In [101]:

```
# Operators: Operator is a symbol that performs certain operations.
# python provide set of operations.
# 1. Arithmetic operators
# 2. Relational opertors or comparsion operators.
# 3. Logical operators.
# 4. Bitwise operators.
# 5. Assignment operators.
# 6. Special operators.
```

In [103]:

```
# Arithmetic operators
a=100
b=200
print('a+b=',a+b)  # + ==> addition
print('a-b=',a-b)  # - ==> subraction
print('a*b=',a*b)  # * ==> multiplication
print('a/b=',a/b)  # / ==> division : it always returns floating point value/
print('a//b=',a//b) # // ==> floor division
print('a%b=',a%b)  # % ==> modulo operator
print('a**b=',a**b) # ** ==> exponent or power operator
```

```
a+b= 300
a-b= -100
a*b= 20000
a/b= 0.5
a//b= 0
a%b= 100
a**b= 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000
```

In [104]:

```
# floor dibision (//) can perform both floating point and integral arithmetic. If arguments are int type then res
ults is
# int type. If atleast one argument is int float type it returns float type
```

In [105]:

```
# we can perform + * operator for str type also
# if we want to use + operator for str type then compulsory both arguments should be str type only.
# if we want to use * operator for str type then compulsory one argument should be int and other argument should
be float.

# + ==> string concatenation operator.
# * ==> string repetetion operator.
```

In [107]:

```
# string concatenation
a='chris'+'jake'
print(a)
```

```
chrisjake
```

In [109]:

```
# string repetetion
a='chris'*3
print(a)
```

```
chrischrischris
```

In [110]:

```
# relational operators
# >,>=,<,<=
# we can apply relational operators for str types also
a=100
b=200
print(a>b)
print(a<b)
print(a>=b)
print(a<=b)
```

```
False
True
False
True
```

In [117]:

```
# relational operators for str types
s1='cat'
s2='cot'
print(s1>s2)
print(s1<s2)
print(s1>=s2)
print(s1<=s2)
```

```
False
True
False
True
```

In [121]:

```
# chaining of relational operators is possible. In the chaining, if all comparisons return true, then only result
is true.
# if atleast one comparison returns false. then result is false.
#eg:
print(10<20)
print(10<20<30)
print(10<20<30>40)
```

```
True
True
False
```

In [2]:

```
# equality operators.
# ==,!=
# we can apply these operators for any type even for incompatible types also
print(10==20)
print(10!=20)
print(10==True)
print('aravind'=='john')
print(10=='aravind')
print(0==False)
print(1==True)
```

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

In [139]:

```
# chaining concept is applicable for equality operators. if atleast one comparison returns false then result is f
alse.
# otherwise result is true.
print(10==20==30==40)
print(10==10==10==10)
```

```
False
True
```

In [140]:

```
# 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 argument is true then only result is true
# not ==> complement
```

In [143]:

```
print(True and False)
print(True or False)
print(not False)
```

```
False
True
True
```

In [3]:

```
# if x evaluates to false return x otherwise y
print(10 and 20)
print(0 and 20)

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

```
20
0
```

In [153]:

```
# x or y
# if x evaluates to true then result is x otherwise result is y
print(10 or 20)
print(0 or 20)
print(1 or 20)
```

```
10
20
1
```

In [150]:

```
# not x
# if x evaluates to false then result is true otherwise false
print(not 10)
print(not 0)
```

```
False
True
```

In [160]:

```
# bitwise operators
# the operators are applicable only for int and boolean types.
print(4&5) # &==> if both bits are 1 then only result is 1 otherwise result is 0
print(4|5) # | ==> if atleast one bit is 1 then result is 1 otherwise result is 0
print(4^5) # ^ ==> if both bits are different then only result is 1 otherwise result is 0.
print(~4) # ~ ==> bitwise complement operator 1 means 0 and 0 means 1
print(10>>4) # bitwise left shift operator
print(10<<4) # bitwise right shift operator
```

```
4
5
1
-5
0
160
```

In [161]:

```
# Assignment operator
# we can use assignment operator to assign value to the variable.
#eg:x=10
# we can combine assignment operator with some other operator to form compound assignment operator.
# eg:x+=10
# list of all possible compound assignment operators in python.
# +=,-=,*=,/=,//=,%=,&=,|=,^=,>>=,<<=
x=10
x+=20
print(x)
```

```
30
```

In [165]:

```
#special operators
# 1.Identity operators
# 2.Membership operators
# Identity operators ==> we can use identity operators for address comparison
# Identity operators are available.
# is, is not

emp1=['aravind','reddy',85000]
emp2=['aravind','reddy',85000]
print(id(emp1))
print(id(emp2))
print(emp1 is emp2)
print(emp1==emp2) # content comparison
```

```
1947069525064
1947070373704
False
True
```

In [168]:

```
# membership operators
# we can use membership operators to check whether the given object present in given collection.(it may be string
, list,
# tuple or dict)

#in -> returns true if the given object is present in given collection
# not in -> returns true if the given object not present in given collection.

x='learning python is very easy'
print('z' in x)
print('l' in x)

emp1=['aravind','reddy',84000,c]
print('aravind' in emp1)
```

```
False
True
True
```

In [172]:

```
# ternary operators
# syntax: x=firstvalue if condition else secondvalue
a,b=100,200
x=a+b if a>b else 400
print(x)
```

```
400
```