination:
---

if we want to perform string concatination to required both inputs are string objects only.

```
>>> x="siva"
>>> y="krishna"
>>> z=3
>>> x+y
'sivakrishna'
>>> x+z
TypeError: can only concatenate str(not "int") to str
```

string multiplication:
---

if we want to perform string multiplication to required at least one input is integer object.

```
>>> x="siva"
>>> y= "krishna"
>>> z=3
```

```
>>> x*y
TypeError: can't multiply sequence by non-int of type 'str'
>>> x*z
'sivasivasiva'
>>> y*z
'krishnakrishnakrishna'
>>> z*y
'krishnakrishnakrishna'

examples in python2.x
---------------------
>>> x=10
>>> y=3
>>> x/y
3
>>> x//y
3

examples in python3.x
---------------------
>>> x=10
>>> y=3
>>> x/y
3.3333333333333335
>>> x//y
3

Comparision/Relational Operators:
---------------------------------
these operators are used to compare the data of the operands.

these operators are to return the output as boolean format(True/False)

                ==
                !=
                <
                <=
                >
                >=

ex:
---
        comp.py
        -------
x,y=10,3
print(x==y)
print(x!=y)
print(x<y)
print(x<=y)
print(x>y)
```

```
print(x>=y)

        output
        ------
False
True
False
False
True
True

Logical Operators:
------------------
        these operators are used to perform mathematical logical operations.

        these operators are takes the input as boolean and to return the output as
boolean.

        logical operation                               logical operator
                                                        c/c++     Python
        -----------------                               ---------------
        logical and                                     &&        and

        logical or                                      ||        or

        logical not                                     !         not

logical operators truth tables:
-------------------------------

                input                               output
-------------------------------------------------------------------------------
        x               y          x and y              x or y  not x           not
y
-------------------------------------------------------------------------------
   False          False     False                False  True            True

   False          True      False                 True  True            False

   True   False     False                True  False           True

   True   True       True                True  False           False

ex:
---
        logical.py
        ----------
print(False and not False)
print(False and not False)
Print(not True and True)
```

```
Print(not False or False)
print(False and True)
Print(True and not False)
Print(not False)

        output
        -----
False
True
False
True
False
True
True
```

Bitwise Operators:
---

these operatiors are used to perform the operation on the binary data.

the bitwise operators takes the input as decimal and to return the output as decimal.

the bitwise operator internally to perform the following opertions,

step1: to convert decimal into binary

step2: to perform the operation on the binary data

step3: to convert binary into decimal

the bitwise operators are

bitwise and --> &

bitwise or --> |

bitwise xor --> ^

bitwise negation --> ~

bitwise leftshift --> <<

bitwise rightshift --> >>

bitwise and(&) example

```
>>> x=10
>>> y=3
>>> x&y
2
```

step1: to convert decimal into binary

x=10 --> 1010

y=3 --> 0011

step2: to perform the operation on binary data

x	y	x&y	
0	0	0	1010
0	1	0	0011
1	0	0	----
1	1	1	0010

step3: to convert binary into decimal

0010 --> 2

bitwise or(|)example:

```
>>> x=10
>>> y=3
>>> x|y
11
```

step1: to convert decimal into binary

x=10 --> 1010

y=3 --> 0011

step2: to perform the operation on binary data

x	y	x|y	
0	0	0	1010
0	1	1	0011
1	0	1	----
1	1	1	1011

step3: to convert binary into decimal

1011 -> 11

bitwise xor(^) example

```
---------------------

>>> x=10
>>> y=3
>>> x^y
9

        step1: to convert decimal into binary

                x=10      -->   1010

              y=3         -->   0011

        step2: to perform the operation on binary data


                x  y  x^y
                ---------
                0  0    0               1010
                0  1  1         0011
                1  0  1         ----
                1  1  0         1001

        step3: to convert binary into decimal

                1001    --> 9


bitwise negation(~) example
----------------------------
>>> x=10
>>> ~x
-11

                                10
                                 | to convert decimal into binary
                                1010
                                 | to apply the one's compliment
                                0101
                                 | to padding one bit i.e., 1 at leftside
                               10101
                                 | to take padding bit result is negative
          -1*2**4+0*2**3+1*2**2+0*2**1+1*2**0
                                 |
           -1*16+0*8+1*4+0*2+1*1
                                 |
                      -16+0+4+0+1
                                 |
                                -11
```

note:
----
the even no.of negations always to return the output as same number.

```
>>> x=3
>>> ~~x
3
>>> y=8
>>> ~~~y
8
>>> z=-3
>>> ~~~z
-3
```

the odd no.of negations to return the output as -(num+1)

```
>>> x=3
>>> ~x
-4
>>> y=8
>>> ~~~y
-9
>>> z=-3
>>> ~~~~~z
2
```

bitwise leftshift(<<) example

```
>>> x=10
>>> x<<2
40
```

bitwise rightshift(>>) example

```
>>> x=10
>>> x>>2
2
```

note:
----
whenever we are moveing the bits at leftside that no. of bits are fit into the memory but whenever we are moeing the bits at rightside that no.of bits are exit from memory.

assignment operators:
---
these operators are used to assign the values.

in python, the assignment operators can be categorized into 3-types, they are

1). normal assignment operators

2). shorthand assignment operators

3). warlus assignment operators

normal assignment operators:

'=' is a normal assignment operator

this operator is used to assign the Rvalue to the Lvalue

ex:
--
x=10

shorthand assignment operators
---
the normal asssignment operator which contains prefix with any another operator like arithmetic,bitwise,....., that type of assignment operators are called shorthand assignment operators.

ex:
--
x=10

x += 3 --> x=x+3 --> x=13

x -= 3 --> x=x-3 --> x=7

x *= 3 --> x=x*3 --> x=30

x &= 3 --> x=x&3 --> x=2

x |= 3 --> x=x|3 --> x=11

note:
---
python dont supporting both increment and decrement operators concept but we can achieve that concept indirectly by using short-hand assignment operators concept.

c/c++		python	
i++	--> i=i+1	i+=1	--> i=i+1
i--	--> i=i-1	i-=1	--> i=i-1

warlus assignment operator
---
this operator is used to perform both initialization and evalution at a time in that case we are using warlus assignment operator concept.

this operator introduced from python3.8 versions onwards.

':=' is a warlus assignment operator

ex:
--
```
x=10
print(x)
y=20
print(y)
print(x+y)
```

output:

10
20
30

ex2:

```
print(x:=10)
print(y:=20)
print(x+y)
```

output:

10
20
30

identity operators:
---
these operators are used to compare the address of the objects.

in python, the identity operators are is,is not

if address are equal the 'is' operator to return True otherwise the 'is' operator to return False

if address are equal the 'is not' operator to return True otherwise the 'is not' operator to return False

ex1:

```
---
>>> x=10
>>> y=10
>>> x==y
True
>>> x is y
True
>>> id(x)
2181360589184
>>> id(y)
2181360589184
>>> x is not y
False

ex2:
---
>>> a=[1,2,3]
>>> b=[1,2,3]
>>> a==b
True
>>> a is b
False
>>> id(a)
2181366006656
>>> id(b)
2181365775744
>>> a is not b
True

membership operators:
--------------------
        these operators are used to searching an element/charecter in a given
iterable object.

        in python, the membership operators are in,not in

if element/charecter is find in that case the 'in' operator to return True
otherwise the 'in' operator to return False.

if element/charecter is not find in that case the 'not in' operator to return True
otherwise the 'not in' operator to return False.

ex1:
---
>>> x="siva"
>>> 'a' in x
True
>>> 'b' in x
False
>>> 'V' not in x
```

```
False
>>> 'k' not in x
True

ex2:
---
>>> y=[4,2,7,9]
>>> 2 in y
True
>>> 3 in y
False
>>> 9 not in y
False
>>> 6 not in y
True
```

operator presidency:
---

whenever our expression contains multiple operators in that we need to identify which operator is executed first,which operator is executed second.,..., which operator is executed last by using operator presidency concept.

in our expression, the multiple operators having same priority in that case we are following left to right associativity.

in python, the arithmetic operators are following PEMDAS rule

P --> ( )

E --> **

MD --> *,/,//,%

AS --> +,-

the logical operators are following

and
or

the bitwise operators are following

~
<< or >>
&
^
|