OPERATORS

- **Operator is a symbol that performs certain operations.**
- **Python provides the following set of operators**

1) **Arithmetic Operators**
2) **Relational Operators OR Comparison Operators**
3) **Logical operators**
4) **Bitwise oeprators**
5) **Assignment operators**
6) **Special operators**

1) Arithmetic Operators:

1) **+ → Addition**
2) **– → Subtraction**
3) *** → Multiplication**
4) **/ → Division Operator**
5) **% → Modulo Operator**
6) **// → Floor Division Operator**
7) **** → Exponent Operator OR Power Operator**

Eg: **test.py**

```
1) a=10
2) b=2
3) print('a+b=',a+b)
4) print('a-b=',a-b)
5) print('a*b=',a*b)
6) print('a/b=',a/b)
7) print('a//b=',a//b)
8) print('a%b=',a%b)
9) print('a**b=',a**b)
```

Output:

```
Python test.py OR py test.py
a+b = 12
a-b= 8
a*b= 20
a/b= 5.0
```

a//b= 5
a%b= 0
a**b= 100

Eg:

```
1)  a = 10.5
2)  b=2
3)
4)  a+b= 12.5
5)  a-b= 8.5
6)  a*b= 21.0
7)  a/b= 5.25
8)  a//b= 5.0
9)  a%b= 0.5
10) a**b= 110.25
```

Eg:

10/2 → 5.0
10//2 → 5
10.0/2 → 5.0
10.0//2 → 5.0

Note:

- / operator always performs floating point arithmetic. Hence it will always returns float value.
- But Floor division (//) can perform both floating point and integral arithmetic. If arguments are int type then result is int type. If atleast one argument is float type then result is float type.

Note:

- We can use +,* operators for str type also.
- If we want to use + operator for str type then compulsory both arguments should be str type only otherwise we will get error.

```
1) >>> "latha"+10
2)  TypeError: must be str, not int
3) >>> "latha"+"10"
4) 'latha10'
```

- If we use * operator for str type then compulsory one argument should be int and other argument should be str type.

2* "latha"

"latha"*2
2.5*"latha" -> TypeError: can't multiply sequence by non-int of type 'float'
"latha"*"latha" -> TypeError: can't multiply sequence by non-int of type 'str'

- **+ → String Concatenation Operator**
- *** → String Multiplication Operator**

Note: For any number x,
x/0 and x%0 always raises "ZeroDivisionError"

10/0
10.0/0
.....

2) Relational Operators: >, >=, <, <=

```
1)  a=10
2)  b=20
3)  print("a > b is ",a>b)
4)  print("a >= b is ",a>=b)
5)  print("a < b is ",a<b)
6)  print("a <= b is ",a<=b)
7)
8)  a > b is  False
9)  a >= b is  False
10) a < b is  True
11)     a <= b is  True
```

We can apply relational operators for str types also.

Eg 2:

```
1) a="  latha"
2) b="  latha"
3)  print("a > b is ",a>b)
4)  print("a >= b is ",a>=b)
5)  print("a < b is ",a<b)
6)  print("a <= b is ",a<=b)
7)
8)  a > b is  False
9)  a >= b is  True
10) a < b is  False
11) a <= b is  True
```

Eg:

```
1)  print(True>True)   False
2)  print(True>=True)   True
3)  print(10 >True) True
4)  print(False > True) False
5)
6) print(10>'latha')
7)      TypeError: '>' not supported between instances of 'int' and 'str'
```

Eg:

```
1)  a=10
2)  b=20
3)  if(a>b):
4)      print("a is greater than b")
5)  else:
6)      print("a is not greater than b")
```

Output: a is not greater than b

Note: Chaining of relational operators is possible. In the chaining, if all comparisons returns True then only result is True. If atleast one comparison returns False then the result is False

```
1)  10<20 → True
2)  10<20<30 → True
3)  10<20<30<40 → True
4)  10<20<30<40>50  → False
```

3) Equality Operators: ==, !=

We can apply these operators for any type even for incompatible types also.

```
1)  >>> 10==20
2)  False
3)  >>> 10!= 20
4)  True
5)  >>> 10==True
6)  False
7)  >>> False==False
8)  True
9) >>> "latha"=="latha"
10) True
11) >>> 10=="latha"
```

```
12)    False
```

Note: **Chaining concept is applicable for equality operators. If atleast one comparison returns False then the result is False. Otherwise the result is True.**

```
1) >>> 10==20==30==40
2) False
3) >>> 10==10==10==10
4) True
```

4) Logical Operators: and, or, not

We can apply for all types.

❖ For boolean Types Behaviour:

and → If both arguments are True then only result is True
or → If atleast one arugemnt is True then result is True
not → Complement

True and False → False
True or False → True
not False → True

❖ For non-boolean Types Behaviour:

0 means False
non-zero means True
empty string is always treated as False

x and y:

If x is evaluates to false return x otherwise return y
Eg:
10 and 20
0 and 20

If first argument is zero then result is zero otherwise result is y

x or y:

If x evaluates to True then result is x otherwise result is y

10 or 20 → 10
0 or 20 → 20

not x:

If x is evalutates to False then result is True otherwise False

not 10 → False
not 0 → True

Eg:

```
1) "latha" and "lathadevi" ==>lathadevi
2) "" and "latha" ==>""
3) "latha" and "" ==>""
4) "" or "latha" ==>"latha"
5) "latha" or ""==>"latha"
6)  not ""==>True
7) not "latha" ==>False
```

5) Bitwise Operators:

- **We can apply these operators bitwise.**
- **These operators are applicable only for int and boolean types.**
- **By mistake if we are trying to apply for any other type then we will get Error.**

- **&, |, ^, ~, <<, >>**

- **print(4&5) → Valid**
- **print(10.5 & 5.6)**
  → TypeError: unsupported operand type(s) for &: 'float' and 'float'

- **print(True & True) → Valid**

- **& → If both bits are 1 then only result is 1 otherwise result is 0**
- **| → If atleast one bit is 1 then result is 1 otherwise result is 0**
- **^ → If bits are different then only result is 1 otherwise result is 0**
- **~ → bitwise complement operator**
- **1 → 0 & 0 → 1**
- **<< → Bitwise Left Shift**
- **>> → Bitwise Right Shift**

- **print(4&5) → 4**
- **print(4|5) → 5**
- **print(4^5) → 1**

Operator	Description
&	If both bits are 1 then only result is 1 otherwise result is 0
|	If atleast one bit is 1 then result is 1 otherwise result is 0
^	If bits are different then only result is 1 otherwise result is 0
~	bitwise complement operator i.e 1 means 0 and 0 means 1
>>	Bitwise Left shift Operator
<<	Bitwise Right shift Operator

Bitwise Complement Operator (~):

We have to apply complement for total bits.

Eg: print(~5)→ -6

Note:

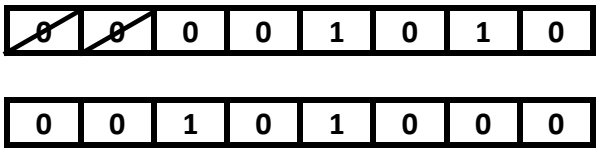
- **The most significant bit acts as sign bit. 0 value represents +ve number where as 1 represents -ve value.**
- **Positive numbers will be repesented directly in the memory where as -ve numbers will be represented indirectly in 2's complement form.**

6) Shift Operators:

<< Left Shift Operator

After shifting the empty cells we have to fill with zero

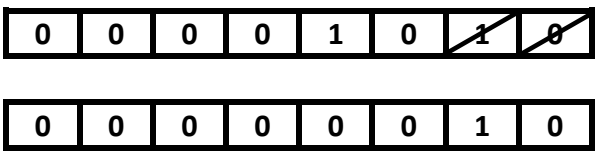
print(10<<2) → 40



>> Right Shift Operator

After shifting the empty cells we have to fill with sign bit.(0 for +ve and 1 for -ve)

print(10>>2) → 2



We can apply bitwise operators for boolean types also

- **print(True & False) → False**
- **print(True | False) → True**
- **print(True ^ False) → True**
- **print(~True) → -2**
- **print(True<<2) → 4**
- **print(True>>2) → 0**

7) Assignment Operators:

- **We can use assignment operator to assign value to the variable.**
  Eg: x = 10

- **We can combine asignment operator with some other operator to form compound assignment operator.**
  Eg: x += 10 → x = x+10

The following is the list of all possible compound assignment operators in Python.

- **+=**
- **-=**
- ***=**
- **/=**
- **%=**
- **//=**
- ****=**
- **&=**
- **|=**
- **^=**
- **>>=**
- **<<=**

Eg:

```
1) x=10
2) x+=20
3) print(x) → 30
```

Eg:

```
1) x=10
2) x&=5
3) print(x) → 0
```

8) Ternary Operator OR Conditional Operator

Syntax: x = firstValue if condition else secondValue

If condition is True then firstValue will be considered else secondValue will be considered.

Eg 1:

```
1) a,b=10,20
2) x=30 if a<b else 40
3) print(x)   #30
```

Eg 2: Read two numbers from the keyboard and print minimum value

```
1) a=int(input("Enter First Number:"))
2) b=int(input("Enter Second Number:"))
3) min=a if a<b else b
4) print("Minimum Value:",min)
```

Output:

```
Enter First Number:10
Enter Second Number:30
Minimum Value: 10
```

Note: Nesting of Ternary Operator is Possible.

Q) Program for Minimum of 3 Numbers

```
1) a=int(input("Enter First Number:"))
2) b=int(input("Enter Second Number:"))
3) c=int(input("Enter Third Number:"))
4) min=a if a<b and a<c else b if b<c else c
5) print("Minimum Value:",min)
```

Q) Program for Maximum of 3 Numbers

```
1) a=int(input("Enter First Number:"))
2) b=int(input("Enter Second Number:"))
3) c=int(input("Enter Third Number:"))
4) max=a if a>b and a>c else b if b>c else c
5) print("Maximum Value:",max)
```

Eg:

```
1) a=int(input("Enter First Number:"))
2) b=int(input("Enter Second Number:"))
3) print("Both numbers are equal" if a==b else "First Number is Less than Second Nu
   mber" if a<b else "First Number Greater than Second Number")
```

Output:

```
D:\python_classes>py test.py
Enter First Number:10
Enter Second Number:10
Both numbers are equal

D:\python_classes>py test.py
Enter First Number:10
Enter Second Number:20
First Number is Less than Second Number

D:\python_classes>py test.py
Enter First Number:20
Enter Second Number:10
First Number Greater than Second Number
```

9) Special Operators:

Python defines the following 2 special operators

1) Identity Operators
2) Membership operators

1) Identity Operators

- We can use identity operators for address comparison.
- There are 2 identity operators are available
  1) is
  2) is not

- r1 is r2 returns True if both r1 and r2 are pointing to the same object.
- r1 is not r2 returns True if both r1 and r2 are not pointing to the same object.

Eg:

```
1) a=10
2) b=10
3) print(a is b)      True
4) x=True
```

```
5) y=True
6) print( x is y)      True
```

Eg:

```
1) a="  latha"
2) b="  latha"
3) print(id(a))
4) print(id(b))
5) print(a is b)
```

Eg:

```
1) list1=["one","two","three"]
2) list2=["one","two","three"]
3) print(id(list1))
4) print(id(list2))
5) print(list1 is list2)   False
6) print(list1 is not list2)   True
7) print(list1 == list2)    True
```

Note: We can use is operator for address comparison where as == operator for content comparison.

2) Membership Operators:

- We can use Membership operators to check whether the given object present in the given collection. (It may be String, List, Set, Tuple OR Dict)
- In → Returns True if the given object present in the specified Collection
- not in → Retruns True if the given object not present in the specified Collection

Eg:

```
1) x="hello learning Python is very easy!!!"
2) print('h' in x)    True
3) print('d' in x)     False
4) print('d' not in x)     True
5) print('Python' in x)    True
```

Eg:

```
1) list1=["sunny","bunny","chinny","pinny"]
2) print("sunny" in list1)  True
3) print("tunny" in list1)  False
4) print("tunny" not in list1)   True
```

Operator Precedence:

If multiple operators present then which operator will be evaluated first is decided by operator precedence.

Eg:
print(3+10*2) → 23
print((3+10)*2) → 26

The following list describes operator precedence in Python

1) **() → Parenthesis**
2) **** → Exponential Operator**
3) **~, - → Bitwise Complement Operator, Unary Minus Operator**
4) ***, /, %, // → Multiplication, Division, Modulo, Floor Division**
5) **+, - → Addition, Subtraction**
6) **<<, >> → Left and Right Shift**
7) **& → Bitwise And**
8) **^ → Bitwise X-OR**
9) **| → Bitwise OR**
10) **>, >=, <, <=, ==, != → Relational OR Comparison Operators**
11) **=, +=, -=, *=... → Assignment Operators**
12) **is , is not → Identity Operators**
13) **in , not in → Membership operators**
14) **not → Logical not**
15) **and → Logical and**
16) **or → Logical or**

```
1)  a=30
2)  b=20
3)  c=10
4)  d=5
5)  print((a+b)*c/d) → 100.0
6)  print((a+b)*(c/d)) → 100.0
7)  print(a+(b*c)/d) → 70.0
8)
9)  3/2*4+3+(10/5)**3-2
10) 3/2*4+3+2.0**3-2
11) 3/2*4+3+8.0-2
12) 1.5*4+3+8.0-2
13) 6.0+3+8.0-2
14) 15.0
```

Mathematical Functions (math Module)

- A Module is collection of functions, variables and classes etc.
- math is a module that contains several functions to perform mathematical operations.
- If we want to use any module in Python, first we have to import that module.
  `import math`
- Once we import a module then we can call any function of that module.

```
1) import math
2) print(math.sqrt(16))
3) print(math.pi)
```

Output

```
4.0
3.141592653589793
```

- We can create alias name by using as keyword. `import math as m`
- Once we create alias name, by using that we can access functions and variables of that module.

```
1) import math as m
2) print(m.sqrt(16))
3) print(m.pi)
```

- We can import a particular member of a module explicitly as follows
  `from math import sqrt`
  `from math import sqrt,pi`

- If we import a member explicitly then it is not required to use module name while accessing.

```
1) from math import sqrt,pi
2) print(sqrt(16))
3) print(pi)
4) print NameError: name (math.pi) 'math' is not defined
```

Important Functions of math Module:

1) ceil(x)
2) floor(x)
3) pow(x,y)
4) factorial(x)
5) trunc(x)
6) gcd(x,y)
7) sin(x)
8) cos(x)
9) tan(x)
10) ....

Important Variables of math Module:

pi3.14
e → 2.71
inf → infinity
nan → not a number

Q) Write a Python Program to find Area of Circle pi*r**2

```
1) from math import pi
2) r = 16
3) print("Area of Circle is :",pi*r**2)
```

Output: Area of Circle is: 804.247719318987

INPUT AND OUTPUT STATEMENTS

Reading Dynamic Input from the Keyboard:

In Python 2 the following 2 functions are available to read dynamic input from the keyboard.

1) **raw_input()**
2) **input()**

1) raw_input():

This function always reads the data from the keyboard in the form of String Format. We have to convert that string type to our required type by using the corresponding type casting methods.

Eg: `x = raw_input("Enter First Number:")`
`print(type(x))` → **It will always print str type only for any input type**

2) input():

input() function can be used to read data directly in our required format.We are not required to perform type casting.

```
x = input("Enter Value)
type(x)
```

10 → int
"latha"-> str
10.5 → float
True → bool

***Note:**

- **But in Python 3 we have only input() method and raw_input() method is not available.**
- **Python3 input() function behaviour exactly same as raw_input() method of Python2. i.e every input value is treated as str type only.**
- **raw_input() function of Python 2 is renamed as input() function in Python 3.**

```
1)  >>> type(input("Enter value:"))
2)  Enter value:10
3)  <class 'str'>
4)
5)  Enter value:10.5
6)  <class 'str'>
7)
```

```
8)  Enter value:True
9)  <class 'str'>
```

Q) Write a program to read 2 numbers from the keyboard and print sum

```
1)  x=input("Enter First Number:")
2)  y=input("Enter Second Number:")
3)  i = int(x)
4)  j = int(y)
5)  print("The Sum:",i+j)
```

```
Enter First Number: 100
Enter Second Number: 200
The Sum: 300
```

```
1)  x=int(input("Enter First Number:"))
2)  y=int(input("Enter Second Number:"))
3)  print("The Sum:",x+y)
```

```
print("The Sum:",int(input("Enter First Number:"))+int(input("Enter Second Number:")))
```

Q) Write a Program to read Employee Data from the Keyboard and print that Data

```
1)  eno=int(input("Enter Employee No:"))
2)  ename=input("Enter Employee Name:")
3)  esal=float(input("Enter Employee Salary:"))
4)  eaddr=input("Enter Employee Address:")
5)  married=bool(input("Employee Married ?[True|False]:"))
6)  print("Please Confirm Information")
7)  print("Employee No :",eno)
8)  print("Employee Name :",ename)
9)  print("Employee Salary :",esal)
10) print("Employee Address :",eaddr)
11) print("Employee Married ? :",married)
```

```
D:\Python_classes>py test.py
Enter Employee No:100
Enter Employee Name:Sunny
Enter Employee Salary:1000
Enter Employee Address:Mumbai
```

Employee Married ?[True|False]:True
Please Confirm Information

Employee No : 100
Employee Name : Sunny
Employee Salary : 1000.0
Employee Address : Mumbai
Employee Married ? : True

How to read multiple values from the keyboard in a single line:

```
1) a,b= [int(x) for x in input("Enter 2 numbers :").split()]
2) print("Product is :", a*b)
```

D:\Python_classes>py test.py
Enter 2 numbers :10 20
Product is : 200

Note: split() function can take space as seperator by default .But we can pass anything as seperator.

Q) Write a program to read 3 float numbers from the keyboard with, seperator and print their sum

```
1) a,b,c= [float(x) for x in input("Enter 3 float numbers :").split(',')]
2) print("The Sum is :", a+b+c)
```

D:\Python_classes>py test.py
Enter 3 float numbers :10.5,20.6,20.1
The Sum is : 51.2

eval():

eval Function take a String and evaluate the Result.

Eg: x = eval("10+20+30")
print(x)
Output: 60

Eg: x = eval(input("Enter Expression"))
Enter Expression: 10+2*3/4
Output: 11.5

eval() can evaluate the Input to list, tuple, set, etc based the provided Input.

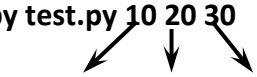
Eg: **Write a Program to accept list from the keynboard on the display**

```
1) l = eval(input("Enter List"))
2) print (type(l))
3) print(l)
```

COMMAND LINE ARGUMENTS

- **argv is not Array it is a List. It is available sys Module.**
- **The Argument which are passing at the time of execution are called Command Line Arguments.**

Eg: **D:\Python_classes py test.py 10 20 30**



Command Line Arguments

Within the Python Program this Command Line Arguments are available in argv. Which is present in SYS Module.

test.py	10	20	30

Note: **argv[0] represents Name of Program. But not first Command Line Argument. argv[1] represent First Command Line Argument.**

Program: **To check type of argv from sys**

```
import argv
print(type(argv))
```

D:\Python_classes\py test.py

Write a Program to display Command Line Arguments

```
1) from sys import argv
2) print("The Number of Command Line Arguments:", len(argv))
3) print("The List of Command Line Arguments:", argv)
4) print("Command Line Arguments one by one:")
5) for x in argv:
6)     print(x)
```

```
D:\Python_classes>py test.py 10 20 30
The Number of Command Line Arguments: 4
The List of Command Line Arguments: ['test.py', '10','20','30']
Command Line Arguments one by one:
test.py
10
20
30
--------------------------
```

```
1) from sys import argv
2) sum=0
3) args=argv[1:]
4) for x in args   :
5)    n=int(x)
6)    sum=sum+n
7) print("The Sum:",sum)
```

```
D:\Python_classes>py test.py 10 20 30 40
The Sum: 100
```

Note 1: Usually space is seperator between command line arguments. If our command line argument itself contains space then we should enclose within double quotes(but not single quotes)

```
1) from sys import argv
2) print(argv[1])
```

```
D:\Python_classes>py test.py Sunny Leone
Sunny

D:\Python_classes>py test.py 'Sunny Leone'
'Sunny

D:\Python_classes>py test.py "Sunny Leone"
Sunny Leone
```

Note 2: Within the Python program command line arguments are available in the String form. Based on our requirement, we can convert into corresponding type by using type casting methods.

```
1) from sys import argv
2) print(argv[1]+argv[2])
3) print(int(argv[1])+int(argv[2]))
```

```
D:\Python_classes>py test.py 10 20
1020
30
```

Note 3: If we are trying to access command line arguments with out of range index then we will get Error.

```
1) from sys import argv
2) print(argv[100])
```

```
D:\Python_classes>py test.py 10 20
IndexError: list index out of range
```

Note: In Python there is argparse module to parse command line arguments and display some help messages whenever end user enters wrong input.

input()
raw_input()

Command Line Arguments

Output Statements:

We can use print() function to display output.

Form-1: print() without any argument
Just it prints new line character

Form-2:

```
1) print(String):
2) print("Hello World")
3) We can use escape characters also
4) print("Hello \n World")
5) print("Hello\tWorld")
6) We can use repetetion operator (*) in the string
7) print(10*"Hello")
8) print("Hello"*10)
9) We can use + operator also
10) print("Hello"+"World")
```

Note:

- If both arguments are String type then + operator acts as concatenation operator.
- If one argument is string type and second is any other type like int then we will get Error.
- If both arguments are number type then + operator acts as arithmetic addition operator.

Note:

```
1) print("Hello"+"World")
2) print("Hello","World")
```

HelloWorld
Hello World

Form-3: print() with variable number of arguments

```
1) a,b,c=10,20,30
2) print("The Values are :",a,b,c)
```

Output: The Values are : 10 20 30

By default output values are seperated by space.If we want we can specify seperator by using "sep" attribute

```
1) a,b,c=10,20,30
2) print(a,b,c,sep=',')
3) print(a,b,c,sep=':')
```

D:\Python_classes>py test.py
10,20,30
10:20:30

Form-4:print() with end attribute

```
1) print("Hello")
2) print("  Latha"  )
3) print("Sister")
```

Output:
Hello
Latha
Sister

If we want output in the same line with space

```
1) print("Hello",end=' ')
2) print("latha",end=' ')
3) print("Sister")
```

Output: Hello Latha Sister

Note: The default value for end attribute is \n, which is nothing but new line character.

Form-5: print(object) statement

We can pass any object (like list, tuple, set etc) as argument to the print() statement.

```
1) l=[10,20,30,40]
2) t=(10,20,30,40)
3) print(l)
4) print(t)
```

Form-6: print(String, variable list)

We can use print() statement with String and any number of arguments.

```
1) s = "   Latha"
2)  a = 48
3)  s1 ="Java"
4)  s2 ="Python"
5)  print("Hello",s,"Your Age is",a)
6)  print("You are teaching",s1,"and",s2)
```

Output:

Hello Latha Your Age is 48
You are teaching java and Python

Form-7: print (formatted string)

1) %i → int
2) %d → int
3) %f → float
4) %s → String type

Syntax: print("formatted string" %(variable list))

Eg 1:

```
1) a=10
2) b=20
3) c=30
4) print("a value is %i" %a)
5) print("b  value is %d and c value is %d" %(b,c))
```

Output

```
a value is 10
b  value is 20 and c value is 30
```

Eg 2:

```
1) s="  Latha"
2) list=[10,20,30,40]
3) print("Hello %s ...The List of Items are %s"  %(s,list))
```

Output: Hello Latha ...The List of Items are [10, 20, 30, 40]

Form-8: print() with replacement operator {}

Eg:

```
1) name = "  Latha"
2) salary = 10000
3) gf = "Sunny"
4) print("Hello {0} your salary is {1} and Your Friend {2} is waiting".
   format(name,salary,gf))
5) print("Hello {x} your salary is {y} and Your Friend {z} is waiting".
   format(x=name,y=salary,z=gf))
```

Output

```
Hello Latha your salary is 10000 and Your Friend Sunny is waiting
Hello Latha your salary is 10000 and Your Friend Sunny is waiting
```