

## 4. Operators

### 1. On The Basis Of Operand:

#### Unary Operators

a++, --a, not True

```
In [7]: var = not print("Hello World!")
        print(var)
```

```
Hello World!
True
```

```
In [8]: if not '':
        print("hi")
        else:
            print("ohh easy")
```

```
hi
```

#### Binary Operators

```
In [11]: 4 + 5
```

```
Out[11]: 9
```

```
In [13]: int.__add__(4, 5)
```

```
Out[13]: 9
```

```
In [14]: 'hello ' + 'world'
```

```
Out[14]: 'hello world'
```

```
In [15]: str.__add__('hello ', 'world')
```

```
Out[15]: 'hello world'
```

```
In [16]: r = [1, 2] + [ 3, 4]
        print(r)
```

```
[1, 2, 3, 4]
```

```
In [10]: list.__add__([1,2],[3,4])
```

```
Out[10]: [1, 2, 3, 4]
```

```
In [12]: {'a': 'b'} + {'c': 'd'}  
# there is no __add__ define for dictionary so it will give error
```

```
-----  
TypeError                                Traceback (most recent call last)  
C:\Users\PANKAJ~1\AppData\Local\Temp\ipykernel_12464\2829448156.py in <module>  
----> 1 {'a': 'b'} + {'c': 'd'}  
      2 # there is no __add__ define for dictionary so it will give error  
  
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

## Ternary Operators

Syntax:

x = firstValue if condition else secondValue

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

```
In [60]: a,b = 10,20  
  
r = a if a > b else b  
print(r)
```

20

Note: Nesting of ternary operator is possible

### Q. Program for minimum of 3 numbers

```
In [64]: a = int(input("a :"))  
b = int(input("b :"))  
c = int(input("c :"))  
  
min = a if a<b and a<c else b if b<c else c  
print("minimum : ", min)
```

```
a :12  
b :7  
c :19  
minimum : 7
```

### Q. Program for maximum of 3 numbers

```
In [65]: a=int(input("Enter First Number:"))  
b=int(input("Enter Second Number:"))  
c=int(input("Enter Third Number:"))  
max=a if a>b and a>c else b if b>c else c  
print("Maximum Value:",max)
```

```
Enter First Number:12  
Enter Second Number:56  
Enter Third Number:28  
Maximum Value: 56
```

## 2. Operators in Python

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

# 1. Arithmetic Operators

|    |                           |
|----|---------------------------|
| +  | <code>__add__</code>      |
| -  | <code>__sub__</code>      |
| *  | <code>__mul__</code>      |
| /  | <code>__truediv__</code>  |
| // | <code>__floordiv__</code> |
| %  | <code>__mod__</code>      |
| ** | <code>__pow__</code>      |

In [2]:

```
a = 10
b = 3

print("a+b =", a+b)
print("a-b =", a-b)
print("a*b =", a*b)
print("a/b =", a/b) #true division
print("a//b =", a//b) #floor division
print("a%b =", a%b)
print("a**b =", a**b)
```

```
a+b = 13
a-b = 7
a*b = 30
a/b = 3.3333333333333335
a//b = 3
a%b = 1
a**b = 1000
```

In [ ]:

## 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

In [3]:

```
a = 10.5
b = 2

print(a/b)
print(a//b)
print(a%b)
```

```
5.25
5.0
```

0.5

In [ ]:

## 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.

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

In [39]:

```
'hello ' + 'world'
```

Out[39]:

```
'hello world'
```

In [6]:

```
s = "hi " * 4
s1 = 4*"hi "
print(s)
print(s1)
```

```
hi hi hi hi
hi hi hi hi
```

In [4]:

```
"Pankaj"+10
```

```
-----
TypeError                                Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp\ipykernel_12244\22085048.py in <module>
----> 1 "Pankaj"+10
```

```
TypeError: can only concatenate str (not "int") to str
```

In [7]:

```
2.5*"pankaj"
```

```
-----
TypeError                                Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp\ipykernel_12244\1996987308.py in <module>
----> 1 2.5*"pankaj"
```

```
TypeError: can't multiply sequence by non-int of type 'float'
```

In [8]:

```
"pankaj"*"pankaj"
```

```
-----
TypeError                                Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp\ipykernel_12244\3212703015.py in <module>
----> 1 "pankaj"*"pankaj"
```

```
TypeError: can't multiply sequence by non-int of type 'str'
```

## Note:

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

In [9]: `10/0`

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
C:\Users\PANKAJ~1\AppData\Local\Temp\ipykernel_12244\530406163.py in <module>  
----> 1 10/0
```

**ZeroDivisionError:** division by zero

## Uses of + and \* with other data types

In [10]: `data = [ 0 ] * 10`  
`print(data)`  
  
`[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

In [ ]:

In [18]: `data = [ 'hello', 'hi' ] * 3`

In [46]: `print(data)`  
  
`['hello', 'hi', 'hello', 'hi', 'hello', 'hi']`

In [ ]:

In [40]: `[1,2 , 3] + [ 4, 5]`

Out[40]: `[1, 2, 3, 4, 5]`

In [ ]:

In [41]: `(1, 2, 3) + (1, 2, 3)`

Out[41]: `(1, 2, 3, 1, 2, 3)`

In [ ]:

In [19]: `lst = [ [] ] * 5`  
`print(id(lst[0]))`  
`print(id(lst[1]))`  
`print(id(lst[2]))`  
`lst[3].append('awesome')`

`2304284932288`  
`2304284932288`  
`2304284932288`

In [20]: `print(lst)`

```
[['awesome'], ['awesome'], ['awesome'], ['awesome'], ['awesome']]
```

In [ ]:

In [19]:

```
lst = [ [] for _ in range(5)]

print(id(lst[0]))
print(id(lst[1]))
print(id(lst[2]))
```

```
1757213671488
1757213671744
1757213673024
```

In [20]:

```
print(lst)
lst[3].append('awesome')
print(lst)
```

```
[[], [], [], [], []]
[[], [], [], ['awesome'], []]
```

In [ ]:

## 2. Relational Operators:

|    |        |
|----|--------|
| <  | __lt__ |
| >  | __gt__ |
| <= | __le__ |
| >= | __ge__ |
| == | __eq__ |
| != | __ne__ |

In [1]:

```
a = 10
b = 20

print("a > b is ",a>b)
print("a >= b is ",a>=b)
print("a < b is",a<b)
print("a <= b is",a<=b)
```

```
a > b is False
a >= b is False
a < b is True
a <= b is True
```

**We can apply relational operators for str types also**

In [2]:

```
a = "pankaj"
b = "kumar"

print("a > b is",a>b)
print("a >= b is",a>=b)
print("a < b is",a<b)
print("a <= b is",a<=b)
```

```
a > b is True
a >= b is True
```

```
a < b is False
a <= b is False
```

In [ ]:

In [3]:

```
print(True>True)
print(True>=True)
print(10>True)
print(False>True)
```

```
False
True
True
False
```

In [4]:

```
print(10>"pankaj")
```

```
-----
TypeError                                Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp\ipykernel_11052\1070455721.py in <module>
----> 1 print(10>"pankaj")
```

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

In [5]:

```
a=10
b=20

if a>b:
    print("a is greater than b")
else:
    print("b is greater than a")
```

```
b is greater than a
```

## 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

In [6]:

```
10<20
```

Out[6]:

```
True
```

In [7]:

```
10<20<30
```

Out[7]:

```
True
```

In [8]:

```
10<20<30<40
```

Out[8]:

```
True
```

In [9]:

```
10<20<30<40>50
```

Out[9]:

```
False
```

## equality operators:

`==, !=`

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

```
In [14]: print(10 == 20) #False
print(10 != 20) #True
print(10==True) #False
print(False==False) #True
print(10=="Pankaj") #False
print("Pankaj" == "Pankaj") # rue
```

False

True

False

True

False

True

### Note:

Chaining concept is applicable for equality operators. If atleast one comparison returns False then the result is False. otherwise the result is True.

```
In [15]: 10==20==30==40
```

False

Out[15]:

```
In [16]: 10==10==10==10
```

True

Out[16]:

### NOte

```
In [17]: lst = [ 1, 2, 3, 4]
lst2 = [ 1, 2, 2, 3]

r = lst < lst2
print(r)
```

False

## 3. 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

```
In [30]: "pankaj"and"pankajkumar"
```

```
Out[30]: 'pankajkumar'
```

```
In [31]: "" and "pankaj"
```

```
Out[31]: ''
```

```
In [32]: "pankaj" and ""
```

```
Out[32]: ''
```

```
In [33]: "" or "pankaj"
```

```
Out[33]: 'pankaj'
```

```
In [34]: "pankaj" or ""
```

```
Out[34]: 'pankaj'
```

```
In [35]: not ""
```

```
Out[35]: True
```

```
In [36]: not "pankaj"
```

```
Out[36]: False
```

## x and y:

if x is evaluates to false return x otherwise return y

```
In [21]: 10 and 20
```

```
Out[21]: 20
```

```
In [24]: 0 and 20
```

```
# if first argument is zero then result is zero  
# otherwise result is y
```

```
Out[24]: 0
```

## x or y:

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

```
In [25]: 10 or 20
```

```
Out[25]: 10
```

```
In [26]: 0 or 20
```

```
Out[26]: 20
```

**not x:**

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

```
In [27]: not 10
```

```
Out[27]: False
```

```
In [28]: not 0
```

```
Out[28]: True
```

## 4. Bitwise Operator

&, |, ^, ~, <<, >>

We can apply these operator bitwise.

These operator are applicable only for int and boolean types

By mistake if we are trying to apply for another type then we will get error

```
In [2]: from IPython.display import Image  
Image(filename='bitwis.jpg')
```

```
Out[2]:
```

| 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                                      |

```
In [37]: print(4 & 5)
```

```
4
```

```
In [54]: print(4|5)
```

```
5
```

```
In [55]: print(4^5)
```

1

```
In [38]: print(10.5 & 5.6)
```

```
-----  
TypeError                                Traceback (most recent call last)  
C:\Users\PANKAJ~1\AppData\Local\Temp\ipykernel_11052\2499797966.py in <module>  
----> 1 print(10.5 & 5.6)
```

**TypeError:** unsupported operand type(s) for &: 'float' and 'float'

## bitwise complement operator(~):

We have to apply complement for total bits

```
In [42]: print(~5)  
  
# 5 -> 000000000000000000000000...101  
# ~5 -> 111111111111111111111111...010  
# 1's of (~5) -> 000000000000000000000000...101  
# 2's of (~5) -> 1's + 1  
# i.e -> 000000000000000000000000...110 = -6
```

-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 represented directly in the memory where as -ve numbers will be represented indirectly in 2's complement form.

```
In [44]: ~ True #bcz(True = 1)
```

-2

Out[44]:

## Shift Operators:

<< left shift operators

After shifting the empty cells we have to fill with zero

shifting the cells in left

```
In [46]: print(10<<1) #10*2^1
```

20

```
In [48]: print(10<<2) #10*2^2
```

40

```
In [4]: Image(filename='left shift.png')
```

Out[4]:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |

>> Right Shift operator

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

shifting the cells in right

In [52]:

```
print(10>>1)
```

5

In [51]:

```
print(10>>2) # 10 // 2^2
```

2

In [5]:

```
Image(filename='right shift.png')
```

Out[5]:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

## 5. Assignment Operators:

We can use assignment operator to assign value to the variable.

Ex:- x = 10

We can combine assignment operator with some other operator to form compound assignment operator.

Ex:- x += 10 ==> x = x+10

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

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

In [56]:

```
x =10
print(x)

x += 20
print(x)
```

10  
30

In [58]:

```
x = 10
x &= 5
print(x)
```

0

# 6. 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.

Two 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

```
In [66]: a = 10
b = 10
print(a is b) # True
```

True

```
In [67]: x = True
y = True
print(x is y) # True
```

True

```
In [68]: a = "pankaj"
b = "pankaj"
print(a is b)
```

True

```
In [69]: l1 = ["one", "two", "three"]
l2 = ["one", "two", "three"]

print(id(l1))
print(id(l2))

print(l1 is l2)
print(l1 is not l2)
print(l1 == l2)
```

1757245999296

1757245999360

False

True

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

```
In [71]: x="hello learning Python is very easy!!!"

print('h' in x) #True
print('d' in x) #False
print('d' not in x) #True
print('Python' in x) #True
```

```
True
False
True
True
```

```
In [73]: list1=["sunny","bunny","chinny","pinny"]

print("sunny" in list1) #True
print("tunny" in list1) #False
print("tunny" not in list1) #True
```

```
True
False
True
```

## 3. Operators Precedence

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

The following list describes operator precedence in Python

|                      |   |
|----------------------|---|
| ()                   | ->Parenthesis                                       |
| **                   | ->exponential operator                              |
| ~, -                 | ->Bitwise complement operator, unary minus operator |
| *, /, %, //          | ->multiplication, division, modulo, floor division  |
| +, -                 | ->addition, subtraction                             |
| <<, >>               | ->Left and Right Shift                              |
| &                    | ->bitwise And                                       |
| ^                    | ->Bitwise X-OR                                      |
|                      | ->Bitwise OR  |
| >, >=, <, <=, ==, != | ->Relational or Comparison operators                |
| =, +=, -=, *=...     | ->Assignment operators                              |
| is, is not           | ->Identity Operators                                |
| in, not in           | ->Membership operators                              |
| not                  | ->Logical not                                       |
| and                  | ->Logical and                                       |
| or                   | ->Logical or  |

```
In [74]: a = 30
          b = 20
          c = 10
          d = 5

          print((a+b)*c/d) # 100.0
          print((a+b)*(c/d)) #100.0
          print(a+(b*c)/d) #100.0
```

100.0  
100.0  
70.0

```
In [81]: print(3/2*4+3+(10/5)**3-2)

          # 3/2*4+3+(10/5)**3-2
          # 3/2*4+3+2.0**3-2
          # 3/2*4+3+8.0-2
          # 1.5*4+3+8.0-2
          # 6.0+3+8.0-2
          # 15.0
```

15.0

## 4. 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.

Once we import a module then we can call any function of that module.

```
In [82]: import math
          print(math.sqrt(16))
          print(math.pi)
```

4.0  
3.141592653589793

We can create alias name by using as keyword.

Once we create alias name, by using that we can access functions and variables of that module

```
In [83]: import math as m
          print(m.sqrt(16))
          print(m.pi)
```

4.0  
3.141592653589793

We can import a particular member of a module explicitly as follows

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

```
In [2]: from math import sqrt, pi
        print(sqrt(16))
        print(pi)
```

```
4.0
3.141592653589793
```

```
In [4]: print(math.pi) #since math is not import here only sqrt and pi are import
```

```
-----
NameError                                Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp\ipykernel_992\1697151955.py in <module>
----> 1 print(math.pi) #since math is not import here only sqrt and pi are import

NameError: name 'math' is not defined
```

important functions of math module:

```
ceil(x)
floor(x)
pow(x,y)
factorial(x)
trunc(x)
gcd(x,y)
sin(x)
cos(x)
tan(x)
```

important variables of math module:

```
pi      3.14
e       2.71
inf     infinity
nan     not a number
```

Q. Write a Python program to find area of circle

```
In [5]: from math import pi
        r=16
        print("Area of Circle is :",pi*r**2)
```

```
Area of Circle is : 804.247719318987
```

## 5. divmod()

```
In [37]: num = 12345
         q, r = divmod(num, 10)
         print(q)
         print(r)
```

```
1234
5
```

## 6. all , any Operator

```
In [7]:
```



```
cond = [ 4 > 5, 6 < 8, 8>7]
print(cond)
```

```
[False, True, True]
```

```
In [8]: all(cond) # all == and
```

```
Out[8]: False
```

```
In [9]: any(cond) # any == or
```

```
Out[9]: True
```

```
In [10]: op = all([True, True, True, ' ', [''], 1])
print(op)
```

```
True
```

```
In [11]: op = all([True, True, True, '', print('hi'), [''], 1])
print(op)
```

```
hi
False
```

```
In [ ]:
```