Basic Operators in Python

Arithmetic operators: Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

OPERATORS

1. Addition: adds two operands || x + y
2. Subtraction: subtracts two operands || x - y
3. Multiplication: multiplies two operands || x * y
4. Division (float): divides the first operand by the second || x / y
5. Division (floor): divides the first operand by the second || x // y
6. Modulus: returns the remainder when first operand is divided by the second || x % y

In [2]:

```
#!pip install nbconvert
```

In [ ]:

```
# Examples of Arithmetic Operator
# Initialize the variables
a = 9
b = 4
```

In [ ]:

```
# Addition of numbers
add = a + b
print (add)
```

In [ ]:

```
# Subtraction of numbers
sub = a - b
print (sub)
```

In [ ]:

```
# Multiplication of number
mul = a * b
print (mul)
```

In [ ]:

```
# Division(float) of number
div1 = a / b
print (div1)
```

In [ ]:

```
# Division(floor) of number
div2 = a // b
print (div2)
```

In [ ]:

```
a,b
```

In [ ]:

```
# Modulo of both number
mod = a % b
print (mod)
```

Relational Operators:

```
Relational operators compares the values. It either returns True or False accord
ing to the condition.
```

1. Greater than: True if left operand is greater than the right || x > y
2. Less than: True if left operand is less than the right || x < y
3. Equal to: True if both operands are equal || x == y
4. Not equal to - True if operands are not equal || x != y
5. Greater than or equal to: True if left operand is greater than or equal to the right || x >= y
6. Less than or equal to: True if left operand is less than or equal to the right || x <= y

In [ ]:

```
# Examples of Relational Operators
a = 12
b = 20

# a > b is False
print(a > b)
```

In [ ]:

```
print (min (a, b))
```

In [ ]:

```
# a < b is True
print(a < b)
```

In [ ]:

```
# a == b is False
print(a == b)
```

In [ ]:

```
# a != b is True
print(a != b)
```

In [ ]:

```
# a >= b is False
a = 20
b = 200
print(a >= b)
```

In [ ]:

```
# a <= b is True
print(a <= b)
```

Logical operators:

```
Logical operators perform Logical AND, Logical OR and Logical NOT operations.
```

1. Logical AND: True if both the operands are true || x and y
2. Logical OR: True if either of the operands is true || x or y
3. Logical NOT: True if operand is false || not x

In [ ]:

```
1+0
```

In [ ]:

```
1*0
```

In [ ]:

```
# Examples of Logical Operator
a = True
b = False
```

In [ ]:

```
# Print a and b is False
print(a and b)
```

In [ ]:

```
# Print a or b is True
a = True
b = False

print(a or b)
```

In [ ]:

```
# Print not a is False
print(not a)
```

Bitwise operators:

```
Bitwise operators acts on bits and performs bit by bit operation.
```

1. Bitwise AND || x & y
2. Bitwise OR || x | y
3. Bitwise NOT || ~x

In [ ]:

```
0 : 0
1 : 01
2 : 10
3 : 11
4 : 100
5 : 101
6 : 110
7 : 111
8 : 1000
9 : 1001
10: 1010
```

In [ ]:

```
# Examples of Bitwise operators
a = 10
b = 4

'''
a_binary = 1010
b_binary = 0100
'''
# Print bitwise AND operation
print(a & b)
```

In [ ]:

```
'''
a_binary = 1010
b_binary = 0100
'''
# Print bitwise OR operation
print(a | b)
```

In [ ]:

```
'''
a_binary = 1010
b_binary = 0100
'''
# ~ tilde symbol
# Print bitwise NOT operation
print(~a)
```

In [ ]:

```
a = True
not a
```

Assignment operators:

```
Assignment operators are used to assign values to the variables.
```

1. Assign value of right side of expression to left side operand || x = y + z
2. Add AND: Add right side operand with left side operand and then assign to left operand || a+=b a=a+b
3. Subtract AND: Subtract right operand from left operand and then assign to left operand || a-=b a=a-b
4. Multiply AND: Multiply right operand with left operand and then assign to left operand || a=*b a=a*b
5. Divide AND: Divide left operand with right operand and then assign to left operand || a/=b a=a/b
6. Modulus AND: Takes modulus using left and right operands and assign result to left operand || a%=b a=a%b
7. Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand || a//=b a=a//b
8. Performs Bitwise AND on operands and assign value to left operand || a&=b a=a&b
9. Performs Bitwise OR on operands and assign value to left operand || a|=b a=a|b

In [ ]:

```
a = 5
b = 2
a += b
#a = a + b
print (a)
```

Assignment : Implement the Assignment Operators in the cells below

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Special operators:

There are some special type of operators like-

Identity operators-

is and is not are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

1. is || True if the operands are identical
2. is not || True if the operands are not identical

In [1]:

```
# Examples of Identity operators
a1 = 3
b1 = 3
a2 = 'DataScience'
b2 = 'DataScience'
a3 = [1,2,3]
b3 = [1,2,3]


print (a1 is not b1) # !=
```

```
False
```

In [2]:

```
print(a2 is b2)
```

```
True
```

In [7]:

```
type(a3)
```

Out[7]:

```
list
```

In [3]:

```
# Output is False, since lists are mutable.
print(a3 is b3)
```

```
False
```

In [9]:

```
4 in a3
```

Out[9]:

```
False
```

Membership operators-

in and not in are the membership operators; used to test whether a value or variable is in a sequence.

1. in || True if value is found in the sequence
2. not in || True if value is not found in the sequence

In [11]:

```
# Examples of Membership operator
x = 'Machine Learning'
y = {3:'a',4:'b'}
```

In [12]:

```
type(y)
```

Out[12]:

```
dict
```

In [13]:

```
print('M' in x)
```

```
True
```

In [14]:

```
print ('m' in x)
```

```
False
```

In [15]:

```
print('machine' not in x)
```

```
True
```

In [16]:

```
print('Machine' not in x)
```

```
False
```

In [17]:

```
print(3 in y)
```

```
True
```