Operators

¶

Operator is a symbol that performs certain operations. Python provides the following set of operators

1. Arithmetic Operators
2. Relational Operators or Comparison Operators
3. Logical operators
4. Bitwise oeprators
5. Assignment operators
6. Special operators

1. Arithmetic Operators:

- '+' ==>Addition
- '-' ==>Subtraction
- '*' ==>Multiplication
- '/' ==>Division operator
- '%' ===>Modulo operator
- '//' ==>Floor Division operator
- '**' ==>Exponent operator or power operator

In [1]:
```
a=10
b=2
print('a+b=',a+b)
print('a-b=',a-b)
print('a*b=',a*b)
print('a/b=',a/b)
print('a//b=',a//b)
print('a%b=',a%b)
print('a**b=',a**b)
```

```
a+b= 12
a-b= 8
a*b= 20
a/b= 5.0
a//b= 5
a%b= 0
a**b= 100
```

In [2]:
```
a = 10.5
b=2

print('a+b=',a+b)
print('a-b=',a-b)
print('a*b=',a*b)
print('a/b=',a/b)
print('a//b=',a//b)
print('a%b=',a%b)
print('a**b=',a**b)
```

```
a+b= 12.5
a-b= 8.5
a*b= 21.0
a/b= 5.25
a//b= 5.0
a%b= 0.5
a**b= 110.25
```

In [3]:
```
print(10/2)
print(10//2)
print(10.0/2)
print(10.0//2)
```

```
5.0
5
5.0
5.0
```

Note:

- / operator always performs floating point arithmetic. Hence it will always returns float value.
- But Floor division (//) can perform both floating point and integral arithmetic. If arguments are int type then result is int type. If atleast one argument is float type then result is float type

Note:
We can use +,* operators for str type also. If we want to use + operator for str type then compulsory both arguments should be str type only otherwise we will get error.

In [4]:
```
# "Pratik"+10
# TypeError: can only concatenate str (not "int") to str
```

In [5]:
```
"Avinash"+"10"
```

Out[5]:
```
'Avinash10'
```

If we use * operator for str type then compulsory one argument should be int and other argument should be str type.

2*"Avinash"*
*"Rahul"*2

2.5*"Rahul" ==>TypeError: can't multiply sequence by non-int of type 'float'*
"Rahul""Rahul"==>TypeError: can't multiply sequence by non-int of type 'str'

'+'====>String concatenation operator
'*'===+>String multiplication operator

Note:
For any number x, x/0 and x%0 always raises "ZeroDivisionError"

10/0 10.0/0

2.Relational Operators:

'>,>=,<,<='

In [6]:
```
a=10
b=20
print("a > b is ",a>b)
print("a >= b is ",a>=b)
print("a < b is ",a<b)
print("a <= b is ",a<=b)
```

```
a > b is  False
a >= b is  False
a < b is  True
a <= b is  True
```

In [1]:
```
# We can apply relational operators for str types also

a="Rahul"
b="Rahul"
print("a > b is ",a>b)
print("a >= b is ",a>=b)
print("a < b is ",a<b)
print("a <= b is ",a<=b)
```

```
a > b is  False
a >= b is  True
a < b is  False
a <= b is  True
```

In [8]:
```
print(True>True)
print(True>=True)
print(10 >True)
print(False > True)
#print(10>'Rahul') # TypeError: '>' not supported between instances of 'in
t' and 'str'
```

```
False
True
True
False
```

In [9]:
```
a=10
b=20
if(a>b):
    print("a is greater than b")
else:
    print("a is not greater than b")
```

```
a is not greater than b
```

Note:
Chaining of relational operators is possible. In the chaining, if all comparisons returns True then only result is True. If atleast one comparison returns False then the result is False

In [11]:
```
print(10<20)
print(10<20<30)
print(10<20<30<40)
print(10<20<30<40>50)
```

```
True
True
True
False
```

3.equality operators:

== , !=

We can apply these operators for any type even for incompatible types also

In [16]:
```
print(10==20)
print(10!=20)
print(10==True)
print("Rahul"=="Rahul")
print("Avinash"=="Vishal")
print(10=="durga")
```

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

Note: Chaining concept is applicable for equality operators. If atleast one comparison returns False then the result is False. otherwise the result is True.

```
In [24]: 10==20==30==40
Out[24]: False
```

```
In [25]: 10==10==10==10
Out[25]: True
```

4.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 arugemnt is True then result is True
- not ==>complement

AND Truth Table

Inputs		Output
A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

Inputs		Output
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

- Note :
  0 is False
  1 is True

```
In [17]: True and False
Out[17]: False
```

```
In [18]: True or False
Out[18]: True
```

```
In [19]: not False
Out[19]: True
```

For non-boolean types behaviour:

- 0 means False
- non-zero means True
- empty string is always treated as False

x and y:

==>if x is evaluates to false return x otherwise return y

```
In [31]: 10 and 20
Out[31]: 20
```

```
In [32]: 0 and 20

         # If first argument is zero then result is zero otherwise result is y
Out[32]: 0
```

x or y:

If x evaluates to True then result is x otherwise result is y

```
In [33]: 10 or 20
Out[33]: 10
```

```
In [34]: 0 or 20
Out[34]: 20
```

not x:

If x is evalutates to False then result is True otherwise False

```
In [35]: not 10
Out[35]: False
```

```
In [36]: not 0
Out[36]: True
```

```
In [1]: "Avi" and "Avinash"
Out[1]: 'Avinash'
```

In [2]:
```
"" and "Avinash"
```
Out[2]: ''

In [3]:
```
"Rahul" and ""
```
Out[3]: ''

In [4]:
```
"" or "Pratik"
```
Out[4]: 'Pratik'

In [5]:
```
"Pratik" or ""
```
Out[5]: 'Pratik'

In [6]:
```
not ""
```
Out[6]: True

In [7]:
```
not "AVi"
```
Out[7]: False

Bitwise Operators:

We can apply these operators bitwise. These operators are applicable only for int and boolean types. By mistake if we are trying to apply for any other type then we will get Error.

&,|,^,~,<<,>>

In [8]:
```
print(4&5)
```
```
4
```

In [10]:
```
# print(10.5 & 5.6)
# TypeError: unsupported operand type(s) for &: 'float' and 'float
```

In [11]:
```
print(True & True)
```
```
True
```

& ==> If both bits are 1 then only result is 1 otherwise result is 0

| ==> If atleast one bit is 1 then result is 1 otherwise result is 0

^ ==>If bits are different then only result is 1 otherwise result is 0

~ ==>bitwise complement operator

1==>0 & 0==>1

<< ==>Bitwise Left shift

> ==>Bitwise Right Shift

In [12]:
```
print(4&5)
```

4

In [13]:
```
print(4|5)
```

5

In [14]:
```
print(4^5)
```

1

AND Truth Table

Inputs		Output
A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

Inputs		Output
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

bitwise complement operator(~):

We have to apply complement for total bits.

Eg: print(~5) ==>-6

Note:
The most significant bit acts as sign bit. 0 value represents +ve number where as 1 represents -ve value.

positive numbers will be repesented directly in the memory where as -ve numbers will be represented indirectly in 2's complement form.

Shift Operators:

<< Left shift operator

After shifting the empty cells we have to fill with zero

print(10<<2)==>40

AND Truth Table

Inputs		Output
A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

Inputs		Output
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

>> Right Shift operator

After shifting the empty cells we have to fill with sign bit.(0 for +ve and 1 for -ve)

print(10>>2) ==>2

AND Truth Table

Inputs		Output
A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

Inputs		Output
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

In [15]:
```
print(True & False)
print(True | False)
print(True ^ False)
print(~True)
print(True<<2)
print(True>>2)
```

```
False
True
True
-2
4
0
```

Assignment Operators:

We can use assignment operator to assign value to the variable.
Eg:
x=10

We can combine asignment operator with some other operator to form compound assignment operator.
Eg: x+=10 ====> x = x+10

The following is the list of all possible compound assignment operators in Python
+=
-=
*=
/=
%=
//=
**=
&=
|=
^=
'>>=
<<=

In [16]:
```
x=10
x+=20
print(x)
```

```
30
```

In [52]:
```
x=10
x&=5
print(x)
```

```
0
```

Ternary Operator:

Syntax:
x = firstValue if condition else secondValue

If condition is True then firstValue will be considered else secondValue will be considered.

In [17]:
```
a,b=10,20
x=30 if a<b else 40
print(x)
```

```
30
```

In [18]:
```
#Eg 2: Read two numbers from the keyboard and print minimum value

a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
min=a if a<b else b
print("Minimum Value:",min)
```

```
Minimum Value: 11
```

Note: Nesting of ternary operator is possible.

In [19]:
```
#Q. Program for minimum of 3 numbers
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))
min=a if a<b and a<c else b if b<c else c
print("Minimum Value:",min)
```

```
Minimum Value: 11
```

In [20]:
```
#Q. Program for maximum of 3 numbers
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))
max=a if a>b and a>c else b if b>c else c
print("Maximum Value:",max)
```

```
Maximum Value: 13
```

In [21]:
```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
print("Both numbers are equal" if a==b else "First Number is Less than Seco
nd Number" if
a<b else "First Number Greater than Second Number")
```

```
First Number is Less than Second Number
```

Special operators:

Python defines the following 2 special operators

1. Identity Operators
2. Membership operators
3. Identity Operators

We can use identity operators for address comparison. 2 identity operators are available

1. is
2. is not

r1 is r2 returns True if both r1 and r2 are pointing to the same object r1 is not r2 returns True if both r1 and r2 are not pointing to the same object

In [58]:
```
a=10
b=10
print(a is b)
x=True
y=True
print( x is y)
```

```
True
True
```

In [2]:
```
a="Rahul"
b="Rahul"
print(id(a))
print(id(b))
print(a is b)
```

```
1791259030448
1791259030448
True
```

In [3]:
```
list1=["one","two","three"]
list2=["one","two","three"]
print(id(list1))
print(id(list2))
print(list1 is list2)
print(list1 is not list2)
print(list1 == list2)
```

```
1791257980992
1791259030400
False
True
True
```

Note:
We can use is operator for address comparison where as == operator for content comparison.

2. Membership operators:
We can use Membership operators to check whether the given object present in the given collection.(It may be String,List,Set,Tuple or Dict)

in ===> Returns True if the given object present in the specified Collection

not in ===> Retruns True if the given object not present in the specified Collection

In [4]:
```
x="hello learning Python is very easy!!!"
print('h' in x)
print('d' in x)
print('d' not in x)
print('Python' in x)
```

```
True
False
True
True
```

In [5]:
```
list1=["sunny","bunny","chinny","pinny"]
print("sunny" in list1)
print("tunny" in list1)
print("tunny" not in list1)
```

```
True
False
True
```

Operator Precedence:
If multiple operators present then which operator will be evaluated first is decided by operator precedence.

Eg: print(3+10*2) ===> 23 print((3+10)*2) ===> 26

The following list describes operator precedence in Python

- () ==> Parenthesis
- ** ==> exponential operator
- ~,- ==> Bitwise complement operator,unary minus operator
- *,/,%,// ==> multiplication,division,modulo,floor division
- +,- ==> addition,subtraction
- <<,>> ===> Left and Right Shift
- & ==> bitwise And
- ^ ==> Bitwise X-OR
- | ==> Bitwise OR
- '>,>=,<,<=, ==, != ==>Relational or Comparison operators
- =,+=,-=,*=... ==>Assignment operators
- is , is not ==> Identity Operators
- in , not in ==> Membership operators
- not ==> Logical not
- and ==> Logical and
- or ==> Logical or

In [6]:
```
a=30
b=20
c=10
d=5
print((a+b)*c/d)
print((a+b)*(c/d))
print(a+(b*c)/d)
```

```
100.0
100.0
70.0
```

Mathematical Functions (math Module)

- A Module is collection of functions, variables and classes etc.
- math is a module that contains several functions to perform mathematical operations
- If we want to use any module in Python, first we have to import that module.

> import math Once we import a module then we can call any function of that module.

In [7]:
```
import math
print(math.sqrt(16))
print(math.pi)
```

```
4.0
3.141592653589793
```

We can create alias name by using as keyword.

In [8]:
```
import math as m
```

Once we create alias name, by using that we can access functions and variables of that module

In [9]:
```
import math as m
print(m.sqrt(16))
print(m.pi)
```

```
4.0
3.141592653589793
```

We can import a particular member of a module explicitly as follows

In [10]:
```
from math import sqrt
from math import sqrt,pi
```

If we import a member explicitly then it is not required to use module name while accessing

In [11]:
```
from math import sqrt,pi
print(sqrt(16))
print(pi)
print(math.pi)
```

```
4.0
3.141592653589793
3.141592653589793
```

important functions of math module:

- ceil(x)
- floor(x)
- pow(x,y)
- factorial(x)
- trunc(x)
- gcd(x,y)
- sin(x)
- cos(x)
- tan(x)

important variables of math module:

pi3.14
e===>2.71
inf ==>infinity
nan ==>not a number

Q. Write a Python program to find area of circle

In [12]:
```
from math import pi
r=16
print("Area of Circle is :",pi*r**2)
```

```
Area of Circle is : 804.247719318987
```

In [ ]: