Chapter-7: Python Operators

What is operator?

Operator is a special symbol which is used to perform operations.
To perform operations, operators required operands.
Based on the number of operands used by operators, operators are classified into 3 categories.

1. Unary Operators
2. Binary Operators
3. Ternary Operators

Unary operators required one operand to perform operation
Binary operator required two operands to perform operation
Ternary operator required three operands to perform operation

Types of operators

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Bitwise Operators
6. Conditional Operator
7. Membership Operator
8. Identity Operator
9. Walrus Operator

1.Arithmetic Operators

Arithmetic operators are binary operators and require 2 operands to perform operations.

Operator	Description
+	Addition and Concatenation
-	Subtraction
*	Multiplying and Repeating
/	Float division
//	Floor division
%	Modular (OR) Modulo
**	Exponent Operator OR Power of Operator

+ Operator :

+ operator in python is used to perform 2 operations

1. Adding numbers
2. Concatenating sequences (list, str, tuple,…)

If two operands are numeric type + operator performs addition
If two operands are sequence type + operator perform concatenation

Example:

```
a=10+20
b=1.5+2.5
c=1+2j+1+3j
d=True+True
e="PYTHON"+"LANGUAGE"
f=[10,20]+[30,40]
print(a,b,c,d,e,f,sep="\n")
```

```
g=10+2.5
h=10+2.5+1+2j
i=1+1+2j
j=1+4.5
k=2.5+1+2j
print(g,h,i,j,k,sep="\n")
l=int(1+4.5)
print(l)
```

Output

```
30
4.0
(2+5j)
2
PYTHONLANGUAGE
[10, 20, 30, 40]
12.5
(13.5+2j)
(2+2j)
5.5
(3.5+2j)
5
```

```
"PYTHON"+[10,20]
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    "PYTHON"+[10,20]
TypeError: can only concatenate str (not "list") to str
```

- Subtraction Operator

It is a binary operator which is used for perform arithmetic subtraction.

```
>>> n1=10-5
>>> n2=1.5-1.3
>>> n3=1+2j-1+1j
>>> print(n1,n2,n3,sep="\n")
5
0.19999999999999996
3j
```

Example:

```
# Write a program to swap two integer numbers

num1=int(input("Enter First Integer:"))
num2=int(input("Enter Second Integer:"))

print(f'Before Swaping {num1},{num2}')
#Method-1 Using Third Variable
num3=num1
num1=num2
num2=num3
print(f'After Swaping {num1},{num2}')

#Method-2 Without Using Third Variable
num1=num1+num2
num2=num1-num2
num1=num1-num2
print(f'After Swaping {num1},{num2}')

num1,num2=num2,num1
print(f'After Swaping {num1},{num2}')
```

Output

```
Enter First Integer:10
Enter Second Integer:20
```

Before Swaping 10,20
After Swaping 20,10
After Swaping 10,20
After Swaping 20,10

* **Multiplication or Repeating of sequence elements**

***is** a binary operator and required 2 operands to perform operations
This operator performs 2 operations

1. Multiplication
2. Repeating sequence of elements

If two operands are numeric type it performs multiplication

```
>>> a=5*2
>>> print(a)
10
>>> b=1.5*2
>>> print(b)
3.0
>>> c=True*4.5
>>> print(c)
4.5
```

If one operand is sequence type and another is operand is integer it perform sequence element repeating

Example:

```
>>> s1="A"*10
>>> print(s1)
AAAAAAAAAA
>>> s2="PYTHON"*5
>>> print(s2)
PYTHONPYTHONPYTHONPYTHONPYTHON
>>> s3=10*"-"
```

```
>>> print(s3)
----------
>>> a=[0]*100
>>> print(a)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> b=[1,2,3]*5
>>> print(b)
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> s4="*"*20
>>> print(s4)
********************
>>> s4=2.5*"A"
Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    s4=2.5*"A"
TypeError: can't multiply sequence by non-int of type 'float'
```

Example

```
# Write a program to find area of circle

radious=float(input("Enter Radious of Circle :"))
area=3.147*radious*radious
print(f'Area of circle with radious {radious} is {area}')
```

Output

```
Enter Radious of Circle :1.2
Area of circle with radious 1.2 is 4.53168
```

Example

Write a program to convert dollars into rs

```
dollars=int(input("Enter Dollars :"))
rs=dollars*91

print(f'''Dollars={dollars}
Rs={rs}''')
```

Output

```
Enter Dollars :2
Dollars=2
Rs=182

Enter Dollars :10
Dollars=10
Rs=910
```

/ float division operator

/ is called division and it is binary operator and required 2 operands to perform operation.
This operator after division it returns quotient.
This operator always return quotient in float type.

Syntax: operand1/operand2

Operand1 is called dividend
Operand2 is called divisor

```
>>> a=4/2
>>> print(a)
2.0
>>> b=5/2
>>> print(b)
2.5
```

```
>>> c=9/3
>>> print(c)
3.0
```

Example:

```
# Write a program to find area of triangle
# area=1/2xbasexheight

base=float(input("Base of Triangle :"))
height=float(input("Height of Triangle :"))

area=1/2*base*height

print(f'Area of triangle is {area:.2f}')
```

Output

```
Base of Triangle :1.5
Height of Triangle :1.2
Area of triangle is 0.90
```

Example:

```
# Write a program to input rollno,name,
# 2 subject marks and calculate total,avg

rno=int(input("Rollno :"))
name=input("Name :")
sub1=int(input("Subject1Marks :"))
sub2=int(input("Subject2Marks :"))

total=sub1+sub2
avg=total/2

print(f"'Rollno {rno}
```

StudentName {name}
Subject1Marks {sub1}
Subject2Marks {sub2}
TotalMarks {total}
AvgMarks {avg:.2f}''')

Output

Rollno :1
Name :naresh
Subject1Marks :60
Subject2Marks :80
Rollno 1
StudentName naresh
Subject1Marks 60
Subject2Marks 80
TotalMarks 140
AvgMarks 70.00

What is indentation?
The space given at the beginning of statement is called indentation.
Indentation is used for creating blocks.

Example:

```
# Write a program to find simple interest
# si=ptr/100

p=int(input("Amount :"))
t=int(input("Time :"))
r=float(input("Rate :"))

si=p*t*r/100

print(f'Simple Interest is {si:.2f}')
```

Output

Amount :50000
Time :12
Rate :2.5
Simple Interest is 15000.00

eval()

eval() is a predefined function in python
This function is used to evaluate string representation of expression and return result.

Example:

```
>>> a=eval("25")
>>> print(a,type(a))
25 <class 'int'>
>>> b=eval("1.5")
>>> print(b,type(b))
1.5 <class 'float'>
>>> c=eval("1+2j")
>>> print(c,type(c))
(1+2j) <class 'complex'>
>>> d=eval("1+2+3+5")
>>> print(d,type(d))
11 <class 'int'>
>>> e=eval("1/2")
>>> print(e,type(e))
0.5 <class 'float'>
```

Example:

```
# Write a program to print sum of two numbers

value1=eval(input("Enter First Number "))
```

```
value2=eval(input("Enter Second Number "))

value3=value1+value2

print(f'Sum of {value1} and {value2} is {value3}')
```

Output

```
Enter First Number 10
Enter Second Number 20
Sum of 10 and 20 is 30

Enter First Number 1.5
Enter Second Number 1.2
Sum of 1.5 and 1.2 is 2.7

Enter First Number 1+2j
Enter Second Number 1+3j
Sum of (1+2j) and (1+3j) is (2+5j)
```

// floor division (OR) integer division operator

// is called division and it is binary operator and required 2 operands to perform operation.
This operator after division it returns quotient.
This operator returns quotient in integer type.

```
>>> a=5/2
>>> print(a)
2.5
>>> b=5//2
>>> print(b)
2
>>> c=7/2
>>> print(c)
```

```
3.5
>>> d=7//2
>>> print(d)
3
>>> e=-7/2
>>> print(e)
-3.5
>>> f=-7//2
>>> print(f)
-4
```

Example:

```
# Write a program input days and convert into weeks

days=int(input("Days :"))
weeks=days//7

print(f'{days}={weeks}')
```

Output

```
Days :15
15=2
```

Example:

```
# Write a program to remove last digit of input number

num=int(input("Enter any number "))
num1=num//10

print(f'Original Number {num}')
print(f'After Deleting {num1}')
```

Output

```
Enter any number 123
```

Original Number 123
After Deleting 12

Enter any number 12
Original Number 12
After Deleting 1

Enter any number 2
Original Number 2
After Deleting 0

% Modular Operator

% is called modular operator and is binary operator, required 2 operands to perform operations.
This operator divides two numbers and return remainder.

```
>>> a=10%4
>>> print(a)
2
>>> b=5%2
>>> print(b)
1
>>> c=9%3
>>> print(c)
0
```

Example:

```
# Write a program to read last digit of input integer value

num=int(input("Enter any integer value "))
last_digit=num%10

print(f'Last Digit is {last_digit}')
```

Output

Enter any integer value 568949
Last Digit is 9

**** Exponential operator**

Exponential is a binary operator and required 2 operands to perform operation.
This operator is used to find power of a number

```
>>> a=2**8
>>> print(a)
256
>>> b=8**3
>>> print(b)
512
>>> c=3**4
>>> print(c)
81
```

Operator Precedence

Operator Precedence

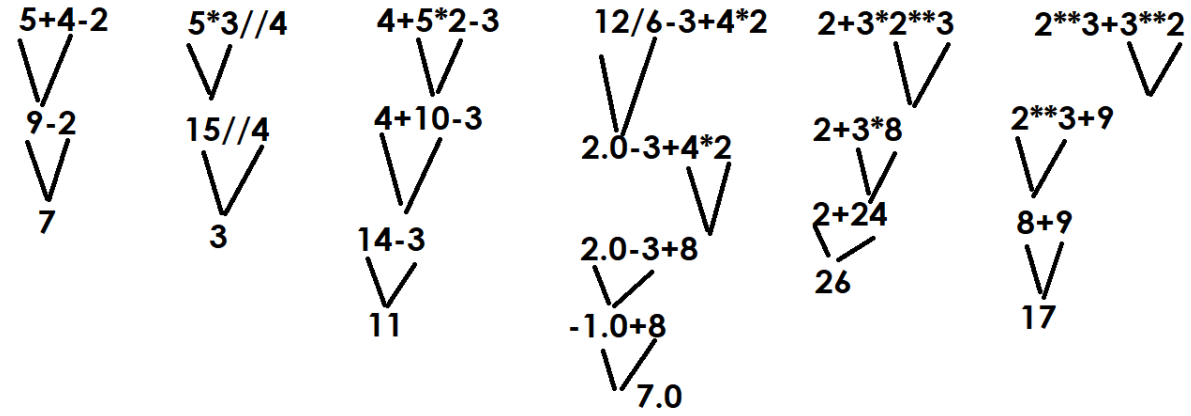
The following table summarizes the operator precedence in Python, from highest precedence (most binding) to lowest precedence (least binding). Operators in the same box have the same precedence. Unless the syntax is explicitly given, operators are binary. Operators in the same box group left to right (except for exponentiation and conditional expressions, which group from right to left).

Operator	Description
(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or parenthesized expression, list display, dictionary display, set display
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
await x	Await expression
**	Exponentiation
+x, -x, ~x	Positive, negative, bitwise NOT
*, @, /, //, %	Multiplication, matrix multiplication, division, floor division, remainder [6]
+, -	Addition and subtraction
<<, >>	Shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
not x	Boolean NOT
and	Boolean AND
or	Boolean OR

Operator	Description
if – else	Conditional expression
lambda	Lambda expression
:=	Assignment expression

Precedence of Arithmetic Operators

Operators	Description
**	Exponentiation
*,/,//,%	Multiplication, Float Division, Floor Division, Modulo
+,-	Addition, Subtraction



Relational Operators

Relational operators are binary operators.
Relational operators are used for comparing values.
An expression using relational operators is called boolean expression.
Relational operators after comparing values it returns Boolean value (True/False).

Operators	Description
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Equal

These operators can be used for comparing scalar type or collection type.

```
a=10
b=20
print(a==b)
print(10==10)
c=10
print(a==c)
username="nit"
password="n123"
uname=input("UserName :")
pwd=input("Password :")
print(uname==username,password==pwd)
```

Output

```
False
True
True
UserName :ab
Password :xy
False False
```

Example:

```
print(10==20)
print(10==10)
print(1.5==1.5)
```

```
print(1.4==1.9)
print("A"=="a")
print(True==True)
print(True==False)
print(1+2j==1+2j)
print("PYTHON"=="python")
```

Output

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

ord(): This function returns ascii values input letter or character

```
>>> ord('A')
65
>>> ord('B')
66
>>> ord('Z')
90
>>> ord('a')
97
>>> ord('b')
98
>>> ord('z')
122
>>> ord('0')
48
```

```
>>> ord('1')
49
>>> ord('9')
57
```

chr() : This function returns the character value of input ascii value.

```
>>> chr(65)
'A'
>>> chr(66)
'B'
>>> chr(97)
'a'
>>> chr(98)
'b'
>>> chr(122)
'z'
>>> chr(48)
'0'
>>> chr(49)
'1'
```

Example:

```
# Write a program input an alphabet in uppercase
# and convert into lowercase

ch=input("Enter Single Alphabet in uppercase(A-Z): ")
ch1=chr(ord(ch)+32)
print(ch)
print(ch1)
ch2=input("Enter Single Alphabet in lowercase (a-z): ")
ch3=chr(ord(ch2)-32)
print(ch2)
print(ch3)
```

Output

```
Enter Single Alphabet in uppercase(A-Z): R
R
r
Enter Single Alphabet in lowercase (a-z): e
e
E
```

Example:

```
print(10>20)
print(20>10)
print(1.5>1.2)
print(1.2>1.5)
print("a">"A")
print("A">"a")
print(True>False)
print(False>True)
```

Output

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

Example:

```
print(10>=10)
print(10>10)
print(10==10)
print(20>=10)
```

```
print(1.5>=1.2)
print(1.5>=1.5)
print("a">="A")
print("A">="a")
```

Output

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

Example:

```
# Write a program to compare input otp is equal to generated
# otp number

import random
otp=random.randint(1000,9999)
print(f'Generated OTP {otp}')
iotp=int(input("Enter OTP number :"))
print(otp==iotp)
```

Output

```
Generated OTP 7825
Enter OTP number :7825
True

Generated OTP 6725
Enter OTP number :6752
False
```

Example:

```
print(10<5)
print(5<10)
print(10<=10)
print(5<=10)
print("A"<="a")
print("A"<="A")
print("a"<="b")
print(1.2<=1.2)
print(1.2<=1.0)
```

Output

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

Example:

```
print(10>5>2)
print(10>5<2)
n1=int(input("Enter any integer "))
print(0<=n1<=10)
```

Output

```
True
False
Enter any integer 8
True
```

Example:

```
print(10>5>2)
print(10>5<2)
n1=int(input("Enter any integer "))
print(0<=n1<=10)
```

Output

```
True
False
Enter any integer 8
True
```

Logical Operators

Logical operators are used to combine two or more boolean expressions.

In python logical operators are represented using keywords

1. and
2. or
3. not

and, or are called binary operators and required 2 operands
not is unary operator and required 1 operand

and operator

"and" is a keyword which represents a logical operator in python. It is a binary operator and requires 2 operands to perform operation.

Truth table "and" operator

Opr1	**Opr2**	**Opr1 and Opr2**
True	True	**True**
True	False	**False**
False	True	**False**

False	False	**False**

Example:

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
>>> 10>5 and 10>9
True
>>> 10>5 and 10>20
False
>>> 10>20 and 10>5
False
>>> 10>20 and 10>30
False
```

In "and" operation, if opr1 is True then python evaluates opr2 and return result of opr2
If opr1 is False, it does not evaluate opr2 it returns result of opr1

POC's

```
>>> 100 and 200
200
>>> 0 and 100
0
>>> 100 and 0
0
>>> "PYTHON" and "JAVA"
'JAVA'
>>> 1.5 and 2.5
```

2.5
>>> 10 and 20 and 30
30
>>> 10 and 0 and 30
0

“or” operator

“or” is keyword, which represents logical operator in python
Truth table of “or” operator

Opr1	**Opr2**	**Opr1 or Opr2**
True	True	True
False	True	True
True	False	True
False	False	False

In “or” operation if any one condition/operand is True, the complete expression returns True.

>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
>>> 10>5 or 10>6
True
>>> 10>5 or 10>20
True
>>> 10>20 or 10>5
True

>>> 10>20 or 10>30
False

In “or” operation,
If operand1 is True, it does not evaluates operand2, it returns result of operand1
If operand1 is False, it evaluates operand2 and returns result of operand2

POC’s
>>> 100 or 200
100
>>> 0 or 100
100
>>> 0 or 0
0
>>> 100 or 200 or 300
100
>>> "python" or "java"
'python'
>>> 0 or 0 or 300
300

“not” operator

“not” is a keyword which represents logical operator
“not” is unary operator and required 1 operand

Opr1	**not Opr1**
True	False
False	True

>>> not True

```
False
>>> not False
True
>>> not 10>30
True
>>> not 10>5
False
```

Precedence of logical operators

1. not
2. and
3. or

```
>>> 100 and 200 or 300
200
>>> 100 or 200 and 300
100
>>> not 100 or 200 and 300
300
>>> (100 or 200) and 300
300
```

Conditional Operator

Conditional operator is ternary operator and requires 3 operands to perform operations.
Conditional operators are for creating conditional expression.
Conditional operator is used to evaluate expression based on condition or test.

Syntax:

variable-name=opr1 if opr2 else opr3

opr2 □ test/condition

opr1 □ true-expression, this expression is evaluated if condition is True
opr3 □ false-expression, this expression is evaluated if condition if False

```
>>> "Hello" if True else "Bye"
'Hello'
>>> 100 if True else 200
100
>>> 100 if False else 200
200
>>> "Hello" if False else "Bye"
'Bye'
```

Example:

```
# Voter Elg App

name=input("Name :")
age=int(input("Age :"))

print(f'{name} is elg to vote') if age>=18 else print(f'{name} is not elg to vote')
```

Output

```
Name :naresh
Age :50
naresh is elg to vote

Name :suresh
Age :13
suresh is not elg to vote
```

Write a program to find a given alphabet is vowel or not vowel

```
ch=input("Enter Alphabet :")
result="Vowel" if ch=='a' or ch=='e' or ch=='i' or ch=='o' or ch=='u'
else "Not Vowel"
print(result)
```

Output

```
Enter Alphabet :i
Vowel

Enter Alphabet :x
Not Vowel
```

Example:

```
# Write a program to find result of student(PASS/FAIL)

name=input("StudentName :")
sub1=int(input("Subject1Marks :"))
sub2=int(input("Subject2Marks :"))
result="PASS" if sub1>=40 and sub2>=40 else "FAIL"

print(f'''StudentName {name}
Subject1Marks {sub1}
Subject2Marks {sub2}
Result {result}''')
```

Output

```
StudentName :suresh
Subject1Marks :99
Subject2Marks :20
StudentName suresh
Subject1Marks 99
Subject2Marks 20
Result FAIL
```

Using multiple conditional operators

Multiple conditional operators are used to check or evaluate multiple conditions.

Syntax:

opr1 if opr2 else opr3 if opr4 else opr5 if opr6 else opr7

Example:

```
# Write a program to find input number is
# +ve, -ve or zero

num=int(input("Enter any number "))
print("+ve") if num>0 else print("-ve") if num<0 else print("zero")
```

Output

```
Enter any number 25
+ve

Enter any number -5
-ve

Enter any number 0
zero
```

 # Write a program to find student grade based on avg marks
 # avg>=90 --> A
 # avg>=70<90 --> B
 # avg>=50<70 --> C
 # avg<50 --> D

```
avg=float(input("Avg Marks :"))
grade="A" if avg>=90 else "B" if avg>=70 and avg<90 else "C" if
avg>=50 and avg<70 else "D"
```

```
print(f'''Avg Marks {avg}
Student Grade {grade}''')
```

 # Write a program to find the max of 3 numbers

```
a=int(input("Enter First Number "))
b=int(input("Enter Second Number "))
c=int(input("Enter Third Number "))

max_value=a if a>b and a>c else b if b>a and b>c else c

print(f'''a={a}
b={b}
c={c}
maximum={max_value}''')
```

Output

```
Enter First Number 10
Enter Second Number 30
Enter Third Number 20
a=10
b=30
c=20
maximum=30
```

Membership operator

Membership operator is a binary operator and requires 2 operands to perform operation.

In python membership operator is represented using a keyword

1. in
2. not in

Membership operator is used for searching a given value inside a collection of values.

Membership operator returns boolean value(True/False).
It returns True, if value exist in collection of values
It return False, if value not exists in collection of values

Syntax:
Operand1 in Operand2

Operand1 can be any type (scalar type/collection type)
Operand2 must be collection type

```
>>> 10 in [100,200,300]
False
>>> 100 in [100,200,300]
True
>>> "abc@gmail.com" in ['nit@nareshit.com',
...                     'naresh@nareshit.com',
...                     'abc@gmail.com']
True
>>> "suresh@gmail.com" in ['nit@nareshit.com',
...                     'naresh@nareshit.com',
...                     'abc@gmail.com']
False
>>> "py" in "python"
True
```

Example:

```
#Login with username
users=["naresh","suresh","ramesh"]
uname=input("UserName :")
```

```
print(f'{uname},welcome') if uname in users else print(f'{uname} is invalid')
```

Output

```
UserName :naresh
naresh,welcome

UserName :kishore
kishore is invalid
```

Example:

```
# Write a program to find input alphabet is vowel or not

ch=input("Enter any alphabet ")
print("Vowel") if ch in "aeiouAEIOU" else print("Not Vowel")
```

Output

```
Enter any alphabet a
Vowel

Enter any alphabet E
Vowel

Enter any alphabet x
Not Vowel
```

Example:

```
>>> "a" not in "java"
False
>>> "a" in "java"
True
```

```
>>> "k" not in "java"
True
>>> "k" in "java"
False
>>> 100 not in [10,20,30,40,50]
True
>>> 10 not in [10,20,30,40,50]
False
```

Identity Operator

Identity operator is a binary operator and requires 2 operands to perform operation.
Identity operator is represented using the following keyword

1. is
2. is not

What is the identity of an object?
Python is an object oriented programming language.
In object oriented programming languages data is represented as objects and every data type is class.
Every object created inside memory has a unique address which is called the identity of the object.

Identity operator is used for comparing if two variables pointing to the same address in memory (OR) identity operator is used for finding two variables pointing to the same value in memory.

Identity operator returns boolean value (True/False)

How to find the id/identity of an object?
id() function returns identity/address of object

Write a program to id of objects

```
a=10
b=20
print(id(a),id(b))
print(a,b)
```

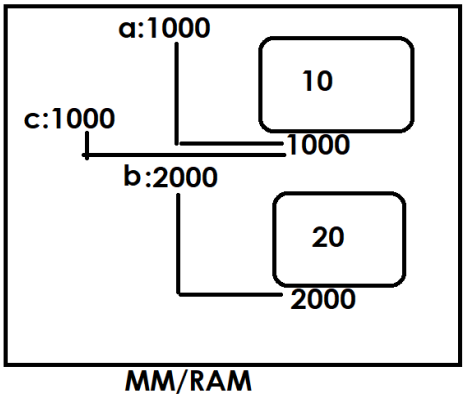
Output

```
1407203092371 44 140720309237464
10 20
```

a=10

b=20

c=a



Python data types or objects can be,

1. mutable objects
2. immutable objects

Mutable objects allow you to do changes within the same object or memory location. These mutable objects are created using mutable data types.

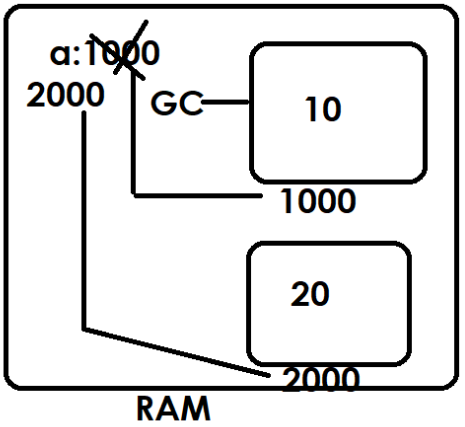
1. List
2. Bytearray
3. Set
4. Dict

Immutable objects are not allowed to do changes in the same object or memory location. Immutable objects are created using immutable data types.

1. Int
2. Float
3. Complex
4. Bool
5. NoneType
6. Range
7. Tuple
8. Str
9. Bytes
10. Frozenset

a=10

a=20



Garbage Collector (GC) is an automatic memory management system that removes unused objects from memory to free resources and prevent memory leaks.

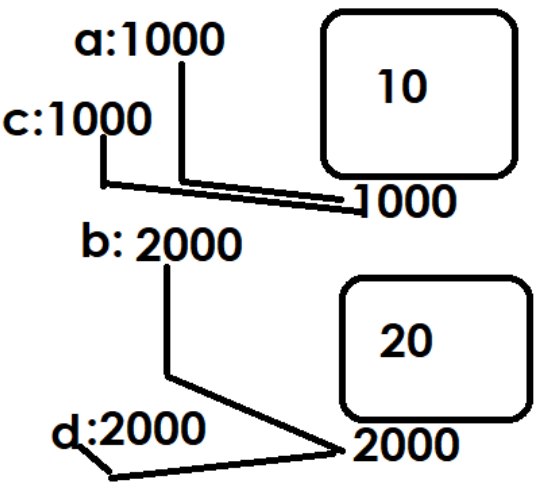
```
>>> a=10
>>> id(a)
140720309237144
>>> b=20
>>> id(b)
140720309237464
>>> a=30
>>> id(a)
140720309237784
```

Immutable objects are sharable objects, it is a shared number of variables.

```
>>> x=100
>>> y=100
>>> z=100
>>> p=100
>>> print(id(x),id(y),id(z),id(p))
140720309240024 140720309240024 140720309240024
140720309240024
```

Find how many variables and objects created in the following code?

```
a=10
b=20
c=b-a
d=20
```




```
>>> A=[10,20,30,40,50]
>>> B=[10,20,30,40,50]
>>> id(A),id(B)
(2576204995008, 2576204994560)
>>> A==B
True
>>> A is B
False
>>> C=A
>>> C==A
True
>>> C is A
```

True

Mutable objects are not sharable.

What is the difference between == and is operators?

==	Is
Relational Operator	Identity Operator
This operator is used for comparing object values.	This operator is used for comparing address (two variables pointing to same address in memory)

Bitwise Operators

Bitwise operators are binary operators and require 2 operands to perform operations.
Python support the following bitwise operators

Operators	Description
&	Bitwise and operator
|	Bitwise or operator
^	Bitwise XOR operator
~	Bitwise not operator
>>	Right shift operator
<<	Left shift operator

Applications of bitwise operators

1. Arithmetic Operations
2. Operating System
3. Networking
4. Image, Audio and Video Processing
5. Embedded Applications
6. Compressing Data
7. Encryption and Decryption

Bitwise operators perform operations on bits (0's and 1's)/Binary data.

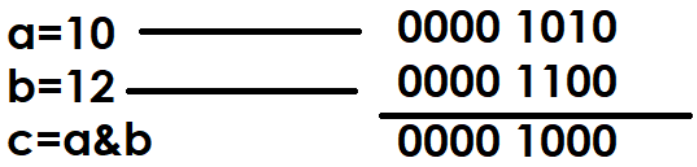
Note: bitwise operators are applied or used only on binary data or integer values.

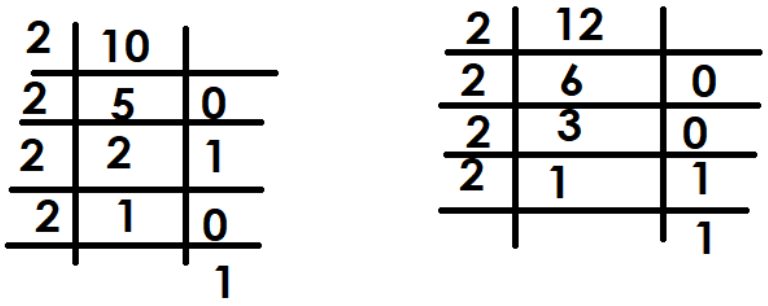
Bitwise & (and) operator

Bitwise & (and) operator is used to apply and gate.

Truth table of bitwise & (and) operator

Opr1	Opr2	Opr1 & Opr2
1	0	0
0	1	0
0	0	0
1	1	1






```
>>> a=10
>>> b=12
>>> c=a&b
>>> print(a,b,c)
10 12 8
>>> print(bin(a),bin(b),bin(c))
```

Example:

```
# Write a program to find input number
# is even or odd using bitwise operator

num=int(input("Enter any number "))
print("even") if num&1==0 else print("odd")
print("even") if num%2==0 else print("odd")
```

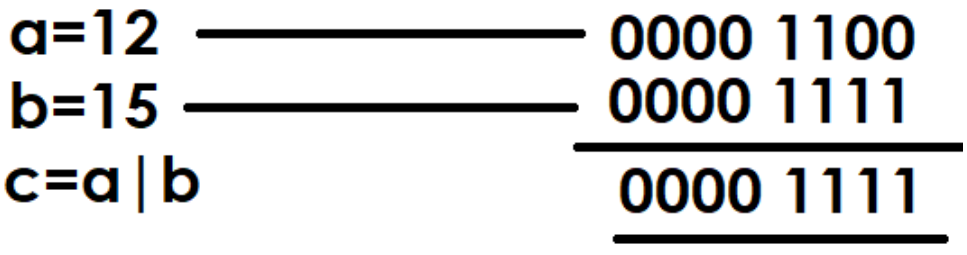
Output

```
Enter any number 5
odd

Enter any number 8
Even
```

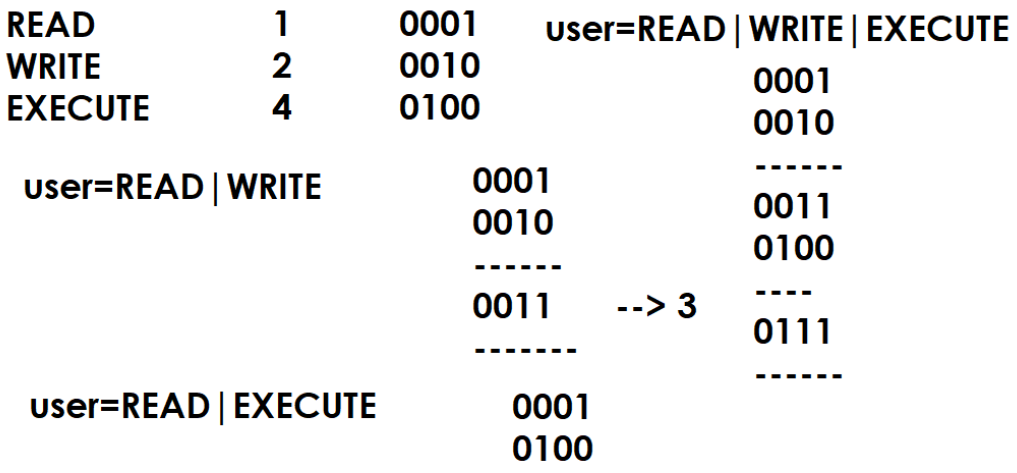
Bitwise (|) or operator

Bitwise | (or) operator is used for applying or gate
Truth table of bitwise or operator

Opr1	Opr2	Opr1 | Opr2
1	0	1
0	1	1
0	0	0
1	1	1



```
>>> a=12
>>> b=15
>>> c=a|b
>>> print(a,b,c)
12 15 15
>>> print(bin(a),bin(b),bin(c))
0b1100 0b1111 0b1111
```



 # Write a program to convert uppercase
 # letter to lowercase using bitwise | or operator

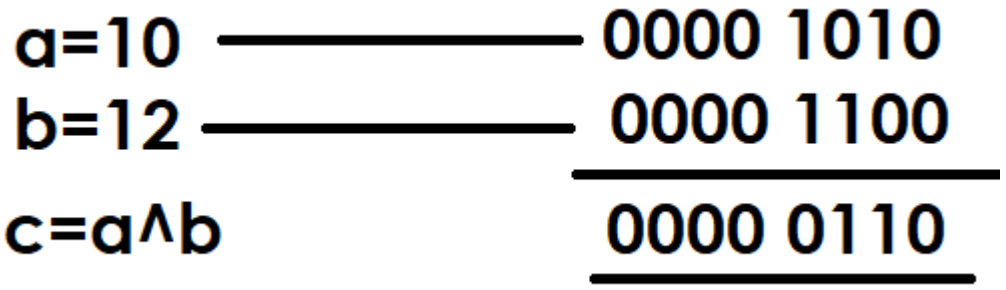
```
ch=input("enter any alphabet in uppercase ")
ch1=chr(ord(ch)|32)
print(ch)
print(ch1)
```

Bitwise ^ (XOR) operator

This operator is used for applying XOR gate

Truth table of XOR operator

Opr1	Opr2	Opr1^Opr2
1	0	1
0	1	1
0	0	0
1	1	0




```
>>> a=10
>>> b=12
>>> c=a^b
>>> print(a,b,c)
10 12 6
>>> print(bin(a),bin(b),bin(c))
0b1010 0b1100 0b110
```

Example:

```
# Write a program to swap two number using bitwise
# ^ (XOR) operator

n1=int(input("Enter first number "))
n2=int(input("Enter second number "))

print(f'Before Swaping {n1},{n2}')
n1=n1^n2
n2=n1^n2
n1=n1^n2
print(f'After Swaping {n1},{n2}')
```

Output

```
Enter first number 10
Enter second number 20
Before Swaping 10,20
After Swaping 20,10
```

~ bitwise not operator

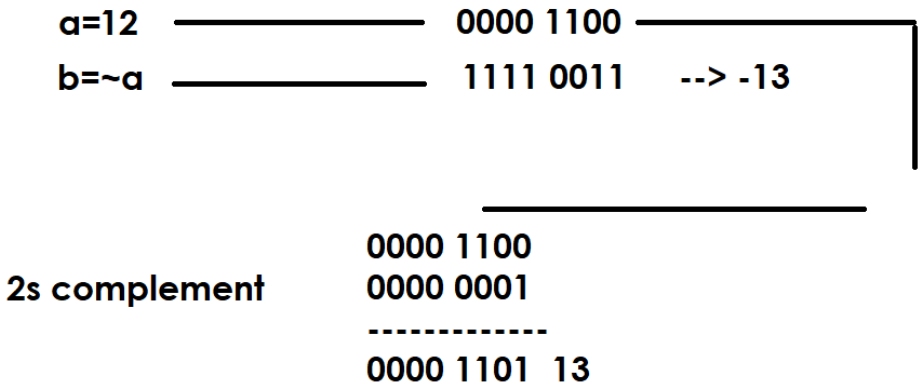
Bitwise not operator used for applying not gate

Bitwise not operator is unary operator

Truth table of bitwise ~ operator

Opr1	~Opr1
1	0
0	1

Bitwise not operator performs –(opr+1)




```
>>> a=12
>>> b=~a
>>> print(a,b)
12 -13
>>> c=~b
>>> print(c)
12
```

Example:

```
# Write a program to convert lowercase to uppercase
# using bitwise operator

ch=input("Enter any alphabet (lowercase) ")
```

```
ch1=chr(ord(ch)+~31)
print(ch)
print(ch1)
```

Output

```
Enter any alphabet (lowercase) a
a
A
```

Shift operators

Shift operators also called bitwise operators.
These operators are used for shifting bits towards left side or right side
By shifting bits towards left side or right side data can be managed.

Python support 2 shift operators

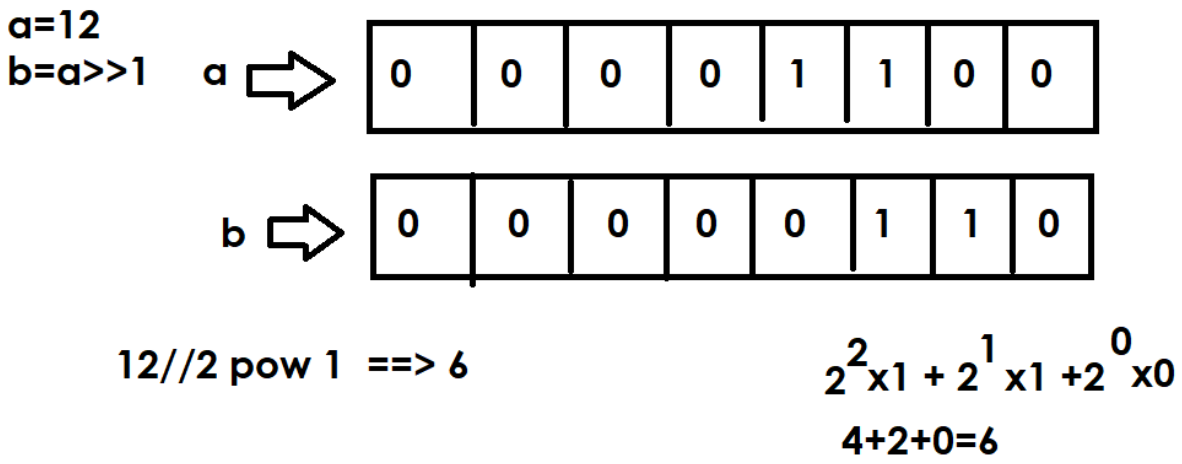
1. >> right shift operator
2. << left shift operator

These both are binary operators and required 2 operands to perform operation.

Right shift operator (>>)

>> right shift operator is used to shift given n bits towards right side
Right sift operator remove or delete n bits from right side.
>> right shift operator perform an arithmetic division operation.

operand>>n operand//2 pow n



Example

```
>>> a=12
>>> b=a>>1
>>> print(a,b)
12 6
>>> print(bin(a),bin(b))
0b1100 0b110
```

Example:

```
# Write a program to divide a given number with 4
# without using division operator

num=int(input("Enter any number "))
res=num>>2
print(num,res)
```

Output

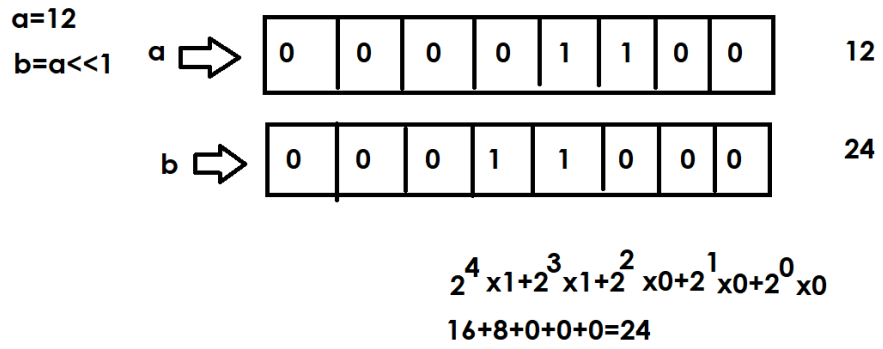
```
Enter any number 16
16 4

Enter any number 5
5 1
```

Left shift operator (<<)

Left shift operator is used for shifting n bits towards the left side.

Left shift operator adds n bits at right side
Left shift operator performs multiplication.

Syntax:
operand<<n operand*2pow n



READ --> 1 0 0
WRITE --> 0 1 0
EXECUTE --> 0 0 1

0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1

```
execute_perm=1
read_perm=execute_perm<<2
write_perm=read_perm>>1
execute_perm=write_perm>>1
```

```
>>> a=12
>>> b=a<<1
>>> print(a,b)
12 24
>>> print(bin(a),bin(b))
0b1100 0b11000
```

Augmented Assignment Statements (OR) Assignment Operators

Augmented assignment statement is nothing but a single operator performing 2 operations.

1. Binary Operation/Binary Operator
2. Assignment/Assignment Operator

These operators are also called update operators.

Note: python programming language does not support ++,--

```
>>> a=5
>>> a++
SyntaxError: invalid syntax
>>> ++a
5
>>> +-a
-5
>>> --a
5
```

+=	a=10 a=a+5 □ a+=5
-=	a=5 a=a-1 □ a-=1
*=	a=10 b=5 a=a*b □ a*=b
/=	a=5 b=2 a=a/b □ a/=b
//=	a=6 b=3 a=a//b □ a//=b
**=	a=5 a=a**2 □ a**=2

%=	a=5 a=a%2 □ a%=2
&=	a=10 b=20 a=a&b □ a&=b
|=	A=10 B=20 A=A|B □ A|=B
^=	A=10 B=20 A=A^B □ A^=B
>>=	A=5 A=A>>1 □ A>>=1
<<=	A=10 A=A<<2 □ A<<=2

```
import time
progress=0
while progress<=100:
    print(progress)
    time.sleep(1)
    progress+=1
```

Example:

```
# Banking Application

# withdraw

accno=1
name="naresh"
balance=56000
tamt=5000
print(f'{accno},{name},{balance},{tamt}')
balance-=tamt # balance=balance-tamt
```

```
print(f'After Withdraw {balance}')
balance+=tamt # balance=balance+tamt
print(f'After Deposit {balance}')
```

Output

```
1,naresh,56000,5000
After Withdraw 51000
After Deposit 56000
```

Walrus operator (:=)

The walrus operator is also called the assignment expression operator.
Walrus operator is introduced in python 3.8 version.
It is a binary operator and requires 2 operands to perform operations.

Syntax:

```
variable=(variable:=expression)operator(variable:=expression)
```

Example:

```
>>> a=5
>>> b=2
>>> c=(d=a+b)*(e=a-b)
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
>>> c=(d:=a+b)*(e:=a-b)
>>> print(c)
21
>>> print(d)
7
>>> print(e)
3
>>> e:=100
SyntaxError: invalid syntax
```

```
>>> e=100
```