Operators in python

1. Arithmetic operator

In [50]:
```
# Addition
5 + 2
```

Out[50]: 7

In [51]:
```
# subtraction
5 - 2
```

Out[51]: 3

In [52]:
```
# Multiplication
5 * 2
```

Out[52]: 10

In [53]:
```
# Division
5/2
```

Out[53]: 2.5

In [54]:
```
# Modulus - It returns the remainder after dividing 5 by 2
5 % 2
```

Out[54]: 1

In [55]:
```
# Exponent - It returns the 5 to the power of 2
5 ** 2
```

Out[55]: 25

In [56]:
```
# Floor division - It returns the division value excluding the decimal
 part
5 // 2
```

Out[56]: 2

2. Assignment operators

In [57]:
```
# here x is a variable
# Assignment - It assi
x = 5
```

In [58]:
```
x += 2
```

In [59]:
```
x %= 4
```

3. Comparision operators

In [60]:
```
# less than and greater than operator

2 < 3   # Which is true
```

Out[60]: True

In [63]:
```
5 < 3   # which is false
```

Out[63]: False

In [65]:
```
# Comparion operator on strings

'b' > 'a'     # since b in greater than a
```

Out[65]: True

In [66]:
```
'hye' > 'hya' # returns true
```

Out[66]: True

Explanation:

h = h
y = y
e > a

In [67]:
```
#  Equals to - True if both operands are equal

'siddhesh' == 'siddhesh'  # string matching
```

Out[67]: True

In [36]:
```
'S' == 's'  # This will not be true since both the strings are differen
t in terms of CAPITALIZATION
```

Out[36]: False

In [37]:
```
# Not equal to - True if operands are not equal

5 != 2    # 5 is not equals to 2 - true
```

Out[37]: True

In [62]:
```
# Greater than or equal to - True if left operand is greater than or eq
ual to the right
5 >= 2   # which is true, infact
```

Out[62]: True

Logical Operator

These logical operators are same as the logical gates which you have learnt in 12th physics. like
AND - returns 1 if both the inputs are 1 else 0
OR - return 0 if both the inputs are 0 else 1
NOT - return 1, if input is 0 and 0, if input is 1

In [68]:
```
# and operator
3 < 4 and 's' == 's'     # both the inputs are true
```

Out[68]:
```
True
```

or operator

x = True y = False

x or y

In [73]:
```
not x
```

Out[73]:
```
False
```

Identity operator

In [76]:
```
# 'is' and 'not is' are the identity operator
x = 5
y = x

x is y     # true, since y references to x
```

Out[76]:
```
True
```

In [77]:
```
x = 5
y = 5
```

```
x is y     # true, since x and y both contain same value
```

Out[77]: True

Membership operator

'in' and 'not in' both are membership operators
Returns True if a sequence with the specified value is present in the object.

In [78]:
```
x = 'siddhesh'
's' in x
```

Out[78]: True

Bitwise operator

These operators work on the bits of the value.
It applies logical operator on the bits of the variable.

In [81]:
```
# & (bitwise and operator)
5 & 3
```

Out[81]: 1

Explanation:

binary representation of 5 is 0...0101 and
binary representation of 3 is 0...0011

AND operation on the bits of 5 and 3

num 5	num 3	result

0	0	0
.	.	.
0	0	0
1	0	0
0	1	0
1	1	1

0...0001 is the binary representation for 1 in decimal

In [86]:
```
# | (bitwise or)
5 | 3
```

Out[86]: 7

Explanation:

binary representation of 5 is 0...0101 and
binary representation of 3 is 0...0011

OR operation on the bits of 5 and 3

num 5	num 3	result
0	0	0
.	.	.
0	0	0
1	0	1
0	1	1
1	1	1

0...0111 is the binary representation for 7 in decimal.

In [85]:
```
# ^ (bitwise xor)
5 ^ 3
```

Out[85]: 6

Explanation:

binary representation of 5 is 0...0101 and
binary representation of 3 is 0...0011

same bits --> 0
different bits --> 1

XOR operation on the bits of 5 and 3

num 5	num 3	result
0	0	0
.	.	.
0	0	0
1	0	1
0	1	1
1	1	0

0...0110 is the binary representation for 6 in decimal.

In [88]:
```
# similarly for ~ (bitwise not)
~5
```

Out[88]: -6

In [89]:
```
~(-6)
```

Out[89]: 5

Zero fill left shift

x<<y It will move the y bits of x to the leftside and fill the space with y zeros

In [90]:
```
5<<1
```

Out[90]: 10

binary representation of 5 is 0...000101
after moving bits by one place to the left
number becomes - 0...001010 which is the binary representation for 10

Signed right shift

Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off.

In [92]:
```
5>>1
```

Out[92]: 2

binary representation of 5 is 0...000101
after moving bits by one place to the right
number becomes - 0...000010 which is the binary representation for 2

In [ ]:
```

```