In [1]:

```
#see the value of multiple statements
#output for all commands in python jupyter notebook
# Specifying which nodes should be run interactively such as 'all','last'..

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

This notebook deals with different types of operators in Python

What are operators in python?

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

For example: 8+3 = 11

Here, + is the operator that performs addition. 8 and 3 are the operands and 11 is the output of the operation.

1. Arithmetic operators
2. Bitwise operators
3. Assignment operators
4. Comparison (Relational) operators
5. Logical (Boolean) operators
6. Special operators

Arithmetic operators:

```
Arithmetic operators are used to perform mathematical operations like additio
n,
subtraction, multiplication and division.
```

- Addition: adds two operands and the syntax is x + y
- Subtraction: subtracts two operands and the syntax is x - y
- Multiplication: multiplies two operands and the syntax is x * y
- Division (float): divides the first operand by the second and the syntax is x / y
- Division (floor): divides the first operand by the second and the syntax x // y
- Modulus:Returns the remainder when first operand is divided by the second and the synatx is x % y

In [2]:

```
print(2 + 4)
print(12878 + 1)
print(7 - 3)
print(127 - 128)
print(8 * 6)
print(17 * 12)
```

```
6
12879
4
-1
48
204
```

There are two division operators: / for everyday (float) division and // for truncating (integer) division

In [4]:

```
print(17 / 4)
print(17 // 4)
```

```
4.25
4
```

The modulo operator % and the exponentiation operator ** are also available.

In [5]:

```
print(18 % 3)
print(72 % 5)
```

```
0
2
```

The usual operator precedence rules and use of parentheses to override that are available

In [6]:

```
print(10 ** 2 * 7 - 3)
print(10 ** (2 * 7) - 3)
print(10 ** (2 * 7 - 3))
```

```
697
99999999999997
100000000000
```

In [3]:

```
a = 123456789
b = a ** 2
c = b ** 3
d = c ** 4
e = d ** 5
print(e)
```

```
958952863472220347910866056628119697853871068539013465444839536426801167
993054293929163280448039645449064125681735868389753896938894669317247873
826022090268514137186729925009454590509679737540744466828401151851734912
328949430367182708052666864754395565918183061461425415310424576068784408
691689720122009297814507919738662894350588988639808054498211481999246620
265727724263303632728183536002926776399146501636024023870460829453965042
281573516177879660268666389774283362707610397338074992295741527447084067
099991981084218755936746700177249185329319981796616794873716079218694887
796144801746541830505541482462955134309726876547981403821624463403579823
482701864443881297209415411583905385414926857107526238526227207683344304
688520972269171491716609536262544669869407652533823580013391749032963471
697200236172979943497791606970504253730471210647484214341251283087641154
070120309491472640661167311715173396372668461182435867576704179829651253
32116703787163047835949441767799201
```

Assignment operators

The standard assignment operators are available. That is, $\alpha \odot = \beta$ is a shorthand for $\alpha = \alpha \odot \beta$ where $\odot$ is any binary arithmetic operator we saw above

In [10]:

```
a = 12
b = 5
a += b
print(a, b)
a -= b
print(a, b)
a *= b
print(a, b)
a **=b
print(a,b)
```

```
17 5
12 5
60 5
777600000 5
```

Boolean

The two constants True and False are defined.

The usual boolean operators are also available: ==, !=, >, >=, <, <=

In [1]:

```
a = 12
b = 13
print(a == b - 1,a == b, a != b, a < b, a >= b)
```

```
True False True True False
```

Float

Python has a float datatype (and it is the same as C's double!) and the above operations are available

In [3]:

```
a = 12.9
b = 3.6
c = a + b
d = a - b
e = a * b #Watch out for the round off error!
f = a / b
print(a, b, c, d, e, f, sep="\n")
print(a, b, c, d, e, f)
```

```
12.9
3.6
16.5
9.3
46.440000000000005
3.5833333333333335
12.9 3.6 16.5 9.3 46.440000000000005 3.5833333333333335
```

Bitwise operators:

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

- Bitwise AND---- x & y
- Bitwise OR---- x | y
- Bitwise NOT---- ~x
- Bitwise XOR---- x ^ y
- Bitwise right shift---- x>>
- Bitwise left shift---- x<<

In [1]:

```
a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101
c = 0

c = a & b;        # 12 = 0000 1100
print ("Value of c is ",c)

c = a | b;        # 61 = 0011 1101
print ("Value of c is ",c)

c = a ^ b;        # 49 = 0011 0001
print ("Value of c is ",c)


c = ~a;           # -61 = 1100 0011
print ("Value of c is ",c)

#The left operands value is moved left by the number of bits specified by the right
 operand.
c = a << 2;       # 240 = 1111 0000
print ("Value of c is ",c)

#The left operands value is moved right by the number of bits specified by the right
operand.
c = a >> 2;       # 15 = 0000 1111
print ("Value of c is ",c)
```

```
Value of c is  12
Value of c is  61
Value of c is  49
Value of c is  -61
Value of c is  240
Value of c is  15
```

Relational Operators:

```
Relational operators compares the values. It either returns True or False acc
ording to the condition.
```

- Greater than: True if left operand is greater than the right---- x > y
- Less than: True if left operand is less than the right---- x < y
- Equal to: True if both operands are equal---- x == y
- Not equal to - True if operands are not equal---- x != y
- Greater than or equal to: True if left operand is greater than or equal to the right---- x >= y
- Less than or equal to: True if left operand is less than or equal to the right---- x <= y

In [4]:

```
# Examples of Relational Operators
a = 25
b = 47

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

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

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

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

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

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

```
False
True
False
True
False
True
```

Logical operators:

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

- Logical AND: True if both the operands are true---- x and y
- Logical OR: True if either of the operands is true--- x or y
- Logical NOT: True if operand is false--- not x

In [5]:

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

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

# Print a or b is True
print(a or b)

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

```
False
True
False
```

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.

- is True if the operands are identical
- is not True if the operands are not identical

In [6]:

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

print(a1 is not b1)

print(a2 is b2)

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

```
False
True
False
```