CHAPTER -2
PYTHON FUNDAMENTALS
Part - 2

Bimlendu Kumar
PGT Computer Sc.
Kendriya Vidyalaya Garhara

OPERATORS IN PYTHON

Operators:

- Operators are the tokens that trigger some computation / action when applied to variables and other objects in an expression.

Operands:

- Variables and Objects on which operations are applied by operators are called operands.

Types of Operators in Python: Python has following types of operators

1. Unary Operators	**6. Relational Operators**
2. Binary Operators	7. Assignment Operators
3. Bitwise Operators	8. Logical Operators
4. Shift Operators	9. Membership Operators
5. Identity Operators	

UNARY OPERATORS

- Unary Operators are those operators that require one operand to operate upon. Following are some unary operators

Operators Symbol	Meaning
+	Unary Plus
-	Unary Minus
~	Bitwise Complement
not	Logical negation

Unary + Operator:

- This operators precedes an operand.
- Operand must be of arithmetic type.
- The result is the value of the argument.

Example: a= 5 then +a means 5

a=-5 then +a means -5

UNARY OPERATORS

Unary - Operator:

- Like Unary plus Unary minus (-) operators also precedes an operand.
- Operand must be of arithmetic type.
- The result is the negation of its operand value.
- Example: a= 5   then -a means -5

  a=-5   then -a means 5

  a = 0   then –a means 0 (there is no quantity known as -0)

Bitwise Complement Operator (~):

- This operator returns the 1's Complement of Binary value of the numeric operand.
- Suppose a=20, its binary equivalent will be 10100 and ~a will be 01011.

Logical Negation (not): It return the negation of the operand with which it operates. if value given is 1 i.e. True then not True means False and if the Operand is 0 i.e. False then not False means True.

BINARY OPERATORS

- Binary Operators are those operators that requires two operands to operate upon. Following are some binary operators in Python

Operators Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder / Modulus
**	Exponent (Raise to Power)
//	Floor Division

BINARY OPERATORS

1. ADDITION OPERATOR (+)

- The arithmetic binary addition operator (+) adds values of the number types (Boolean Values True and False are treated as number types. Internally True is treated as 1 and False is treated as 0 and hence True + 1 will return 2).

Example

<u>Operations with integers</u>

4 + 20	evaluates in 24 (Literal + Literal)	
a + 5 (where a = 2)	evaluates 7	(Variable + Literal)
a + b (where a =5 and b = -3)	evaluates 2	(Variable + Variable)

<u>Operations with real literals</u>

5.0 + 2.5	evaluates 7.5

<u>Operations With Mixed Operands</u>

5 + 3.5	evaluates 8.5

BINARY OPERATORS

ADDITION OPERATOR (+)

Example

Operations with Strings (Concatenation)

‘4’ + ’20’	evaluates in 420
“20” + “4”	evaluates 204
‘20’ + “25”	evaluates 2025
20 + ‘Mangoes’	Invalid Operation (We can not combine numbers and strings as operands with a + operator)

Operations with Lists (*List in Python are containers that are used to store a list of values of any type. Python Lists ate mutable*, i.e. We can change the elements of list in place.)

When + Operators is applied on two lists, it joins the two lists.

List1= [10, 20, 30];

List2= [15, 25, 35];

List3=List1+ List2 will result in concatenated list [10, 20, 30, 15, 25, 35]

BINARY OPERATORS

ADDITION OPERATOR (+)

Example

Operations with tuples (A standard data type in Python that can store a sequence of values belonging to any type. Tuples are immutable sequences of Python i.e. its value of elements can not be changed in place).

```
>>>t1=(10, 20, 30)
>>>t2=(“Mango”, “Apple”, “Banana”)
>>>t3=t1+t2
```

It will concatenate the two tuples. It is clear from the following command

```
>>>t3↲
```

Output

```
(10, 20, 30, “Apple”, “Mango”, “Banana”)
>>>t1+2              #invalid as integer can not be added to tuple
```

+ Operator when used with tuples requires that both the operands must be of tuple types.

BINARY OPERATORS

2. SUBTRACTION OPERATOR (-)

The subtraction operator subtracts the second operand from first operand.

Example:

14 – 3	evaluates to 11
(-14) – 3	evaluates to -17
14 – (-3)	evaluates to 17
a – b (when a=10 and b=5)	evaluates to 5

a – b (when a=10 and b= -20) evaluates to 30

3. MULTIPLICATION OPERATOR (*)

The multiplication operator * multiplies the values of its operands.

3 * 4	evaluates to 12
a * b (when a=5 and b=10)	evaluates to 50
2.5 * 2.5	evaluates to 6.25

The operand may be integer or floating point number types.

Python also offers * as Replication Operator when used with strings. (both operand can't string)

Example "@" * 5 = "@@@@@" "abc" * 2 = "abcabc" "1" * 5 = "11111"

BINARY OPERATORS

4. DIVISION OPERATOR (/)

The / operator in python divides its first operand with second operand and always returns result as a float value.

Example:

10 / 4	evaluates to 2.5
-10/ 2	evaluates to -5.0
10/-2	evaluates to -5.0
-10/-2	evaluates to 5.0
7/2.5	evaluates to 2.8
15.5 / 1.5	evaluates to 9.0
-7/4	evaluates to -1.75
7/-4	evaluates to -1.75
-7/-4	evaluates to 1.75

BINARY OPERATORS

5. FLOOR DIVISION OPERATOR (//)

Python also offers another operator // which performs the floor division. The floor division is the division in which only the whole part of the result is given in the output and the fractional part is truncated.

Example:

10 // 4	evaluates to 2
-10// 4	evaluates to -3
10//-4	evaluates to -3
-10//-4	evaluates to 2
10.25//4	evaluates to 2.0
10 // 4.0	evaluates to 2.0

BINARY OPERATORS

6. MODULUS OPERATOR (%)

The % operator finds the modulus i.e. remainder of the first operand relative to the second operand. I.e. it provides remainder of dividing the first operand by the second operand. Unlike C, C++ Python accepts both integer and float as its operand. It may get Boolean Operand True and False also.

Example:

10 % 4	evaluates to 2
4%10	evaluates to 4
10.0%4	evaluates to 2.0
-7.25%4.24	evaluates to -1.25
True % 2	evaluates to 1
False % 2	evaluates to 0

BINARY OPERATORS

7. EXPONENTIAL OPERATOR (**)

The ** operator performs exponentiation calculation. It returns the result of a number raised to a power.

2**3	evaluates to 8
2**10	evaluates to 1024
2.5**3	evaluates to 15.625
2.5**2.5	evaluates to 9.882117688026186
2**-2	evaluates to 0.25

RELATIONAL OPERATORS

- The relational operator determine the relation among different operands.
- Python provides six relational operators for comparing values thus also called comparison operators.
- If the comparison is true, the relational expressions results in Boolean value True and to Boolean value false if the comparison is false.
- These operators as below:

<	Less than
<=	Less than or Equal to
>	Greater than
>=	Greater than or Equal to
==	Equal to
!=	not Equal to

RELATIONAL OPERATORS

- Relational Operators work with nearly all types of data in Python such as numbers, string, lists, tuples etc.

Principle of functioning of Relational Operators:

- For numeric types the values are compared after removing trailing zeros after decimal point from a floating point number. For example 4 and 4.0 will be treated equal (After removing trailing zeros from 4.0 it becomes 4).
- Strings are compared on the lexicographical order i.e. dictionary order.
- Capital alphabets are considered lesser than small alphabets.
- Two lists or two tuples are treated similar if they have same elements in same order.
- Boolean True is equal to 1 and Boolean False is equal to 0 for comparison purposes.

RELATIONAL OPERATORS

P	Q	P<Q	P<=Q	P==Q	P>Q	P>=Q	P!=Q
3	3.0	False	True	True	False	True	False
6	4	False	False	False	True	True	True
'A'	'A'	False	True	True	False	True	False
'a'	'A'	False	False	False	True	True	True
"God"	"Godess"	True	True	False	False	False	True

RELATIONAL OPERATORS

USE OF RELATIONAL OPERATORS WITH FLOATING POINT NUMBERS:

- While using floating point numbers with relational operators, we must keep in mind that floating point numbers are **approximately** presented in memory in binary form up to the allowed precision (15 digits precision in case of python). This approximation may yield unexpected result if you are comparing floating point numbers especially for equality (= =). Numbers such as 1/3 can not be fully represented in binary as it yields 0.333333… etc. and to represent it in binary some approximation is done internally.
- hence th3 result of the expression 0.1 + 0.1 + 0.1 = = 0.3 results in False.
- When we use Print(0.1 + 0.1 + 0.1) it yields 0.30000000000000004 and not 0.3

RELATIONAL OPERATORS

USE OF RELATIONAL OPERATORS WITH ARTIHMETIC OPERATORS

- Relational operators have lower precedence than that of art=ithmetic operators

hence

a + 5 < c – 2

corresponds to

(a + 5) < (c-2) and not

a + (5<c)-2.

- One silly mistake generally we do while working with relational operator is that instead of operator == (equality operator) we use = (assignment operator). This results in unexpected output.

IDENTITY OPERATORS

- There is two identity operators in Python. **is** and **is not**.
- The identity operators are used to check if both the operands reference the same memory object.
- It means the identity operators compares the memory location of two objects and return True of False accordingly.

Operator	Usage	Description
is	a is b	Return True if a and b both pointing to same memory location i.e. same object otherwise False.
is not	a is not b	Returns True if a and b both pointing to different memory locations i.e. different objects otherwise False.

IDENTITY OPERATORS

At python command prompt type the following and see the result

```
>>>a=10
>>>b=10
>>>a is b
True
>>>id(a)
1606149024
>>>id(b)
1606149024
>>>b=40;
>>>a is b
False
```

IDENTITY OPERATORS

At python command prompt type the following and see the result

```
>>>a=235
>>>b=240
>>>c=235
>>>a is c
True
>>>print(“Address of variable a = “, id(a), “address of c = “,id(c))
address of a =  1644294576 and address of c =  1644294576
```

Both a and c are having same address hence the output is True.

```
>>>a is b
False
>>>print(“Address of variable a = “, id(a), “address of b = “,id(c))
address of a =  1644294576 and address of b =  1644294656
```

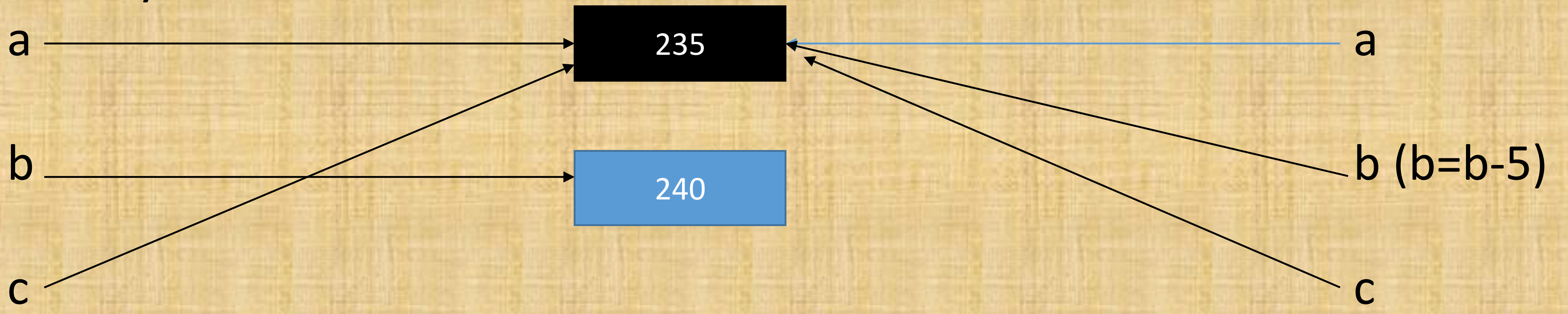
Address of a and b is not same hence the output is false

IDENTITY OPERATORS

```
>>>b=b-5
>>>a is b
True
>>>print("Address of variable a = ", id(a), "address of b = ",id(b),, "and address of c = ",id(c))
Address of a = 1644294576 address of b = 1644294576 and address of c = 1644294576
```

On subtracting 5 from b it becomes 235. In python each and every constant is also an object and it is stored at some memory location. For each unique constant there is only one address. So what happened here is

initially



So now reference to constant object 240 has been lost and it is cleard from memory automatically.

IDENTITY OPERATORS

is not operator is opposite to **is** operator. It returns True when both the operands are not referencing the same memory address.

```
>>A = 200
>>>B=150
>>>A is not B
True
```

Since both the operand A and B nor=t referencing the same memory location.

it can be seen from output of following python command

```
>>>print(“Address of A = “, id(A), “and Address of B = “,id(B))
Address of A =  1644294016 and Address of B =  1644293216
```

Now subtract value 50 from A and do the following.

```
>>>A=A-50
>>A is not B
False
>>>print(“Address of A = “, id(A), “and Address of B = “,id(B))
Address of A =  1644293216 and Address of B =  1644293216
```

Equality (= =) and Identity (is) Operator-Important Relation

```
>>>A=200
>>>B=200
>>>print(A,B)
Output
200   200
>>>A is B
Output
True
>>>A==B
output
True
```

Note: When two operands are referring to the same value i.e. same memory address the **is** operator returns True. It implicitly means that the equality operator **(==)** will also return True and It is clear from above output. But it not always true.

Equality (= =) and Identity (is) Operator-Important Relation

There are some cases where we will find that the two objects are having just the same value, equality operator (==) returns True whereas **is** Operator returns False.

```
 >>>S1="ABC"
>>>S2=input("Enter a String")
```

and we type ABC as input

```
>>>S1==S2
```

Output

```
True
```

```
>>>S1 is S2
```

Output

```
False
```

The String variable S1 and S2 both are having same value ABC but the Euqality Operator (==) returns True for S1==S2 and identity Operator (is) returns False for S1 is S2.

Equality (= =) and Identity (is) Operator-Important Relation

Now type the following

```
>>>S3="ABC"
>>>S1==S3
```

<u>output</u>

```
True
>>>S1 is S3
```

<u>output</u>

```
True
```

Let us take another Example for complex numbers

```
>>>F1=2+3.5J
>>>F2=2+3.5J
>>>F1==F2
True
>>>F1 is F2
False
```

Different Output with Equality and Identity Operator

Equality (= =) and Identity (is) Operator-Important Relation

Let us Take third Example related to floating Point Literals

```
>>>K=3.5
>>>L=float(input(“Enter a Real number:- “))
```

Enter the same value i.e.3.5 from keyboard for variable L.

```
>>>K==L
```

output

True

```
>>>K is L
```

output

False

Different Output

One must check that the two variable is referring the same address of not. If not, with same value for two different variables, Equality operator **==** operator will return **True** while identity operator **is** will return **False**.

Reasons behind Returning False by Identity Operator is that Python creates two different objects for the following cases

1. Input of string from the console / Keyboard
2. Writing Integers with many digits (big integers)

LOGICAL OPERATORS

Let us Take third Example related to floating Point Literals

```
>>>K=3.5
>>>L=float(input("Enter a Real number:- "))
```

Enter the same value i.e.3.5 from keyboard for variable L.

```
>>>K==L
```

output

```
True
```

```
>>>K is L
```

Different Output

output

```
False
```

One must check that the two variable is referring the same address of not. If not, with same value for two different variables, Equality operator == operator will return **True** while identity operator **is** will return **False**.

Reasons behind Returning False by Identity Operator is that Python creates two different objects for the following cases

1. Input of string from the console / Keyboard
2. Writing Integers with many digits (big integers)

BITWISE OPERATORS

- These are the Operators that work on individual bits rather than entire entity.

Operators Symbol	Meaning
&	Bitwise AND
^	Bitwise Exclusive OR (XOR)
| (Pipe Symbol)	Bitwise OR
~	Complement Operator

IDENTITY OPERATORS

- These are the Operators that work on individual bits rather than entire entity.

Operators Symbol	Meaning
&	Bitwise AND
^	Bitwise Exclusive OR (XOR)
| (Pipe Symbol)	Bitwise OR

Thanks for Watching This Presentation