Operators in Python

There are 7 types of operators in python

1. Arithmetic Operators
2. Assignment Operataors
3. Comparison Operators
4. Bitwise Operators
5. Logical Operators
6. Membership Operators
7. identity Operators

1. Arithmetic Operators

There are 7 types of **arithmetic operators.**

- Addition
- Subtraction
- Multiplication
- Division
- Modulo
- Floor Division
- Exponential or Power

1.1 Addition

This operator add two or more numbers. It is represented by + symbol.

In [2]:
```
2 + 5
# Here the numbers 2 and 5 are called as operands and + is called operators. Since
# that is why the sign of the final sum is positive.
```
Out[2]:
```
7
```

In [3]:
```
-5 + 8    #----> Here the greater number is 8 whose sign is positive that is why tl
```
Out[3]:
```
3
```

In [4]:
```
-8 + (-7)   #----> Here both the numbers have same negative sign that is why the s
```
Out[4]:
```
-15
```

In [216…
```
5 + 2.5     #----> Here the sum of integer and float always gives a float value.
```
Out[216]:
```
7.5
```

In [6]:
```
5.6 + 2.5        #Float addition
```

```
Out[6]: 8.1

In [63]: 8.2 + (-1.6)      #Float addition

Out[63]: 6.6

In [75]: 5 + 6 + 8 + 15        #Integer addition with more than 2 numbers

Out[75]: 34

In [74]: 2.3 + 6.1 + 5.1 + 7.2         #Float addition with more than 2 numbers

Out[74]: 20.7
```

1.2 Subtraction

This operator subtract two or more numbers. It is represented by - symbol.

```
In [12]: 5 - 2        #Integer subtraction

Out[12]: 3

In [13]: -2 - 5        #Integer subtraction

Out[13]: -7

In [14]: 2.3 - 5.6        #Float subtraction

Out[14]: -3.3

In [48]: 28.8 - 14.9        #Float subtraction

Out[48]: 13.9

In [76]: 15 - 6 - 8 - 5         #Integer subtraction with more than 2 numbers

Out[76]: -4

In [96]: 12.3 - 6.1 - 5.1 - 4.9     #Float subtraction with more than 2 numbers

Out[96]: -3.7999999999999999
```

1.3 Multiplication

This operator multiply two or more numbers. It is represented by * symbol.

```
In [65]: 2 * 5         #Integer multiplication

Out[65]: 10

In [66]: 85 * 65        #Integer multiplication

Out[66]: 5525

In [67]: 45.3 * 86.5        #float multiplication
```

```
Out[67]: 3918.45

In [69]: -45.8 * 45.14         #float multiplication (Only one number has negative sign that

Out[69]: -2067.412

In [97]: 85 * 65 * 56         #Integer multiplication with more than 2 numbers

Out[97]: 309400

In [99]: -45.8 * 45.14 * 32.2       #float multiplication with more than 2 numbers

Out[99]: -66570.6664
```

1.4 Division

This operator divide two numbers. It gives **float quotient** when two number are divided. It is represented by / symbol.

NOTE: Here the quotient is always float (must have decimal point). It doesn't matter whether we divide integer to integer or integer to float.

```
In [100…  4/2        #Integer division----------> see the quotient is a float value instead 

Out[100]: 2.0

In [102…  5/4        #Integer division

Out[102]: 1.25

In [130…  -5/4      #---> Here the numerator has negative sign that is why the sign of the re

Out[130]: -1.25

In [131…  5/-4      #---> Here the denominator has negative sign that is why the sign of the 

Out[131]: -1.25

In [132…  -5/-4     #---> Here both numerator and denominator have negative sign that is why

Out[132]: 1.25

In [106…  25.65/1.25        #Float division

Out[106]: 20.52

In [113…  1245/25.6      #integer float division

Out[113]: 48.6328125
```

1.5 Modulo

This operator gives the remainder when two numbers are divided. It is represented by % symbol.

NOTE: When a number is evenly divisible by another number then the remainder is always 0.

```
In [114…  4%2
Out[114]: 0
```

```
In [115…  5%2
Out[115]: 1
```

Note:

1. The sign of the remainder is same as the sign of the **denominator** not **numerator**.
2. if the sign of the numerator and denominator is **not the same**, then the modulo will be obtained by subtracting the numerator from the next multiple of the denominator.

```
In [178…  11%-5
          # Here you can see clearly that the sign of 11 and -5 are different. The sign of de
          # the sign of the remainder must be negative. In this case the next multiple of 5 w
          # that is why -4 is the answer instead of -1.
Out[178]: -4
```

```
In [153…  -11%5
          # Here you can see clearly that the sign of 11 and -5 are different. The sign of de
          # the sign of the remainder must be positive. In this case the next multiple of 5 w
          # that is why 4 is the answer instead of 1.
Out[153]: 4
```

```
In [177…  -11%-5
          # Here you can see clearly that the sign of -11 and -5 are same. The sign of denom
          # the sign of the remainder must be negative. Here both numerator and denominator 
          # will be -1.
Out[177]: -1
```

```
In [179…  -22%7   #---------> Try to understant the logic here by yourself why the answer is
Out[179]: 6
```

```
In [122…  19%7
Out[122]: 5
```

1.6 Floor Division

This operator divide two numbers. It gives **Integer quotient** when two number are divided. It is represented by **//** symbol.

NOTE: Here the quotient is always integer if both the numbers are integer. If any number is negative (whether the denominator or numerator), then the quotient will float rather than integer.

```
In [181…  4//2
Out[181]: 2

In [182…  5//2  #----> Here the real quotient should be 2.5 but since we use floor division,
Out[182]: 2

In [184…  8//3  #----> Here the real quotient should be 2.6667 but the answer is 2 (Integer)
Out[184]: 2

In [188…  8.5 //3.2     #----> Here the quotient is float although we used floor division.
Out[188]: 2.0
```

1.7 Exponential or power

This operator basically evaluates the exponent of a number for the given power. It is represented by ** symbol.

```
In [190…  2**3
Out[190]: 8

In [191…  5**5
Out[191]: 3125

In [197…  (-5)**4    #-------> Here the base is (-5) and the power is even that is why the re
Out[197]: 625

In [199…  -5**4    #-------> Here the base is 5 not (-5). This means the expression is (-1)*
Out[199]: -625

In [ ]:   (-5)**5   #-------> Here the base is (-5) and the power is odd that is why the resu

In [194…  5**(-2)    #-------> Here 5**(-2) means 5^(-2)= 1/(5^2)=1/25 that is why the answe
Out[194]: 0.04
```

Note: Float arithmetic operation gives approximate value which is depends upon the system bit (32 bit or 64 bit). That is why please take care while performing float operation. If you want to perform operation with precision, then you can Decimal module.

2. Assignment Operators

There are 8 types of **assignment operators.**

- Assignment
- Addition Assignment

- Subtraction Assignment
- Multiplication Assignment
- Division Assignment
- Modulo Assignment
- Floor Division Assignment
- Exponential or Power Assignment

2.1 Assignment

This operator basically assign the value to a variable. It is represented by = symbol.

In [200…

```
# Here a is the variable and we are assigning 5 to the variable a.
a = 5
print(a)
```

```
5
```

In [202…

```
# We can reassign another value to the same variable which is already defined.
a = 8
print(a)
```

```
8
```

In [206…

```
x, y, z = 10, 20, 30
# We can assign more than 1 variable in a single line. Please not both side must ho
# are three variables which means on the right side there must be three values else
print(x)
print(y)
print(z)
```

```
10
20
30
```

In [207…

```
p = 4
q = p
print(p)
print(q)
# Here we are assigning the value of variable p to another variable q.
```

```
4
4
```

In [209…

```
str = "hello"
print(str)
# Here we are assigning string to the variable str.
```

```
hello
```

In [210…

```
boolean = True
print(boolean)
# Here we are assigning boolean datatype to the variable boolean.
```

```
True
```

2.2 Addition Assignment

This operator first add the value and then assign the increment value to the variable. It is represented by += symbol.

In [211…

```
a=5       #----> Here we first assign 5 to the variable 5.
a+=4       #----> Next we first add 4 to the initial value of a which was 5 then as
print(a)
```

```
9
```

Note: The + symbol must comes before the = symbol

In [212…

```
a=+5    #----> This is not an addition assignment operator, this will works a simp
print(a)
```

2.3 Subtraction Assignment

This operator first subtract the value and then assign the decrement value to the variable. It is represented by -= symbol.

In [218…

```
a=5       #----> Here we first assign 5 to the variable 5.
a-=4    #----> Next we first subtract 4 to the initial value of a which was 5 then
print(a)
```

```
1
```

In [220…

```
a=-5    #----> Simple assignment operators, we are assigning -5 to a.
print(a)
```

```
-5
```

Note: The - symbol must comes before the = symbol

2.4 Multiplication Assignment

This operator first multiply the value and then assign the result to the variable. It is represented by *= symbol.

In [222…

```
a=5       #----> Here we first assign 5 to the variable 5.
a*=4   #----> Next we first multiply 4 to the initial value of a which was 5 then
print(a)
```

```
20
```

In [224…

```
a=*5    #----> It will raise an error as *  is an operator and 5 is an integer we
print(a)
```

```
  Cell In[224], line 1
    a=*5    #----> It will raise an error as *  is an operator and 5 is an integer
we can't assign both different things to a.
      ^
SyntaxError: can't use starred expression here
```

Note: The * symbol must comes before the = symbol

2.5 Division Assignment

This operator first divide the value and then assign the result to the variable. It is represented by /= symbol.

Note: This will always gives float value.

In [226…
```
a=5       #----> Here we first assign 5 to the variable 5.
a/=5    #----> Next we first divide the initial value of a which was 5 by 5 then as
print(a)
```
```
1.0
```

In [227…
```
a=/5      #----> It will raise an error.
print(a)
```
```
  Cell In[227], line 1
    a=/5
      ^
SyntaxError: invalid syntax
```

Note: The / symbol must comes before the = symbol

2.6 Modulo Assignment

This operator first find the remainder then assign the result to the variable. It is represented by %= symbol.

In [228…
```
a=5       #----> Here we first assign 5 to the variable 5.
a%=2    #----> Next it find the remainder which is 1 as 5%2=1, so the final value c
print(a)
```
```
1
```

2.7 Floor Division Assignment

This operator first divide the value and then assign the result to the variable. It is represented by //= symbol.

Note: This will always gives integer value until float value is given.

In [229…
```
a=5       #----> Here we first assign 5 to the variable 5.
a//=2   #----> Next we first divide the initial value of a which was 5 by 2 then as
print(a)
```
```
2
```

In [230…
```
a=5       #----> Here we first assign 5 to the variable 5.
a//=2.0  #Next we first divide the initial value of a which was 5 by 2.0 then assig
print(a)   # Here the result is float rather than integer.
```
```
2.0
```

2.8 Exponential or power Assignment

This operator first evaluate the power and then assign the result to the variable. It is represented by **= symbol.

In [231…
```
a=5       #----> Here we first assign 5 to the variable 5.
a**=2   #----> Next the value of 5^2 is calculated, 25 comes out which then assigne
print(a)
```
```
25
```

3. Comparison Operators

There are 6 types of **comparison operators.**

- Equal
- Not equal
- Less than
- Greater than
- Less than or equal
- Greater than or equal

  Note: This operators basically compare the condition, expression, values and returns only boolean value that is either True or False.

3.1 Equal

This operator return **True** if both side of the **equal to** symbol is same **(value and datatype)** else **False**. It is represented by == symbol.

In [233…
```
a=5
b=4
a==b
# Here the value of a and b are not same that is why the result comes out False wh
```
Out[233]:
```
False
```

In [234…
```
a=25
b="25"
a==b
# Here the datatype of a and b are not same that is why the result comes out False
```
Out[234]:
```
False
```

In [236…
```
a=5
b=2
a+b == b+a
# Here both side expression compares which is equal that is why the result comes ou
```
Out[236]:
```
True
```

3.2 Not Equal

This operator return **False** if both side of the **equal to** symbol is same **(value and datatype)** else **True**. It is represented by != symbol. It is basically opposite of equal operator.

In [237…
```
a=5
b=2
a!=b
# Here both the value are different that is why the result comes out True if both i
# be False.
```
Out[237]:
```
True
```

In [244…
```
a=5
b=2
a+b != b+a
# Here both side expression compares which is equal that is why the result comes ou
```

Out[244]:
```
True
```

3.3 Less than

This operator return **True** if the left side of the < symbol is **less than** the right side else return **False**. It is represented by < symbol.

Note: The datatype of both side must be same else it will raise an error.

In [240…
```
a=5
b=2
a < b     #----> Here the value of a is greater than b that is why the result come
```

Out[240]:
```
False
```

In [241…
```
b < a
```

Out[241]:
```
True
```

3.4 Greater than

This operator return **True** if the left side of the > symbol is **greater than** the right side else return **False**. It is represented by > symbol.

Note: The datatype of both side must be same else it will raise an error.

In [245…
```
a=5
b=2
a > b     #----> Here the value of a is greater than b that is why the result come
```

Out[245]:
```
True
```

In [246…
```
b > a
```

Out[246]:
```
False
```

3.5 Less than or equal to

This operator return **True** if the left side of the <= symbol is **less than or equal to** the right side else return **False**. It is represented by <= symbol.

Note: The datatype of both side must be same else it will raise an error.

In [249…
```
a=5
b=2
a <= b     #----> Here the value of a is greater than b that is why the result com
```

Out[249]:
```
False
```

In [250…
```
a=5
b=5
a <= b     #----> Here the value of a is equal to b that is why the result comes o
```

Out[250]:
```
True
```

In [251…
```
a=2
b=5
a <= b     #----> Here the value of a is less than b that is why the result comes o
```
Out[251]:
```
True
```

In [252…
```
a=2
b=5
a =< b
```
```
  Cell In[252], line 3
    a =< b     #----> Here the value of a is less than b that is why the result co
mes out True.
       ^
SyntaxError: invalid syntax
```

Note: Please take care of the symbol <=, the symbol < must comes first and then = otherwise it will raise an error.

3.6 Greater than or equal to

This operator return **True** if the left side of the <= symbol is greater than or equal to the right side else return False. It is represented by >= symbol.

Note: The datatype of both side must be same else it will raise an error.

In [253…
```
a=5
b=2
a >= b     #----> Here the value of a is greater than b that is why the result come
```
Out[253]:
```
True
```

In [254…
```
a=5
b=5
a >= b     #----> Here the value of a is equal to b that is why the result comes ou
```
Out[254]:
```
True
```

In [255…
```
a=2
b=5
a >= b     #----> Here the value of a is less than b that is why the result comes o
```
Out[255]:
```
False
```

Note: Please take care of the symbol >=, the symbol > must comes first and then = otherwise it will raise an error.

4. Bitwise Operators

There are 6 types of **bitwise operators.**

- Binary AND (&)
- Binary OR (|)
- Binary XOR (^)
- Binary Ones compliment (~)

- Binary Left Shift (<<)
- Binary Right Shift (>>)

  Note: This operators works on the basis of binary. If we use these operator, then the operands will be considered as binary instead of normal numbers.

In [257…

```
# Consider the values are:-
a=60
b=13
# The binary of a (which is equal to 60) will be 00111100 and the binary of b (whic
```

4.1 Binary AND

This operator works on bits and perform bit by bit operation using the binary AND operator.

Rules for AND operator in binary.

1 & 1 = 1

1 & 0 = 0

0 & 1 = 0

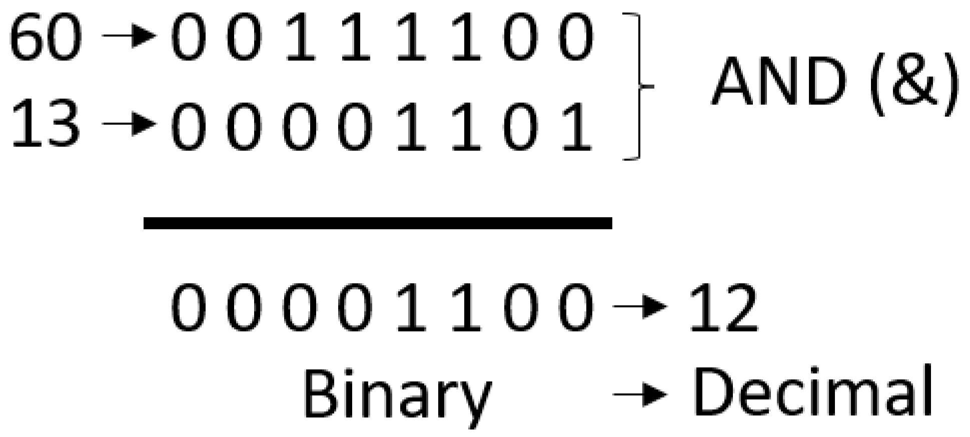
0 & 0 = 0

In [263…

```
a&b
# Here first a and b converted to binary and then AND operation performed on binary
# which then converted back to decimal number.
# The binary conversion of 12 is 00001100.
```

Out[263]:

```
12
```



4.2 Binary OR

This operator works on bits and perform bit by bit operation using the binary OR operator.

Rules for OR operator in binary.

1 | 1 = 1

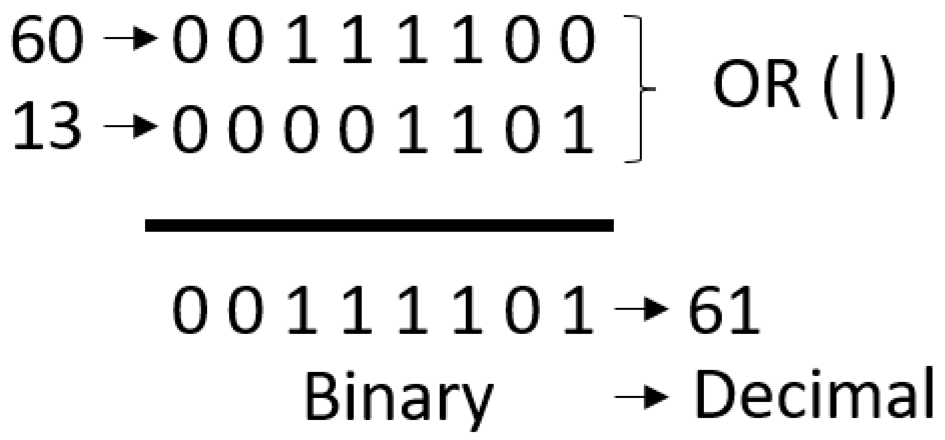
1 | 0 = 1

0 | 1 = 1

0 | 0 = 0

In [264…
```
a|b
# Here first a and b converted to binary and then OR operation performed on binary
# which then converted back to decimal number.
# The binary conversion of 61 is 00111101.
```

Out[264]:
```
61
```



4.3 Binary XOR

This operator works on bits and perform bit by bit operation using the binary XOR operator.

Rules for XOR operator in binary.

1 ^ 1 = 0

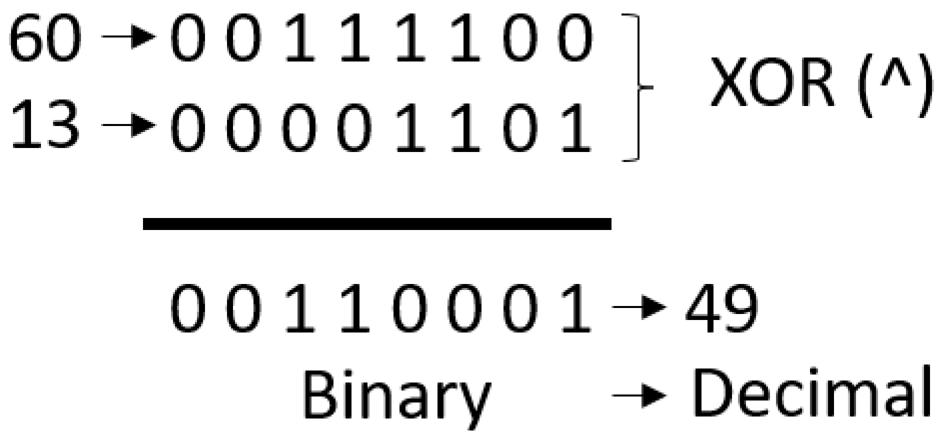
1 ^ 0 = 1

0 ^ 1 = 1

0 ^ 0 = 0

In [265…
```
a^b
# Here first a and b converted to binary and then XOR operation performed on binar
# which then converted back to decimal number.
# The binary conversion of 49 is 00110001.
```

Out[265]:
```
49
```



4.4 Binary Ones Compliment

This operator convert 0 to 1 and 1 to 0.

Rules for Compliment operator in binary.

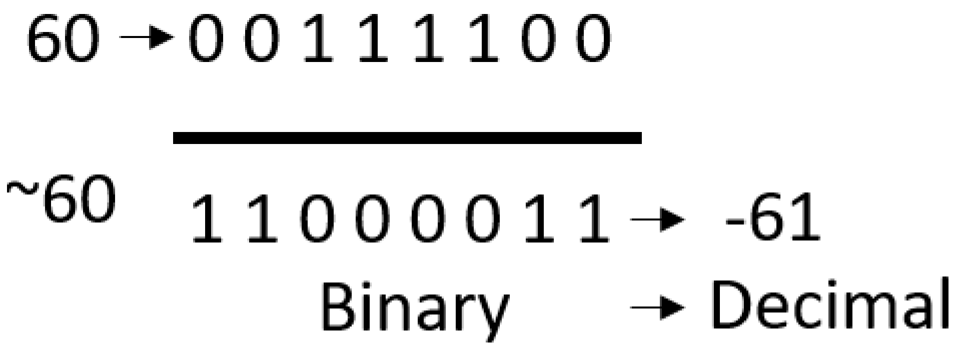
~ 1 = 0

~ 0 = 1

Notes: Its an unary operator that works on a single variable.

In [266…
```
~a
# Here first a converted to binary and compliment operation performed on binary nur
# which then converted back to decimal number.
# The binary conversion of -61 is 11000011.
```

Out[266]:
```
-61
```



In [267…
```
~b
# The binary conversion of -14 is 11110010.
```

Out[267]:
```
-14
```

4.5 Binary Left Shift

Python bitwise left shift operator shifts the left operand bits towards the left side for the given number of times in the right operand. In simple terms, the binary number is appended with 0s at the end.

In [268…

```
a=60
a<<2
#  Here first the variable a converted to binary. The binary conversion of 60 is 11
# After applying left shift operator, two more 0's are added at the extreme right o
# The resut comes out 11110000 which is equal to 240.
```

Out[268]:

```
240
```

In [273…

```
a<<3
#  Here first the variable a converted to binary. The binary conversion of 60 is 11
# After applying left shift operator, three more 0's are added at the extreme right
# The resut comes out 111100000 which is equal to 480.
```

Out[273]:

```
480
```

4.6 Binary Right Shift

Python right shift operator is exactly the opposite of the left shift operator. Then left side operand bits are moved towards the right side for the given number of times. In simple terms, the right side bits are removed.

In [276…

```
a=60
a>>2
# Here first the variable a converted to binary. The binary conversion of 60 is 11
# After applying right shift operator, two bits are removed from the extreme right
# The resut comes out 1111 which is equal to 15.
```

Out[276]:

```
15
```

In [277…

```
a=60
a>>3
# Here first the variable a converted to binary. The binary conversion of 60 is 11
# After applying right shift operator, three bits are removed from the extreme righ
# The resut comes out 111 which is equal to 7.
```

Out[277]:

```
7
```

5. Logical Operators

There are 3 types of **logical operators.**

- Logical AND
- Logical OR
- Logical NOT

  Note: This operators basically merge two or more condition and gives the boolean result that is either True or False.

5.1 Logical AND

This operator generate the combined output of two or more test condition using the AND logical operator.

True and True ---> True

True and False ---> False

False and True ---> False

False and False ---> False

Note: The combined result is only True when all the conditions are True else the result will be False. This operator basicaly works with comparison operators or to merge two or more test condition.

In [291…
```
a=5
b=3
a > b and a == b
# Here the condition a > b is True whereas the condition a == b is False. Therefore
```
Out[291]:
```
False
```

In [294…
```
a="Hello"
b="hello"
c="hello"
a == b and b == c
# Here a is not equal to which means the condition a == b will give False as an out
# condition b == c will give True as an output. Since False and True gives False as
```
Out[294]:
```
False
```

In [283…
```
True and True
```
Out[283]:
```
True
```

In [284…
```
True and False
```
Out[284]:
```
False
```

Note: Python is a **case sensitive** language so take care of the case of the letters in the variable, operators, expression. The logical "and" must be in lower case not in upper case else it will not gonna work.

5.2 Logical OR

This operator generate the combined output of two or more test condition using the OR logical operator.

True and True ---> True

True and False ---> True

False and True ---> True

False and False ---> False

Note: The combined result is only False when all the conditions are False else the result will be True. This operator basicaly works with comparison operators or to merge two or more test condition.

In [295…
```
a=5
b=3
a > b and a == b
# Here the condition a > b is True whereas the condition a == b is False. Therefore
```
Out[295]:
```
True
```

In [296…
```
a="Hello"
b="hello"
c="hello"
a == b or b == c
# Here a is not equal to which means the condition a == b will give False as an out
# condition b == c will give True as an output. Since False and True gives True as
```
Out[296]:
```
True
```

In [297…
```
True or True
```
Out[297]:
```
True
```

In [298…
```
False or False
```
Out[298]:
```
False
```

In [299…
```
True or False
```
Out[299]:
```
True
```

Note: Python is a **case sensitive** language so take care of the case of the letters in the variable, operators, expression. The logical "or" must be in lower case not in upper case else it will not gonna work.

5.3 Logical NOT

This operator generate the compiment of any test condition result. That is if the result is True, the using this operator, the result become False and vice versa.

NOT True ---> False

Not False ---> True

In [301…
```
a = 5
b = 3
not a > b
# Here the conition a>b is True as a is greater than b, but when "not" operator is
```
Out[301]:
```
False
```

In [302…
```
not True
```
Out[302]:
```
False
```

In [303…
```
not False
```
Out[303]:
```
True
```

Note: Python is a **case sensitive** language so take care of the case of the letters in the variable, operators, expression. The logical "not" must be in lower case not in upper case else it will not gonna work.

6. Membership Operators

There are 2 types of **membersship operators.**

- in
- not in

  Note: This operators also return either True or False depending upon the condition. This operator is used to check whether a particular elements, character, number value, variable is inside the list, tupe, set, etc or not.

6.1 IN operator

This operator return True if a particular element is in the list, set or tuple, else return False.

```
In [304…
l1=[1,2,3,4,5,6]    #-----> Consider l1 is any list having some elements.
2 in l1       #-----> As l1 contain 2 in the list, that is why the result is True.
Out[304]:
True
```

```
In [305…
9 in l1           #-----> As l2 does not contain 9 in the list, that is why the res
Out[305]:
False
```

```
In [306…
5 in (7,8,9,1,2,3,4,5)
Out[306]:
True
```

```
In [307…
"Hello" in {"hello", "hell", "Helo"}
Out[307]:
False
```

Note: Python is a **case sensitive** language so take care of the case of the letters in the variable, operators, expression. The operator "in" must be in lower case not in upper case else it will not gonna work.

6.2 NOT IN operator

This operator return True if a particular element is not in the list, set or tuple, else return False.

```
In [308…
l1=[1,2,3,4,5,6]    #-----> Consider l1 is any list having some elements.
2 not in l1       #-----> As l1 contain 2 in the list, that is why the result is f
Out[308]:
False
```

```
In [309…
5 not in (7,8,9,1,2,3,4)
```

Out[309]:
```
True
```

In [310…
```
"Hello" not in {"hello", "hell", "Helo"}
```

Out[310]:
```
True
```

Note: Python is a **case sensitive** language so take care of the case of the letters in the variable, operators, expression. The operator "not in" must be in lower case not in upper case else it will not gonna work.

7. Identity Operators

There are 2 types of **identity operators.**

- is
- is not

  Note: This operators also return either True or False depending upon the condition. This operator is used to check whether the variables have same value or not.

7.1 IS operator

This returns True if both variables are the same object else return False.

In [325…
```
x=5
y=5
x is y
# Here variables x and y both store the same value 5 which means they both have sa
```

Out[325]:
```
True
```

In [326…
```
print(id(x))
print(id(y))
# Here you can clearly see the id of both the variable is same.
```

```
2263471489392
2263471489392
```

In [314…
```
a = 3
b = 5
a is b
```

Out[314]:
```
False
```

In [316…
```
a = 5
b = "5"
a is b
```

Out[316]:
```
False
```

In [323…
```
a = 5/2
b = 25/10
a is b
# Here, although both the variable have same value 2.5 but they are not same objec
# Because of that the result comes out False instead of True, here we can use "=="
```

```
Out[323]: False
```

```
In [324…  print(id(a))
          print(id(b))
          # Here you can clearly see the id of both the variable is different.

          2263582483984
          2263582479312
```

```
In [320…  5/2

Out[320]: 2.5
```

```
In [322…  25/10

Out[322]: 2.5
```

Note: Operator == can compare the variable or condition whereas "is" operator compare objects.

7.2 IS NOT operator

This returns False if both variables are the same object else return True.

```
In [328…  x=5
          y=5
          x is not y
          # Here variables x and y both store the same value 5 which means they both have sar

Out[328]: False
```

```
In [331…  a = 5/2
          b = 25/10
          a is not b

Out[331]: True
```

Note: Python is a **case sensitive** language so take care of the case of the letters in the variable, operators, expression. The operator "is not" must be in lower case not in upper case else it will not gonna work. Also note, in "not in" operator "not" comes first and then "in" whereas in "is not" operator "not" comes after "is".

Click this link to learn more: 👉 https://github.com/Abhishekk-Git