Python Notes

OPERATORS CONCEPT -

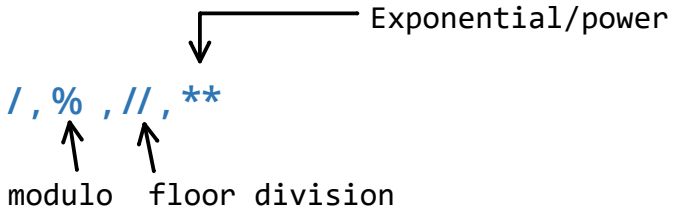
A General definition of operator - Any person who is doing certain activity is called operator **ex**- **train driver - operating train, cameraman- operating camera.**

In langauge term - symbol which is reponsible to perform certain operation.

Types of operators -

1. **Arithmetic operators**
2. **Relational operators**
3. **Logical operators**
4. **Assignment operaotrs**
5. **Bitwise operators**
6. **Special operators**

1. Arithmetic Operators - +,-,*,/,%,//,**



- + operator does addition operations.
  ex --> **print(10 + 20) #30**

- - operator does substraction operations
  ex --> **print(20 - 10) #10**

- * operator does multiplication operations.
  ex --> **print(2*3)** #6

- % operator does division operation but gives us remainder as output.
  ex --> **print(10%2)** #0
  print (5%2) #1

- / operator does normal division operation and gives you quotient as output.
  ex --> **print(10/2)** #5.0

```
print( 10.0/2 ) #5.0
```

Why 5.0 ? Why not 5 ?

> **NOTE:** In python, / operator always performs floating point arithmetic only. Therefore the result of / operator is float value only.

- **//(floor division operator)** can perform both integral arithmetic and floating point arithmetic.
  (i) If atleast one argument is float type, then result is always float type.
  (ii) If both arguments are int type, then result is always int type.

NOTE: The result is always floor value.
2.5 --> floor int value is **#2**
2.5 --> floor float value is **#2.0**

- ** operator : power operator/exponential operator.
  ex --> 23 means 2 to the power 3. #8**
  23.0 #8.0**

NOTE : we can use + operator for strings also
'durga' + 'soft' --> durgasoft --> concatenation(link two things together)
'durga' + 10 --> error. Both arguments should be string type.

*** operator for the strings :**
one argument should be str and other argument should be int.
It is called repeatetion operator
ex --> 'ankit'*5 --> #ankitankitankitankitankit

Anything divide with zero --> error we are going to get.

2. Relational operators (comparison operators) - <, <=, >,>=

- we use these relational operators for number comaprison -->
  example -->

```
x=10
y=20
print(x>y) #false
print(x>=y)#false
print(x<y)#true
print(x<=y)#true
```

- we can also use these relational operators for strings -->

- comparison is based on unicode/ascii value
  A --> **65** **similarly** a --> **97**
  B --> **66** b --> **98**
  upto upto
  Z--> **90** z --> **122**

- ord() is a function to find the ascii value of a character.

  example -->
  ```
  s1='durga'
  s2='ravi'
  print(s1<s2) #True
  print(s1>s2) #False
  ```

  Here in this example, the ascii value of 'd' in durga is less than the ascii value of 'r' in ravi and hence s1<s2.

  Remember that if the first character is same in both the strings then the ascii value of 2nd character of the string will be compared and if 2nd char is same then 3rd char will be comparted and so on.

  we can also use relational operators for booleans -

  True --> **1**
  False --> **0**

  example -->
  ```
  print(True<False) #False
  print(True>False) #True
  ```

  NOTE : Chaining/Nesting of relational operator is possibe.

1. If all comparisons returns True then only result is True
   example --> `print(10<20<30<40<50)` **#True**

2. If atleast one comparison fails then result is False.
   example --> `print(10<20<30<40>50)` **#False**

3. Equality Operators - ==, !=

== --> **equals to**
!= --> **not equals to**

we use these operators to check if the two values are equal or not .

example -->
```
10==20              #false
10 !=20             #True
'durga' == 'durga'  #True
False == False      #True
```

- These operators won't raise any error even the types are incomparable.
  If the types are incomparable then we will ger False.
  example --> 10 == 'durga' #False

- Chaining/Nesting of Equality operator is possibe.

1. If all comparisons returns True then only result is True
   example --> print(10==20==30==40==50) #True
   print(10!=20!=30!=40!=50) #True

2. If atleast one comparison fails then result is False.
   example --> print(10==10==10==40) #False

What is the difference between == operator and is operator ?

== operator is always meant for content comparison where as is operator is meant for adrress/reference comparison.
example -->

l1=[10,20,30]
l2=[10,20,30]
If we print(l1==l2) the output we get: **True**
"==" **operator comparing content.**

If we print address of l1 & l2 both are different even though the content is same but the objects are different.
print(l1 is l2) the output is : **False.**

REMEMBER: l1 and l2 returns TRUE iff both l1 and l2 are pointing to same objects.

4. Logical Operators - and, or, not

we can apply for all types :

For boolean types -->

and ==> If both the arguments are True then only the result is True
or ==> If atleast one argument is True then the result is True
not ==> complement(not true means false and not false means true)

For non-boolean types -->

int ==> for int 0 means False and non-zero means True.

REMEMBER : If a and b are non-boolean then the result is boolean is non-boolean only. The result wil be either a or b.
But the Question is when is A or When is B ?

a and b -->

- If a is evaluated as True, then the result is b.
- If a is evaluated as False,then the result is a.

```
                      a      b
 example --> print(10 and 20) #20
             print(10 and 0)  #0
```

a or b -->

- If a is evaluated to True,the result is always a.
- If a is evaluated to False, the result is b.

not x -->

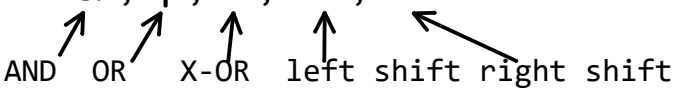
The result type is always boolean.

- If x is evaluated to True then result is False
- If x is evaluated to False then result is True

str ==> **empty string is always treated as False**
non-empty string is always treated as True

None ==> **None is always treated as false.**

Bitwise Operators --> & , | , ^ , << , >>



Why the Name Bitwise ?
Because these operators will be evaluated bit by bit.
These are applicable only for int and boolean.

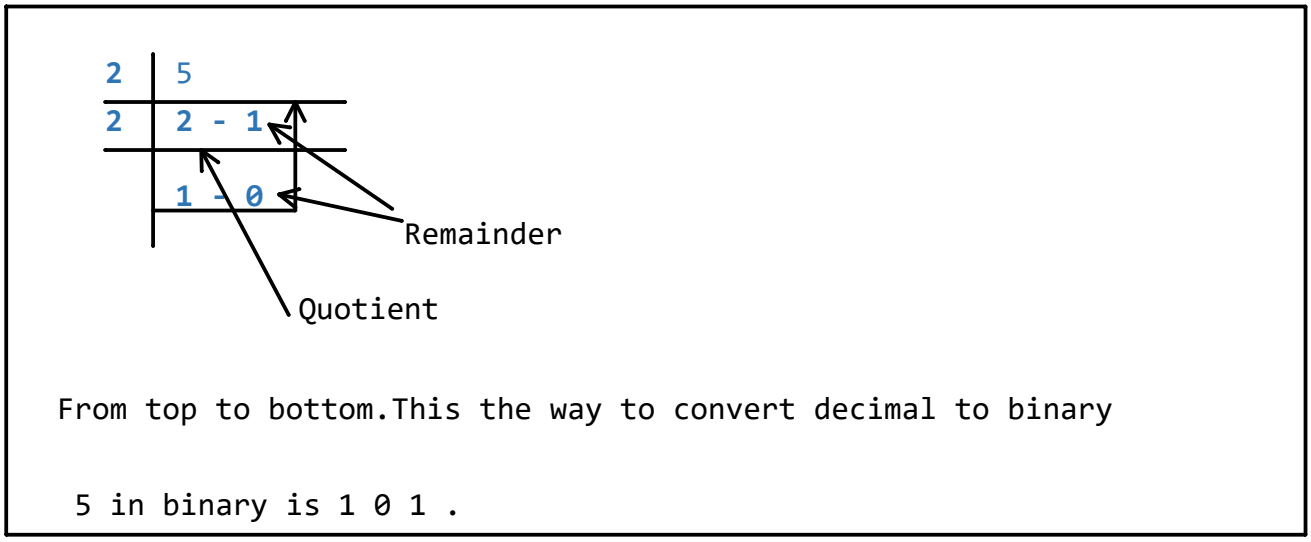
```
& ----> If both bits are 1 then only result is 1.
print(4 & 5)  #4
4 in binary --> 1 0 0
5 in binary --> 1 0 1
                1 0 0 --> 4 in decimal

| ----> If atleast one bit is 1 then result is 1.
print(4 | 5)  #5
4 in binary --> 1 0 0
5 in binary --> 1 0 1
                1 0 1 --> 5 in decimal

^  ----> If both bits are different then only result is 1.
print( 4 ^ 5)  #1
4 in binary ---> 1 0 0
5 in binary ---> 1 0 1
```

0 0 1 --> 1 in decimal

bitwise complement operator(~) : means reversing the bits: 1 --> 0, 0--->1



From top to bottom.This the way to convert decimal to binary

5 in binary is 1 0 1 .

In memory, 5 is represented in 28 bits in python:
First bit reserved for sign :
0 means +ve number
1 means -ve number

If it is +ve number the remaining bits represents value directly.

If it is -ve number the reamining bits represents value in 2's complement form.

Now 5 is a positve integer -->
5 ==> 0 000000..... 0 1 0 1
~5==> 1 111111..... 1 0 1 0
first bit is 1. So it is a -ve number therefore the remaining bits represents value in 2's complement.

How to find 2's complement ?
first find 1's complement and then add 1.

How to find 1's complement ?
Buy interchanging 0's and 1's.

~5==> **1 remaining bits --> 111111..... 1 0 1 0**
+1

1's complement --> 000000..... 0 1 0 1
add 1 + 1
1 1 0

these are the remainig bits --> 00000..110 --> 6
Now 1 00000..110 --> -6
-ve remaining bits representing 6

Addtion in binary numbers --> 0 + 0 = 0
1 + 0 = 1
1 + 1 = +1 1
+ 1
1 0

Left shift operator (<<) -->
left shift 10 by 2 bits
10 << 2
After left shift, right empty cells should be filled with 0 always
28 bits representation of 10:
00000..0001010
Now we will eliminate 2 bits from the far left side and add two 0 bits on the right side.
0 0 0. . 0 0 0 1 0 1 0 X X ← Empty cells
0 0 0. . 0 0 0 1 0 1 0 0 0

↑
most significant bit
It is 0 so the number is positive.
The Answer will be 40.

Right shift operator(>>) -->
10 >> 2
If we perform right shift then empty cells will be on left side.\
The empty cells always filled with sign bit
0 0 0 0 .. 0 0 0 1 0 1 0
X X 0 0 0 0 ..0 0 0 1 0
0 0 0 0 0 0.. 0 0 0 1 0

empty cells on the left side
If a positive number empty cells will be filled with 0
In our example 10 was a positive integer.

6. Assignment Operators --> =

x = 10 --> assigning value 10 to x.

x += 10 --> compound assignment operators
    example --> x = 10
                x += 10 ==> add 10 to x.   #20

                x=20
                x -= 10 ==> substract 10 from 20   #10

we can use -->
*=

&=

/=

//=

%=

**=

|=

>>=

<<=

etc..

Remember that Increment and Decrement operators not available in Python.

Ternary Operators -->

a+b --> operators for two arguments is called Binary operators
~a --> operators for one arguments is called unary operators

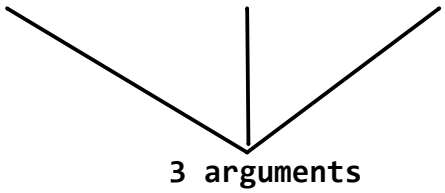
In the same way If we apply operator for three arguments such type of operator is called ternary operator.

Syntax for ternary operator in Python ->
 x = firstvalue if condtion else secondvalue

x= 30 if 10<20 else 40

```
x = firstvalue if condtion else secondvalue

x=   30   if   10<20   else   40
```



Output --> 30

Special Operators -->

1. **Identity operators**
2. **Membership operators**

- **Identity operators --> `is and is not`**

`a is b returns True iff both a and b are poinitng to the same object`
(address comparison or reference comparison)

`a is not b returns True iff both a and b are not poinitng to the same object.`

> **NOTE : These operaotrs will work when object reusability is there.**
> **In case of list data type this operators won't since object reusability is not there.**

- **Membership operators --> `in operator and not in operator`**

x in sequence ==> returns True if x is present in sequence
x not in sequence ==> returns True if x not present in sequence

Operator Precedence -->

`which operator will be evaluated first is depends on operator precedence.`

In python Precedence is --->

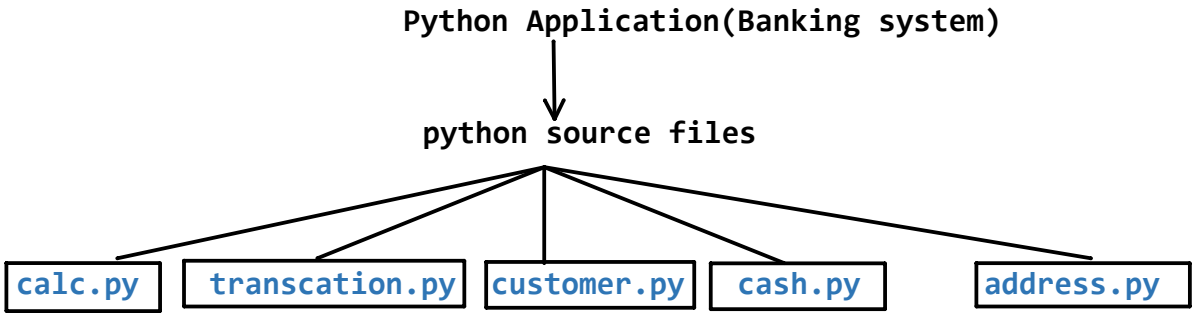
() > ** > ~,- > *,/,%,// > +,- > <<,>> > & > ^ >| > <,>,<=,>=,!=, == > =,+=,-=.. > is,is not >in,not in, > not > and > or

NOTE: *if more than one operator are there of same precedence then solve Left to Right*

How to perform Mathematical operations by using Math module :

Module : The most important word in python.

We know that a single python application can have many source files.



Here we have different source files but There will be some common functions(user-defined or pre-defined) that is going to use in every python files. Those functions that are common in all the python files,we will define those function in a single python file(calc.py) and that file will be called Module.

calc.py is my own python source file which contains the commonly required functions and these functions can be used in any python source file.

Conclusion: *Module is nothing but python source file which contains functions,variables,classes,etc.*
Example in VS code.

Now if you want to use module in your python file, we have to import
module : `import modulename`
`modulename.function`

NOTE: ***IF we execute the module file , we will not any output since we are just defining or declaring things there.***

Module Aliasing : Calling someone with alternate name,that is easy to call.

In order to use module,we have to write module name in everyline(calc.). Suppose the module name is very big, so it will be hard to write it down in every freaking line. That's why the Aliasing is there.
syntax : `import calc as c`

Now Suppose we just want to call only one function from module .

```
syntax : from calc import sum
            sum(10,20)
syntax : from calc import sum,product
          sum(10,20)
          product(20,30)
syntax : from calc import * ——this star means all defined fun,var
etc.
          sum(10,20)
          product(20,30)
          print(BankName)
```

In all these cases, we don't need to write calc. in every line.

Aliasing Members name(functions,variables,etc) --->

```
syntax: from calc import sum as s
        s(10,20)
```

Suppose we forgot or don't know how many members are there in calc.

```
syntax : import calc
          print(dir(calc)) or help(calc)
```

Math Module : It is predefined module which contain several functions for mathematical operations.

```
syntax : import math
          help(dir(math))
```