By Sai Kumar

PYTHON 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 operators

5) Assignment operators

6) Special operators

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**

```
a=10
b=2
print('a+b=',a+b)
print('a-b=',a-b)
print('a*b=',a*b)
print('a/b=',a/b)
print('a//b=',a//b)
print('a%b=',a%b)
print('a**b=',a**b)
```

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) >>> **"sai"+10**

TypeError: must be str, not int

3) >>> **"sai"+"10"**

'sai10'

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

- 2*"sai" or "sai"*2

  2.5*"sai" TypeError: can't multiply sequence by non-int of type 'float'

  "sai"*"sai" 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

.....

Relational Operators: >, >=, <, <=

1) **a=10**

2) **b=20**

3) **print("a > b is ",a>b)**

a > b is False

4) **print("a >= b is ",a>=b)**

a >= b is False

5) **print("a < b is ",a<b)**

a < b is True

6) **print("a <= b is ",a<=b)**

a <= b is True

We can apply relational operators for str types also.

Eg 2:

1) **a="sai"**

2) **b="sai"**

3) **print("a > b is ",a>b)**

a > b is False

4) **print("a >= b is ",a>=b)**

a >= b is True

5) **print("a < b is ",a<b)**

a < b is False

6) **print("a <= b is ",a<=b)**

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) **print(10>'sai')**

TypeError: '>' not supported between instances of 'int' and 'str'

Eg:

1) **a=10**

2) **b=20**

3) **if(a>b): print("a is greater than b")**

   else: 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 return 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

Equality Operators: ==, !=

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

1) >>> **10==20**

False

2) >>> **10!= 20**

True

3) >>> **10==True**

False

4) >>> **False==False**

True

5) >>> **"sai"=="sai"**

True

6) >>> **10=="sai"**

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**

False

2) >>> **10==10==10==10**

True

Logical Operators: and, or, not

We can apply for all types.

For Boolean Types Behavior:

and → If both arguments are True, then only result is True

or → If atleast one argument is True, then result is True

not → Complement

True and False → False

True or False → True

not False →True

For non-Boolean Types Behavior:

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 returns 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 evaluates to False then result is True otherwise False

not 10 → False

not 0 → True

Eg:

1) **"sai" and "saisoft" ==>saisoft**

2) **"" and "sai" ==>""**

3) **"sai" and "" ==>""**

4) **"" or "sai" ==>"sai"**

5) **"sai" or ""==>"sai"**

6) **not ""==>True**

7) **not "sai" ==>False**

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
- << → 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

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 assignment 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

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

<u>**Note:**</u> 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 Number" if a<b else "First Number Greater than Second
   Number")
```

Output:

python test.py

```
Enter First Number:10

Enter Second Number:10

Both numbers are equal
```

python test.py

```
Enter First Number:10

Enter Second Number:20

First Number is Less than Second Number
```

python test.py

```
Enter First Number:20

Enter Second Number:10

First Number Greater than Second Number
```

Special Operators:

Python defines the following 2 special operators

1) Identity Operators

2) Membership operators

By Sai Kumar

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="sai"**

2) **b="sai"**

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 whereas == 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)

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

By Sai Kumar

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