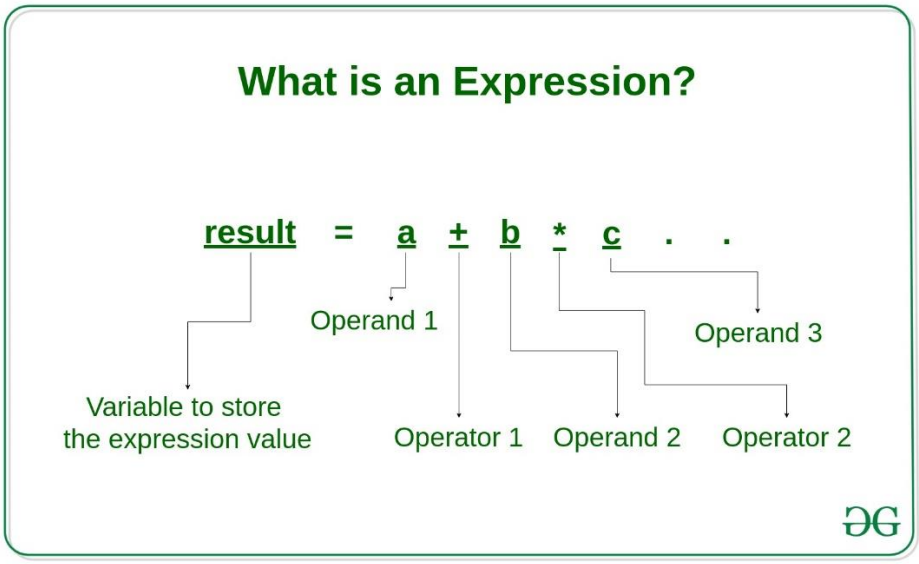
EXPRESSIONS

A statement that uses set of operations to perform action is said to be an expression, it uses more than one operator in one statement



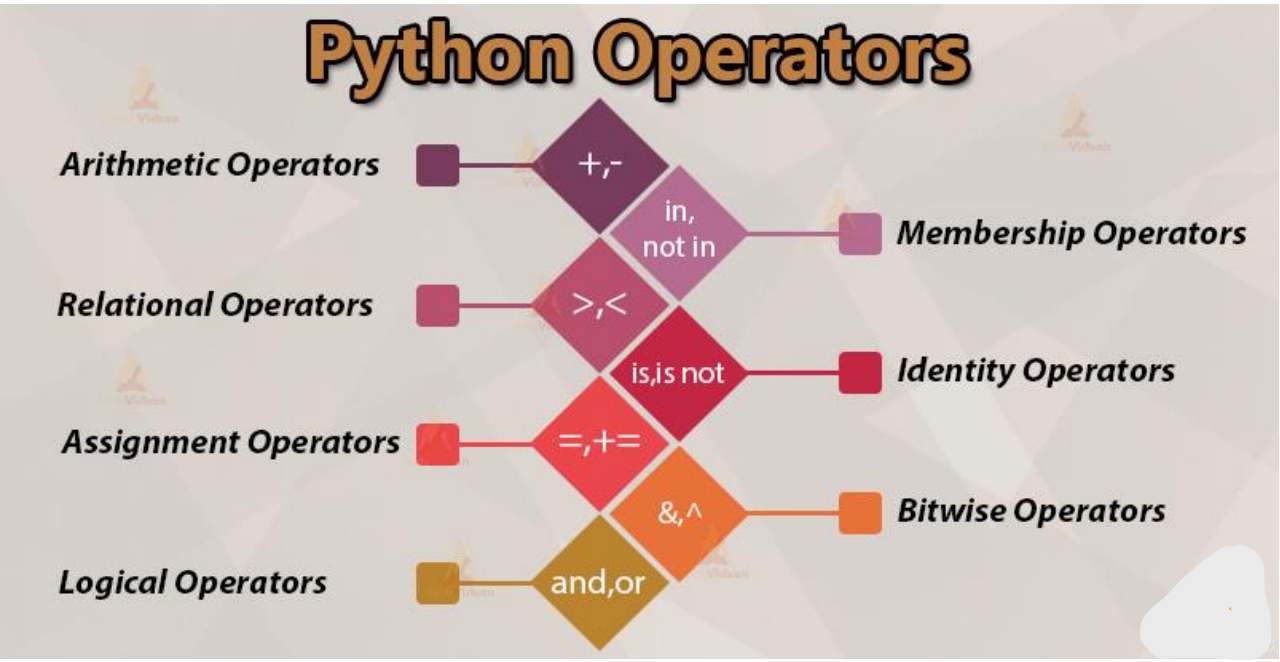
Invalid expressions	Explanation
1 + 2 + a + 4	Operand must be a number, a is not a number
(1 + 3))	Unbalanced parentheses
(1 + 3) * ((2)	Unbalanced parentheses
33.3 + + 1	Operators require a left and right numeric operands
(33) (1)	Operands must be separated by an operator
) (	No preceding left parenthesis for the right parenthesis
( )	No operand within the bracket

Examples of valid expressions with explanation and the evaluated result are given below:

valid expressions	Reasons	Result of evaluation
1.2 + -3	Operand operator operand	-1.8
((1 + 3))	Balanced parentheses, operand operator operand	4
(1 + 3) * ((2))	Balanced parentheses, operand operator operand	8
33.3 + -2 + 1	Operand operator operand	32.3
(33) / (-1)	Operand operator operand	-33.0
(33) // (1)	Operand operator operand	33

OPERATORS

- Operators are used for calculation and manipulation of data between two values or performs conditions, types of operators are given below



- **ARITHMETIC OPERATORS:**
  Operators that are specially used for calculations using numbers as integer.
  For example:
  - 2+5
  - 'hello'+'world' (string uses only this symbol **+** as concatenation)
  - 4-2
  - 5//10
  - 2**3
  - 6/2
  - 8%10

Name	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%
Exponentiation	**
Floor Division	//

- **RELATIONAL OPERATORS:**

  It is used to describe values as of using signs **(<, >, <=, >=)** it shows True or False conditions for given statement.

  For example:

  Print(5>8)

  Output:

  False

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

- **ASSIGNMENT OPERATORS:**

  It's just a shorter way or logical way of arithmetic operator it differs only in written format but functionally it is same.

  For example:

  Let's say a= 10 we want to add 15 more to a in that case what we do is

  Code:

```
a= 10
a=10+15
print(a)
```

  Output:

```
25
```

  (Or by using assignment operator)

  Code:

```
a=10
a+=15
print(a)
```

  Output:

```
25
```

Similarly we have different operators that is combination of assignment and arithmetic

Operator	Example	Equivalent Expression (m=15)	Result
=	y = a+b	y = 10 + 20	30
+=	m +=10	m = m+10	25
-=	m -=10	m = m-10	5
*=	m *=10	m = m*10	150
/=	m /=10	m = m/10	1.5
%=	m %=10	m = m%10	5
**=	m**=2	m = m**2 or $m = m^2$	225
//=	m//=10	m = m//10	1

- **LOGICAL OPERATORS:**

  Operator that performs between to relational operators or between two conditions with the help of and, or, not

  and = if any one value is false between two values it gives output false. It must have both values true

  syntax: a=10
  b=20
  print(a<b and b>a)

  or = if any one value is true it gives true as output

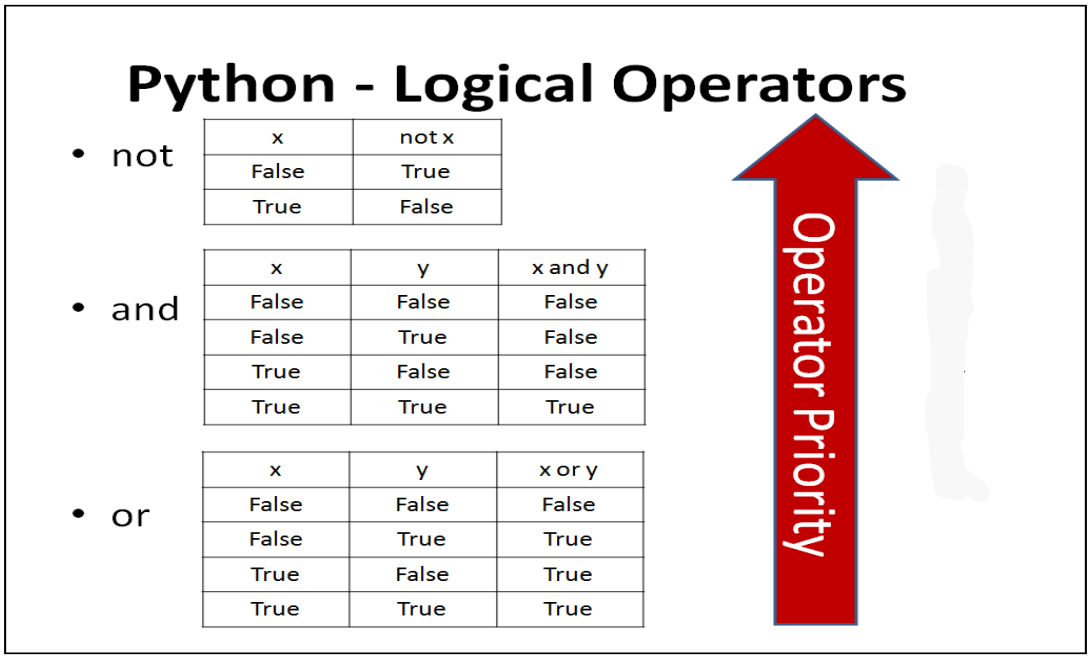
  syntax: a=10
  b=20
  print(a<b and b>a)

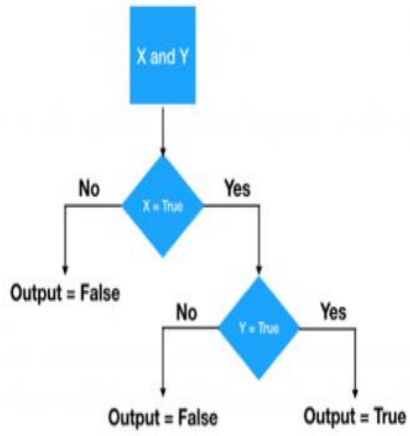
not = it helps to convert condition True to False and False to True (note: first letter must be capital for True and False)

syntax: a=True

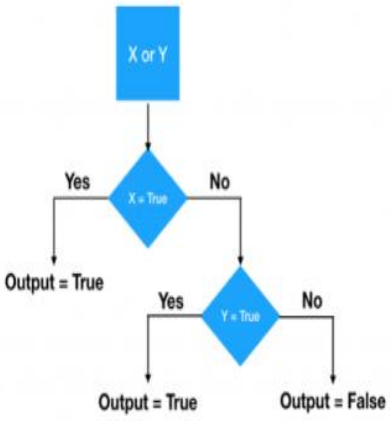
print(not a)

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\TECHtroniX\OneDrive\Desktop\test\hj.py ==========
>>> a=10
>>> b=20
>>> print(a<b and a>b)
False
>>> print(a<b or a>b)
True
>>> #for both true in and
>>> print(a<b and b>a)
True
>>> #for both false in or
>>> print(a<b or b>a)
True
>>> #using not
>>> x=True
>>> x
True
>>> #above we just assigned True to x and printed x
>>> #now by using not we can change True to False
>>> not x
False
>>> x
True
>>> #here we can see x is True, not x is False
```

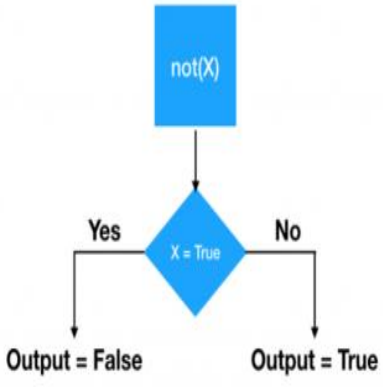




Logical AND Operator Flow Chart



Logical OR Operator Flow Chart



Logical NOT Operator Flow Chart

- **Membership operator:**

  It is used for reading sequence or in general reads every single letter in word and displays result as True or False if requested letter is in word or not. By using **in, not in**

  For example:

  Take a word python and see that t is present python or not

  (for in) Syntax: a=’python’

  b=’t’

  print(b in a)

  Output:

  True

  (for not in) Syntax: a=’python’

  b=’t’

  print(b not in a)

  Output:

  False

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\TECHtroniX\OneDrive\Desktop\test\hj.py ==========
>>> a="python"
>>> b="t"
>>> print(b in a)
True
>>> #here we know that python has t and is true
>>> print(b not in a)
False
>>> #it is false because python has t in it and we are printing it is not
>>> #lets have another example
>>> c="j"
>>> #we know python does not have letter j in it
>>> print(c not in a)
True
>>>
```

- **IDENTITY OPERATORS:**

  It uses is, is not to check the values of two variables if two variables have same value it gives True otherwise gives False.

  For example:

  For is

  Syntax: a=26
  b=30
  c=26
  print(a is b)
  print(a is c)

  Output:

  False
  True

  For is not

  Syntax: print(c is not a)

```
>>> ========== RESTART: Shell ==========
>>> a=26
>>> b=30
>>> c=26
>>> print(a is b)
False
>>> print(b is a)
False
>>> # above we get false because a and b values are different
>>> print(a is c)
True
>>> print(c is a)
True
>>> # here we get both true because a and c has same values
>>>
>>> #for is not
>>> print(a is not b)
True
>>> #statement is true as a is not same as b
>>> print(c is a)
True
>>> print(c is not a)
False
>>> #c and a are same but if we print c is not a it gives false
```

- **BITWISE OPERATOR:**

  Bitwise operator works with binary numbers.

  - Binary number consists of 0's and 1's can be found using **print(bin(10))**
  - For example
    - For 2 binary number is 0010
    - Let's see how it works

n…..16	8	4	2	1
n……..0	0	0	1	0

- Above 0's and 1's in table are binary numbers that represents whole number
- Similarly we don't have 3 to have 3 what we do is

8	4	2	1
0	0	1	1

- In above table 2 and 1 have 1's and other has 0's
- Binary number for above 3 has 0011
- 2+1=3 hence 1 is marked only at 1 and 2 other than that is 0's

Let's see types of bitwise operator:

(note: it is different from logical operator)

Types of Bitwise Operators

Operator	Name	Example	Result
&	Bitwise AND	6 & 3	2
|	Bitwise OR	10 | 10	10
^	Bitwise XOR	2^2	0
~	Bitwise 1's complement	~9	-10
<<	Left-Shift	10<<2	40
>>	Right-Shift	10>>2	2

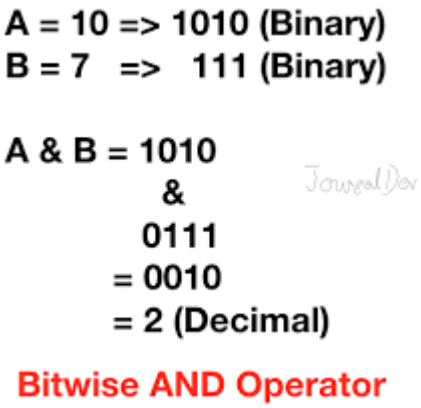
1. **Bitwise AND:**
   - It uses symbol **'&', ** where it takes two values for operation
   - Two values are considered as in binary number

Here binary numbers are

1010 (10) **for 1 and 1 only**

0111 (7) **it gives 1 otherwise**

0010 (2) **it gives 0**



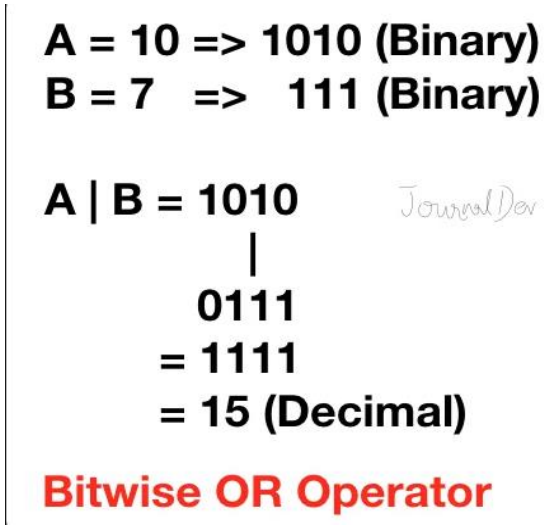
2. **Bitwise OR:**
   - It uses symbol '|', where it takes two values for operation
   - Two values are considered as binary number
   - If any one binary number is 1 it takes 1 as result

Here binary numbers are

1010 (10) **for any occurrence**

0111 (7) **of 1 it gives 1, it gives**

1111 (15) **0 only when both are 0's**



3. **Bitwise XOR:**
   - It uses symbol '^', where it takes two values for operation
   - Two values are considered as binary number

- For different it takes 1 otherwise it takes 0 for same binary number occurrence

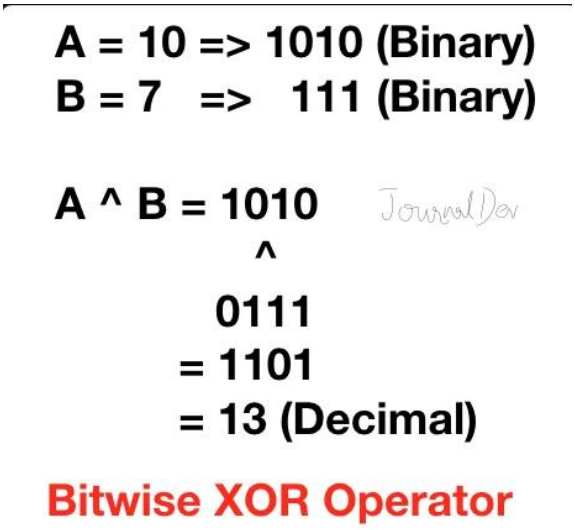
Here binary numbers are

1010 (10) **for any occurrence same**

0111 (7) **1 or same 0 it gives 0, it gives**

1101 (13) **1 only when both are different**

We can see that 0 is occurred when both are 1(same)



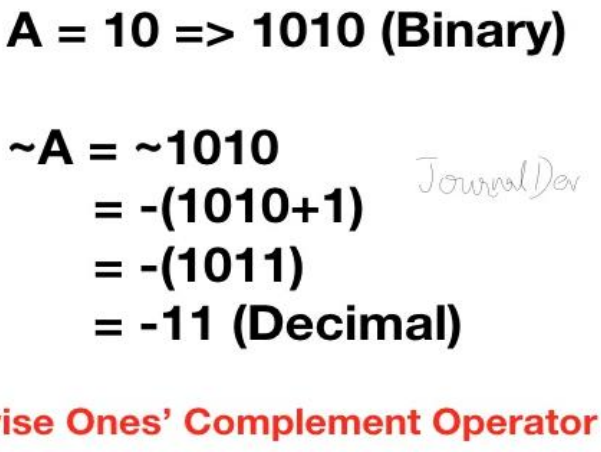
4. **Bitwise 1's compliment:**
   - It uses symbol **'~'**, where it takes one value for operation
   - value is considered as binary number
   - And 1 is added to that binary number with negative sign

Here binary number of 10 is

1010 (10)

+1 (1's compliment)

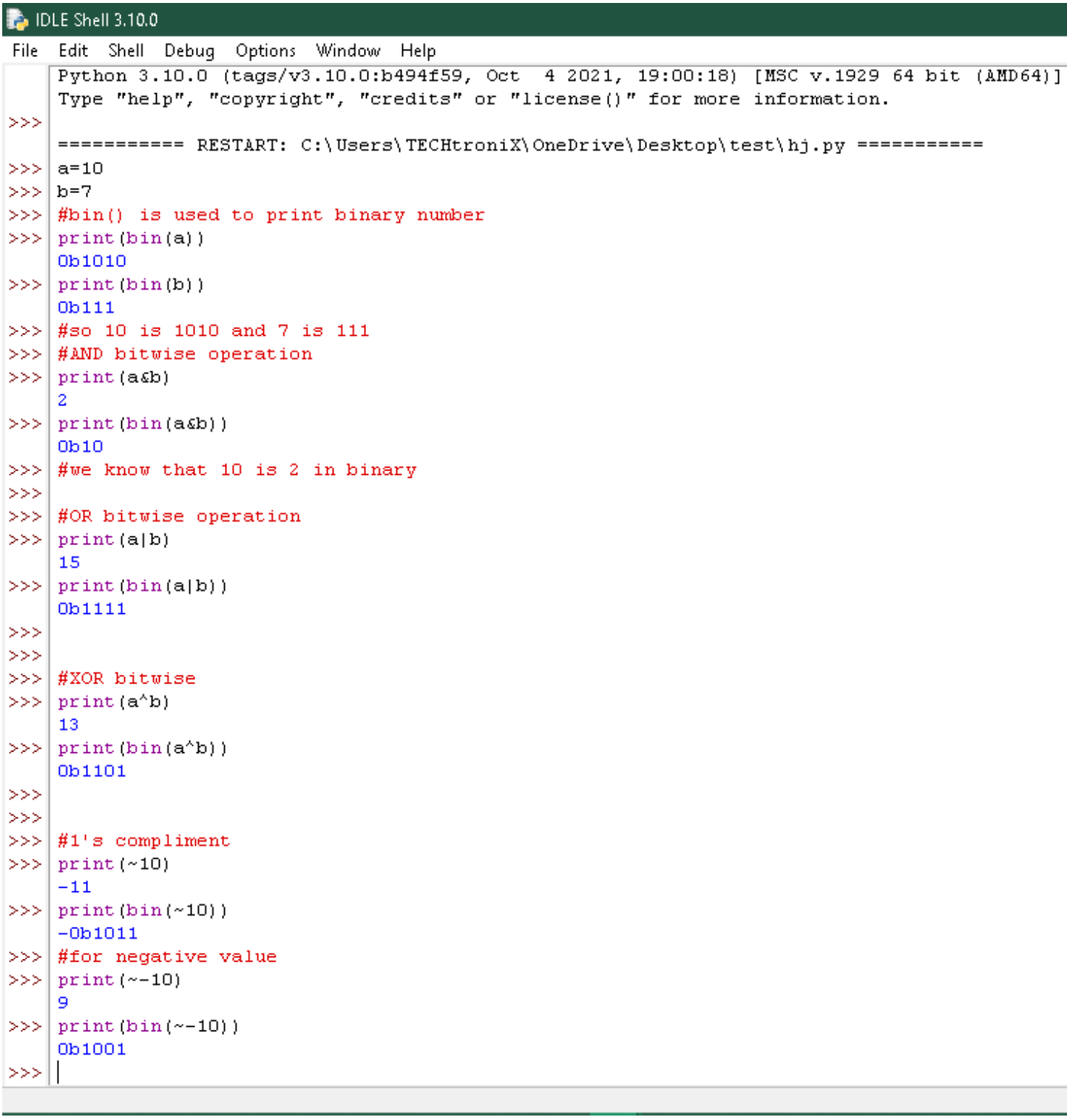
1011 (11)



Note: for every positive number **~10** it increments 1 with negative sign that is -11

And similarly, if we give negative number **~-10** it decrements 1 with positive number as 9

As shown in picture below…..



5. **Bitwise left-shift:**

- Python bitwise left shift operator shifts the left operand bits towards the left side for the given number of times in the right operand. In simple terms, the binary number is appended with 0s at the end.

Here binary of 10 is:

1010 for left shift depending

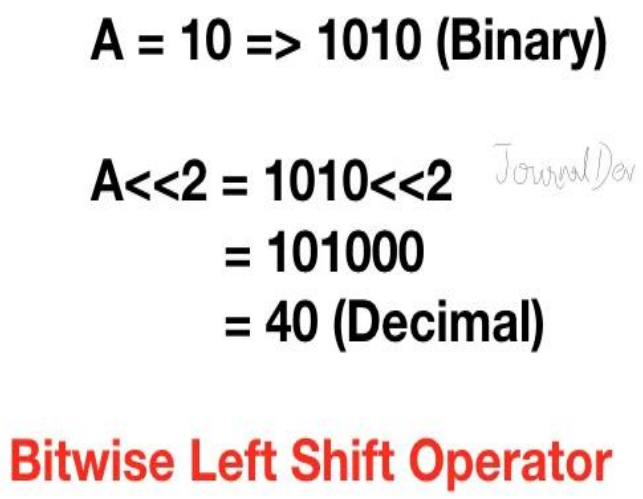
On number given it adds

Zeroes on left hand side of

Binary 10

1010<<2 (adding two zeroes)

101000 (40)



6. **Bitwise right-shift:**

- Python right shift operator is exactly the opposite of the left shift operator. Then left side operand bits are moved towards the right side for the given number of times. In simple terms, the right side bits are removed.

Here binary of 10 is:

1010 for right shift depending

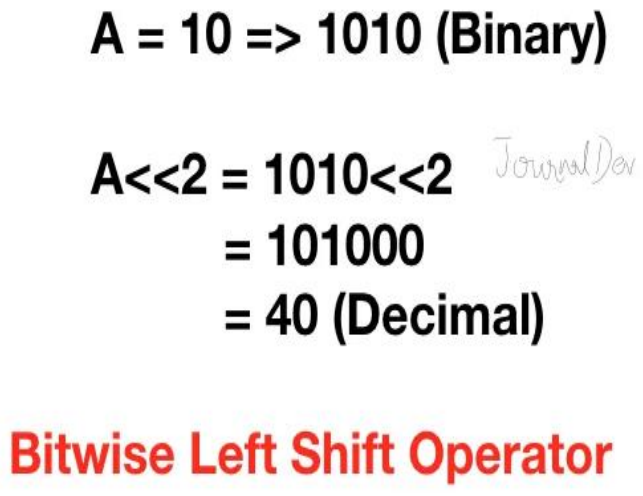
On number given it adds

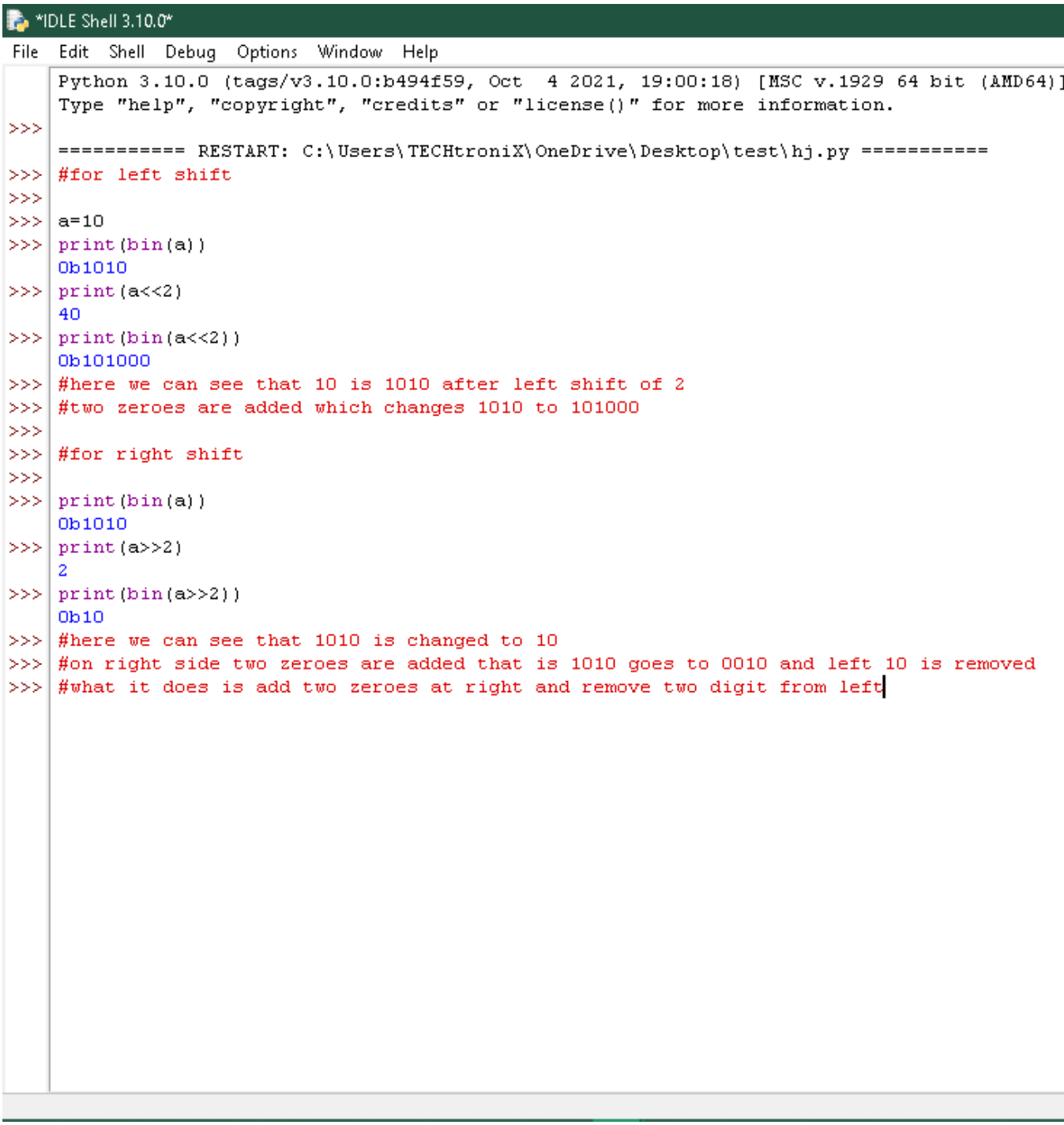
Zeroes on right hand side of

Binary 10

1010>>2 (adding two zeroes)

0010 (40)




IDLE Shell 3.10.0
File Edit Shell Debug Options Window Help
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\TECHtroniX\OneDrive\Desktop\test\hj.py ==========
>>> #for left shift
>>>
>>> a=10
>>> print(bin(a))
0b1010
>>> print(a<<2)
40
>>> print(bin(a<<2))
0b101000
>>> #here we can see that 10 is 1010 after left shift of 2
>>> #two zeroes are added which changes 1010 to 101000
>>>
>>> #for right shift
>>>
>>> print(bin(a))
0b1010
>>> print(a>>2)
2
>>> print(bin(a>>2))
0b10
>>> #here we can see that 1010 is changed to 10
>>> #on right side two zeroes are added that is 1010 goes to 0010 and left 10 is removed
>>> #what it does is add two zeroes at right and remove two digit from left