Python Programming

Operators and Expressions

Outline

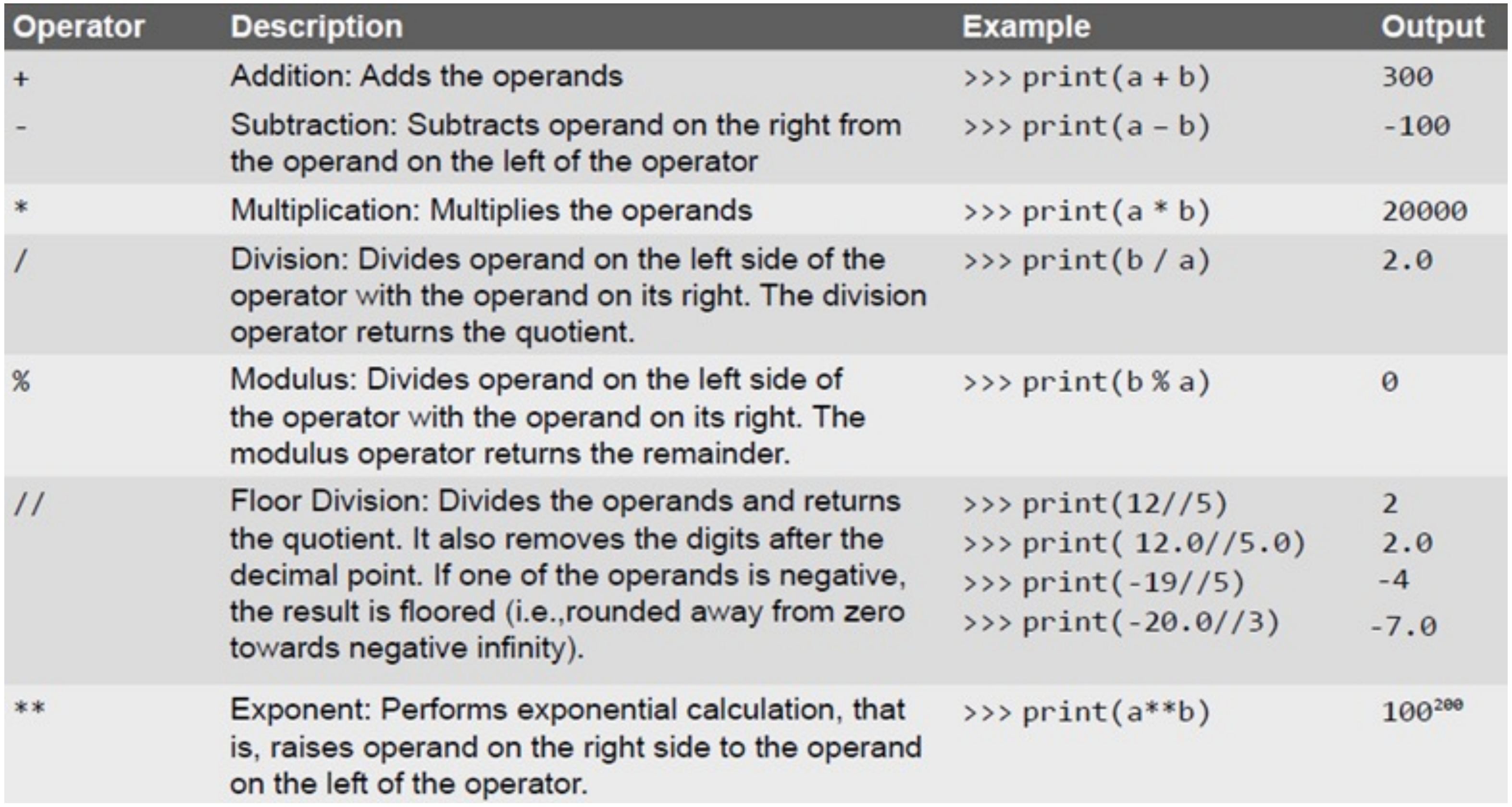
- Types of Operator
- Precedence of Operators
- Associativity of Operators

Operators and Expressions

- Operators are the constructs, which can manipulate the value of operands.
- Consider the expression 4 + 5 = 9.
- Here, 4 and 5 are called operands and + is called the operator.
- **Types of Operator**
  - Arithmetic Operators
  - Comparison (Relational) Operators
  - Assignment Operators
  - Logical Operators
  - Bitwise Operators
  - Membership Operators
  - Identity Operators

Operators and Expressions

- **Arithmetic Operators**
- If variable a holds the value 100 and variable b holds the value 200, then



Operator	Description	Example	Output
+	Addition: Adds the operands	`>>> print(a + b)`	300
-	Subtraction: Subtracts operand on the right from the operand on the left of the operator	`>>> print(a – b)`	-100
*	Multiplication: Multiplies the operands	`>>> print(a * b)`	20000
/	Division: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient.	`>>> print(b / a)`	2.0
%	Modulus: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder.	`>>> print(b % a)`	0
//	Floor Division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (i.e.,rounded away from zero towards negative infinity).	`>>> print(12//5)`	2
		`>>> print( 12.0//5.0)`	2.0
		`>>> print(-19//5)`	-4
		`>>> print(-20.0//3)`	-7.0
**	Exponent: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator.	`>>> print(a**b)`	100^{200}

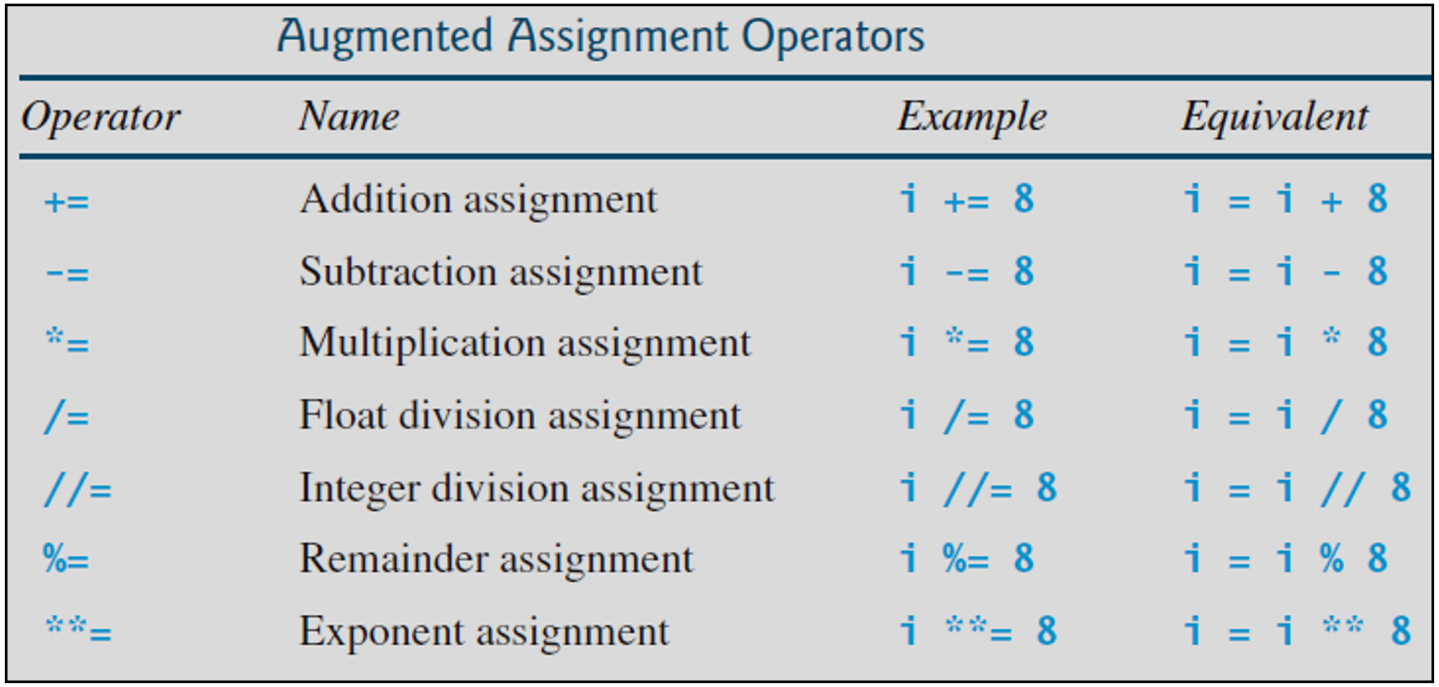
Operators and Expressions

- **Comparison (Relational) Operators**
- If variable a holds the value 100 and variable b holds the value 200, then

Operator	Description	Example	Output
`==`	Returns True if the two values are exactly equal.	`>>> print(a == b)`	`False`
`!=`	Returns True if the two values are not equal.	`>>> print(a != b)`	`True`
`>`	Returns True if the value at the operand on the left side of the operator is greater than the value on its right side.	`>>> print(a > b)`	`False`
`<`	Returns True if the value at the operand on the right side of the operator is greater than the value on its left side.	`>>> print(a < b)`	`True`
`>=`	Returns True if the value at the operand on the left side of the operator is either greater than or equal to the value on its right side.	`>>> print(a >= b)`	`False`
`<=`	Returns True if the value at the operand on the right side of the operator is either greater than or equal to the value on its left side.	`>>> print(a <= b)`	`True`

Operators and Expressions

- **Assignment Operators**
- Assigns values from right side operands to left side operand i.e. c = a + b assigns value of a + b into c
- Python allows you to combine assignment operators with other operators known as an augmented (or compound) assignment operator.



Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Float division assignment	`i /= 8`	`i = i / 8`
`//=`	Integer division assignment	`i //= 8`	`i = i // 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`
`**=`	Exponent assignment	`i **= 8`	`i = i ** 8`

Operators and Expressions

- **Logical Operators**
- AND (&&)
- OR (||)
- NOT(Used to reverse the logical state of its operand) (!)
- Assume variable e_1 holds True/False and variable e_2 holds True/False then truth table of logical operators:

e_1	e_2	e_1 and e_2	e_1 or e_2	not e_1
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Operators and Expressions

- **Logical AND (&&) operator** is used to simultaneously evaluate two conditions or expressions with relational operators. If expressions on both the sides (left and right side) of the logical operator are true, then the whole expression is true.
- For example, If we have an expression (a>b) && (b>c), then the whole expression is true only if both expressions are true. That is, if b is greater than a and c.
- **Logical OR (||) operator** is used to simultaneously evaluate two conditions or expressions with relational operators. If one or both the expressions of the logical operator is true, then the whole expression is true.
- For example, If we have an expression (a>b) || (b>c), then the whole expression is true if either b is greater than a or b is greater than c.

Operators and Expressions

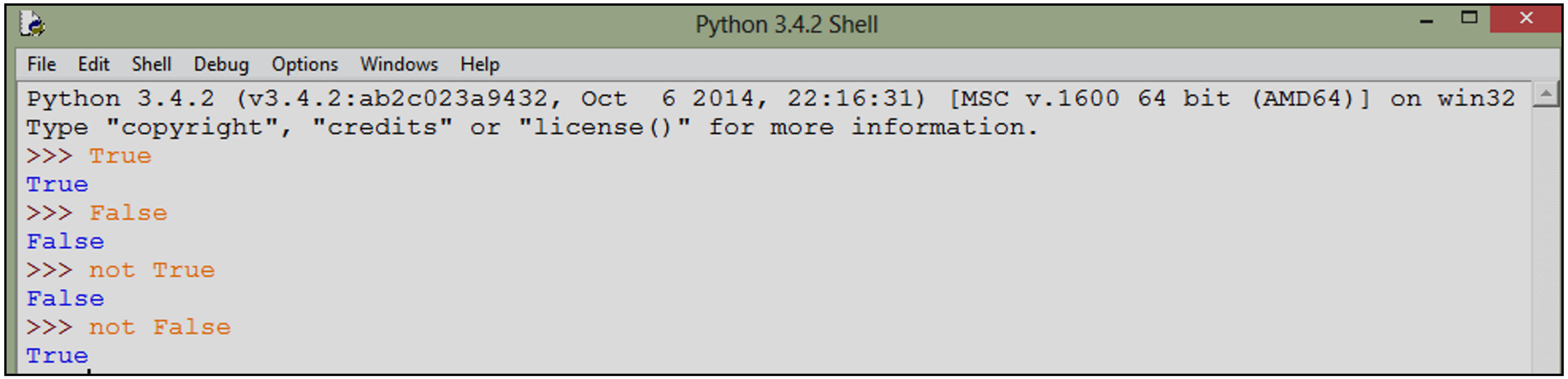
- **Logical not (!) operator** takes a single expression and negates the value of the expression. Logical NOT produces a zero if the expression evaluates to a non-zero value and produces a 1 if the expression produces a zero. In other words, it just reverses the value of the expression.
- For example, a = 10, b b = !a; Now, the value of b = 0. The value of a is not zero, therefore, !a = 0. The value of !a is assigned to b, hence, the result.

Operators and Expressions

- **Boolean Operators**
- Arithmetic expressions evaluate to numeric values; a Boolean expression, sometimes called a predicate, may have only one of two possible values: false or true.
- True and False are special values that belong to the **class bool**, they are not strings.
- The term Boolean comes from the name of the British mathematician George Boole. A branch of discrete mathematics called Boolean algebra is dedicated to the study of the properties and the manipulation of logical expressions.
- **and, or and not** are the three basic Boolean operators.
- **Boolean** operators are also called logical operators.
- not operator has highest precedence followed by and then or.

Operators and Expressions

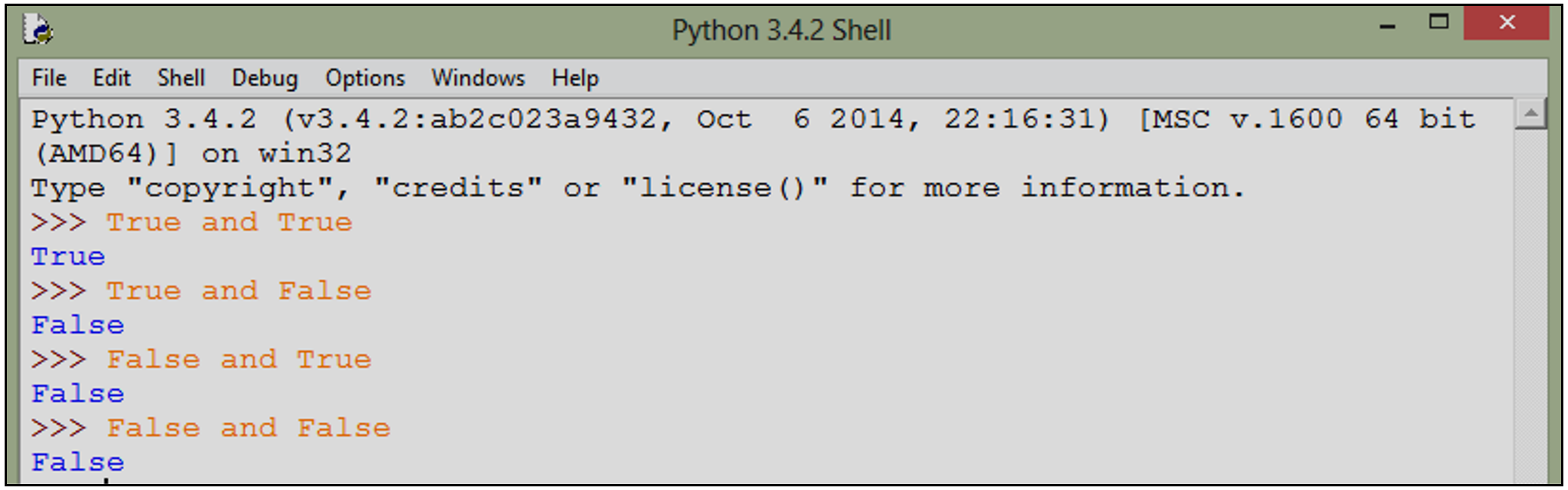
- not operator



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> True
True
>>> False
False
>>> not True
False
>>> not False
True
```

Operators and Expressions

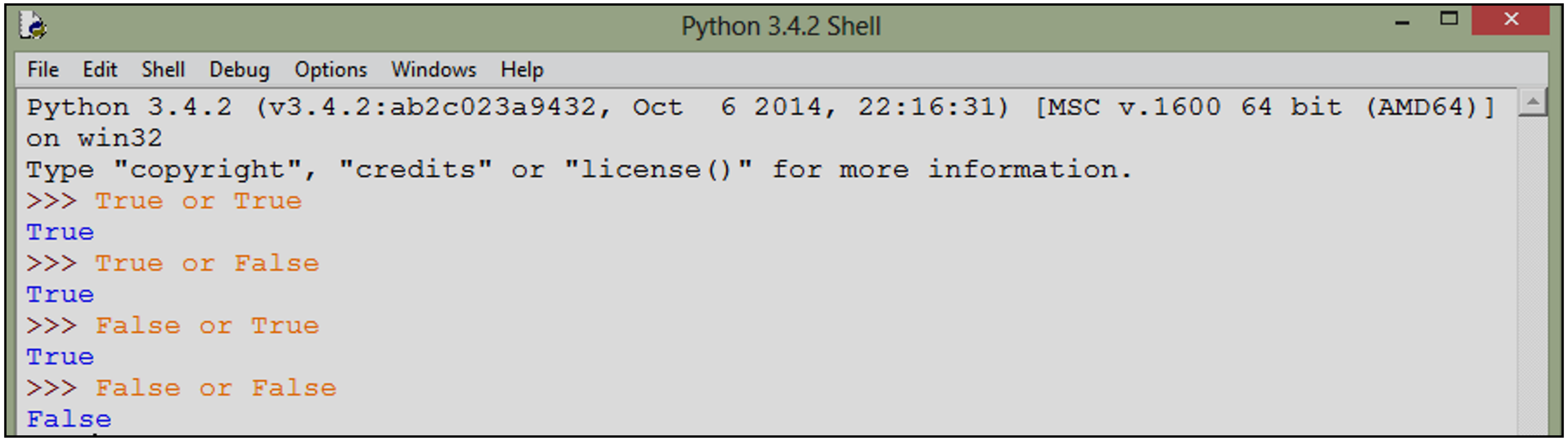
- and operator



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31) [MSC v.1600 64 bit
(AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

Operators and Expressions

- or operator



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

Operators and Expressions

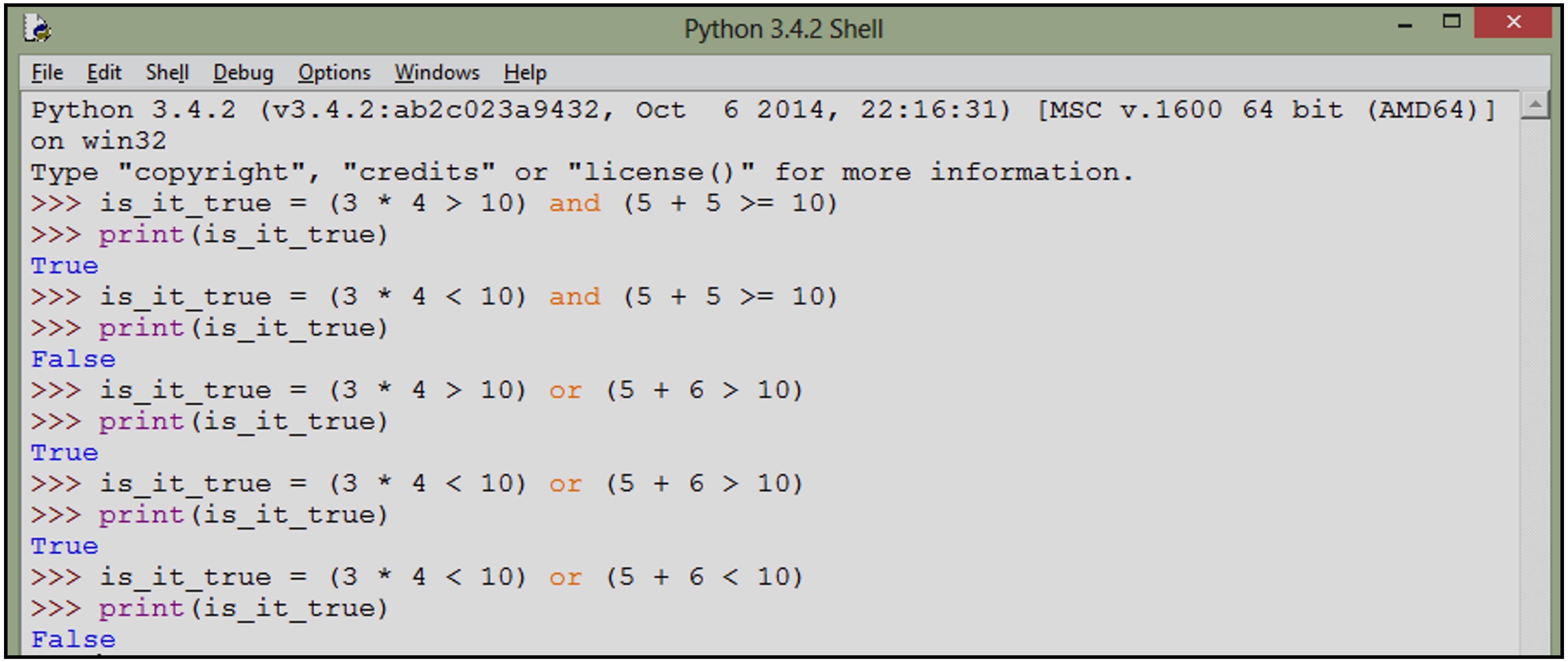
- **Using Number and Strings with Boolean Operators**
- Boolean Operators operate based on logic.
  - and

    If both operands are true outcome must be true otherwise false
  - or

    If one of the operand is true outcome must be true otherwise false.
  - not just negates or inverts its Boolean value(unary operator)

Operators and Expressions

- **Using Number and Strings with Boolean Operators**



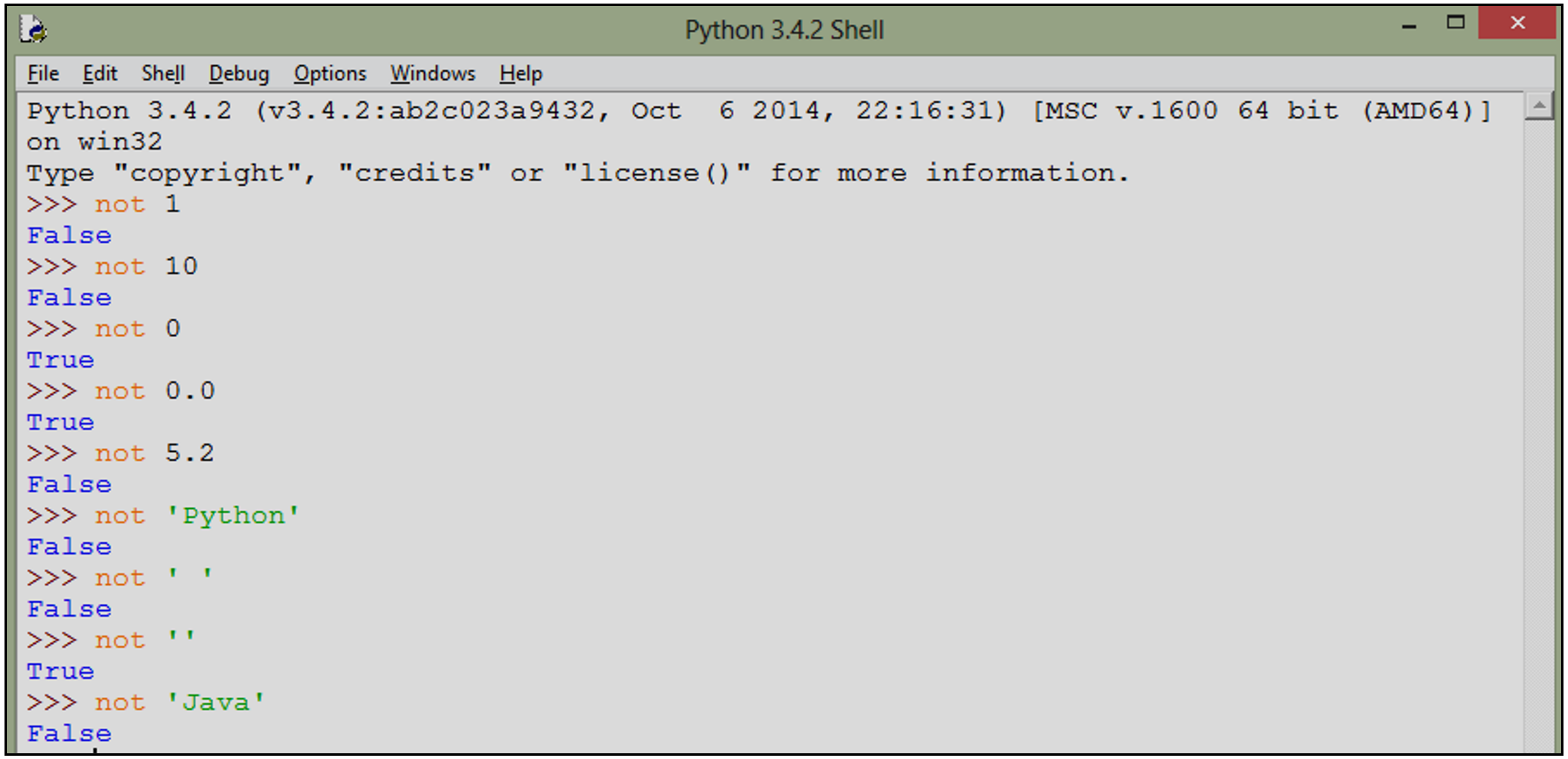
```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> is_it_true = (3 * 4 > 10) and (5 + 5 >= 10)
>>> print(is_it_true)
True
>>> is_it_true = (3 * 4 < 10) and (5 + 5 >= 10)
>>> print(is_it_true)
False
>>> is_it_true = (3 * 4 > 10) or (5 + 6 > 10)
>>> print(is_it_true)
True
>>> is_it_true = (3 * 4 < 10) or (5 + 6 > 10)
>>> print(is_it_true)
True
>>> is_it_true = (3 * 4 < 10) or (5 + 6 < 10)
>>> print(is_it_true)
False
```

Operators and Expressions

- **Using Number and Strings with Boolean Operators**
- Boolean operator not treats all numbers as True.
- If you write not 1,then python substitute 1 as True and evaluate it as not True which returns False.
- If you write not 0 or 0.0,then python substitute it as False and evaluate it as not False which returns True.
- If you apply not operator on strings then Python treats all strings as True.
- If you write not 'Python', then python substitute it as True and evaluate it as not True which returns False.
- If you write not '', i.e empty string then python substitute it as False and evaluate it as not False which returns True.

Operators and Expressions

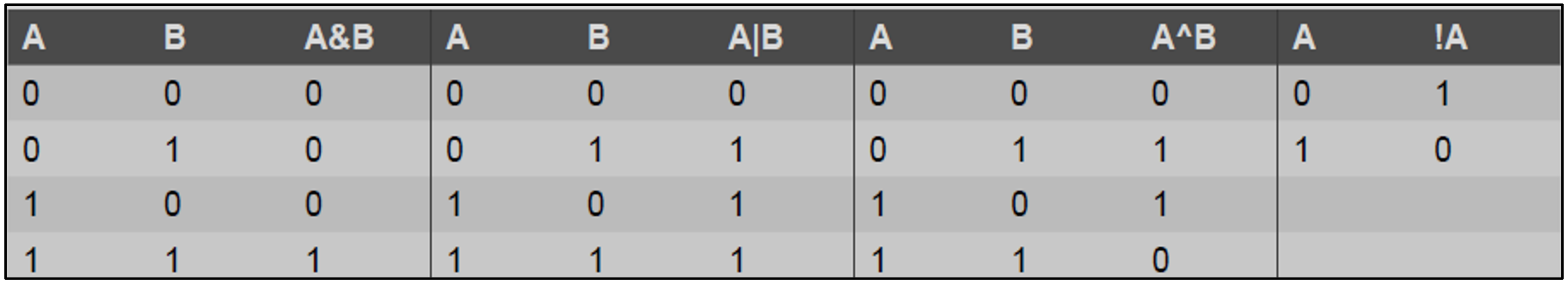
- **Using Number and Strings with Boolean Operators**



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> not 1
False
>>> not 10
False
>>> not 0
True
>>> not 0.0
True
>>> not 5.2
False
>>> not 'Python'
False
>>> not ' '
False
>>> not ''
True
>>> not 'Java'
False
```

Operators and Expressions

- **Bitwise Operators**
- Bitwise operator works on bits and performs bit-by-bit operation.
- These operators include bitwise AND(&), bitwise OR(|), bitwise XOR(^), and shift operators. Bitwise operators expect their operands to be of integers and treat them as a sequence of bits.
- The truth tables of these bitwise operators are given below.



A	B	A&B	A	B	A|B	A	B	A^B	A	!A
0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1	1	0
1	0	0	1	0	1	1	0	1		
1	1	1	1	1	1	1	1	0		

Operators and Expressions

- **Bitwise Operators**
- Python supports two bitwise shift operators. They are shift left (<<) and shift right (>>). These operations are used to shift bits to the left or to the right.
- Right Shift (>>)
- If we have to right shift 4 by 2 bits then
- 4>>2

Convert 4 into binary format i.e. 0100 or 8 bit format as

0000 0100

Right shift by 2 bits

0000 0001

i.e. 4>>2 is now 1.

Operators and Expressions

- **Bitwise Operators**
- Left Shift (<<)
- If we have to left shift 4 by 2 bits then
- 4<<2

Convert 4 into binary format i.e. 0100 or 8 bit format as

        0000 0100

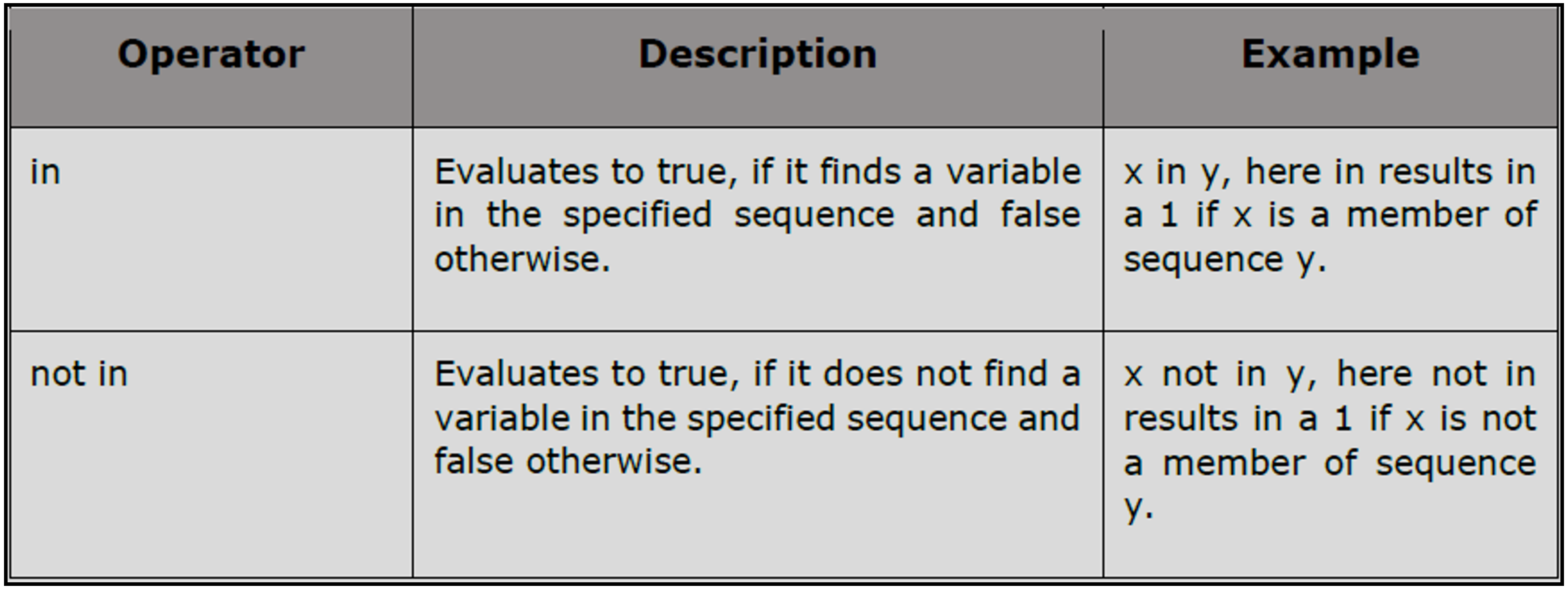
Left shift by 2 bits

        0001 0000

i.e. 4<<2 is now 16.

Operators and Expressions

- **Membership Operators**
- Python's membership operators test for membership in a sequence, such as strings, lists or tuples.
- There are two membership operators as



Operator	Description	Example
in	Evaluates to true, if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true, if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

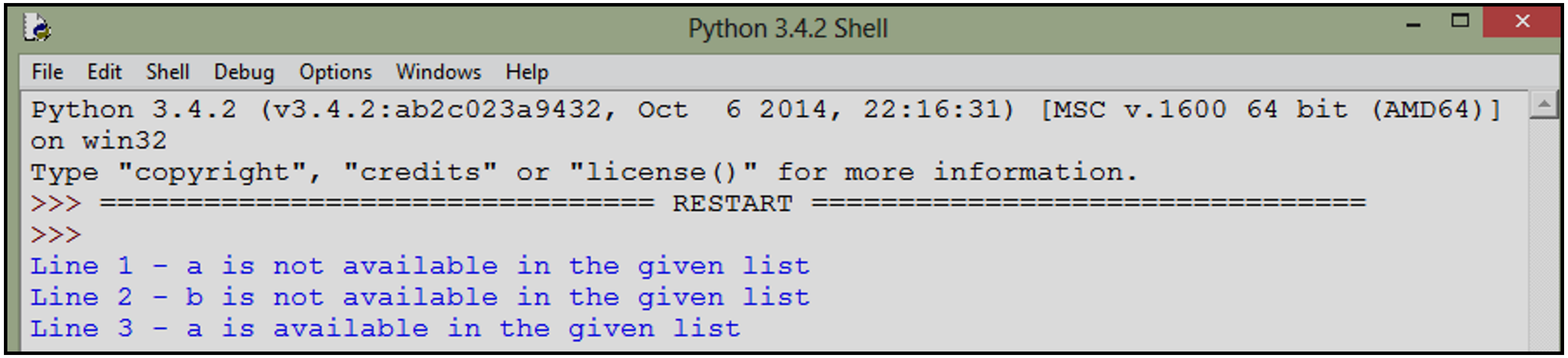
Operators and Expressions

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ]
if ( a in list ):
   print("Line 1 - a is available in the given list")
else:
   print("Line 1 - a is not available in the given list")
if ( b not in list ):
   print("Line 2 - b is not available in the given list")
else:
   print("Line 2 - b is available in the given list")
c=b/a
if ( c in list ):
   print("Line 3 - a is available in the given list")
else:
   print("Line 3 - a is not available in the given list")
```

Membership Operators

Operators and Expressions

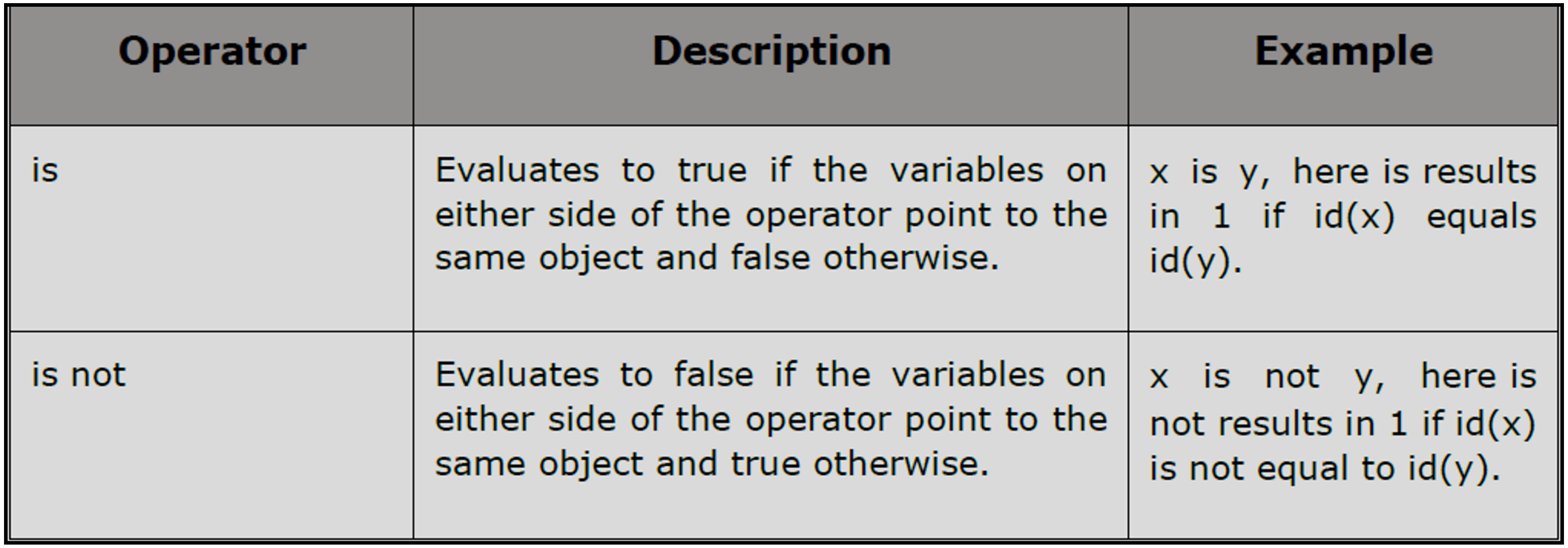
- **Membership Operators**



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list
```

Operators and Expressions

- **Identity Operators**
- Identity operators compare the memory locations of two objects. There are two Identity operators as



Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

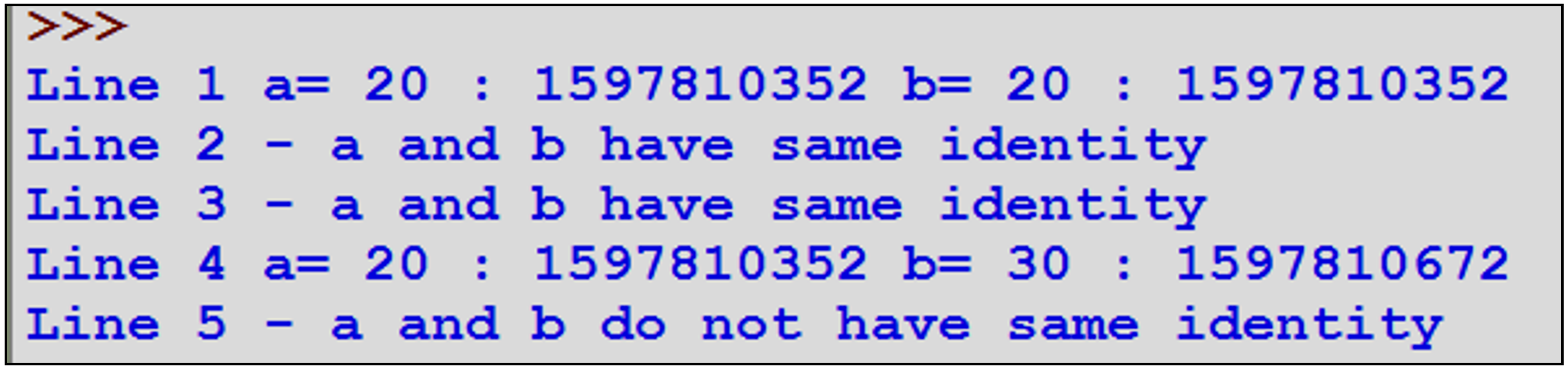
Operators and Expressions

- **Identity Operators**

```
a = 20
b = 20
print ('Line 1','a=',a,':',id(a), 'b=',b,':',id(b))
if ( a is b ):
   print ("Line 2 - a and b have same identity")
else:
   print ("Line 2 - a and b do not have same identity")
if ( id(a) == id(b) ):
   print ("Line 3 - a and b have same identity")
else:
   print ("Line 3 - a and b do not have same identity")
 b = 30
 print ('Line 4','a=',a,':',id(a), 'b=',b,':',id(b))
if ( a is not b ):
   print ("Line 5 - a and b do not have same identity")
else:
   print ("Line 5 - a and b have same identity")
```

Operators and Expressions

- **Identity Operators**



```
>>>
Line 1 a= 20 : 1597810352 b= 20 : 1597810352
Line 2 - a and b have same identity
Line 3 - a and b have same identity
Line 4 a= 20 : 1597810352 b= 30 : 1597810672
Line 5 - a and b do not have same identity
```

Operators and Expressions

- **Precedence of Python Operators**
- The combination of values, variables, operators and function calls is termed as an expression.
- Python interpreter can evaluate a valid expression.
- For example:
- >>> 5 – 7
- -2
- Here 5 - 7 is an expression.
- There can be more than one operator in an expression.
- To evaluate these type of expressions there is a rule of precedence in Python. It guides the order in which operation are carried out.

Operators and Expressions

- **Precedence of Python Operators**
- For example, multiplication has higher precedence than subtraction.
- # Multiplication has higher precedence
- # than subtraction
- >>10 - 4 * 2

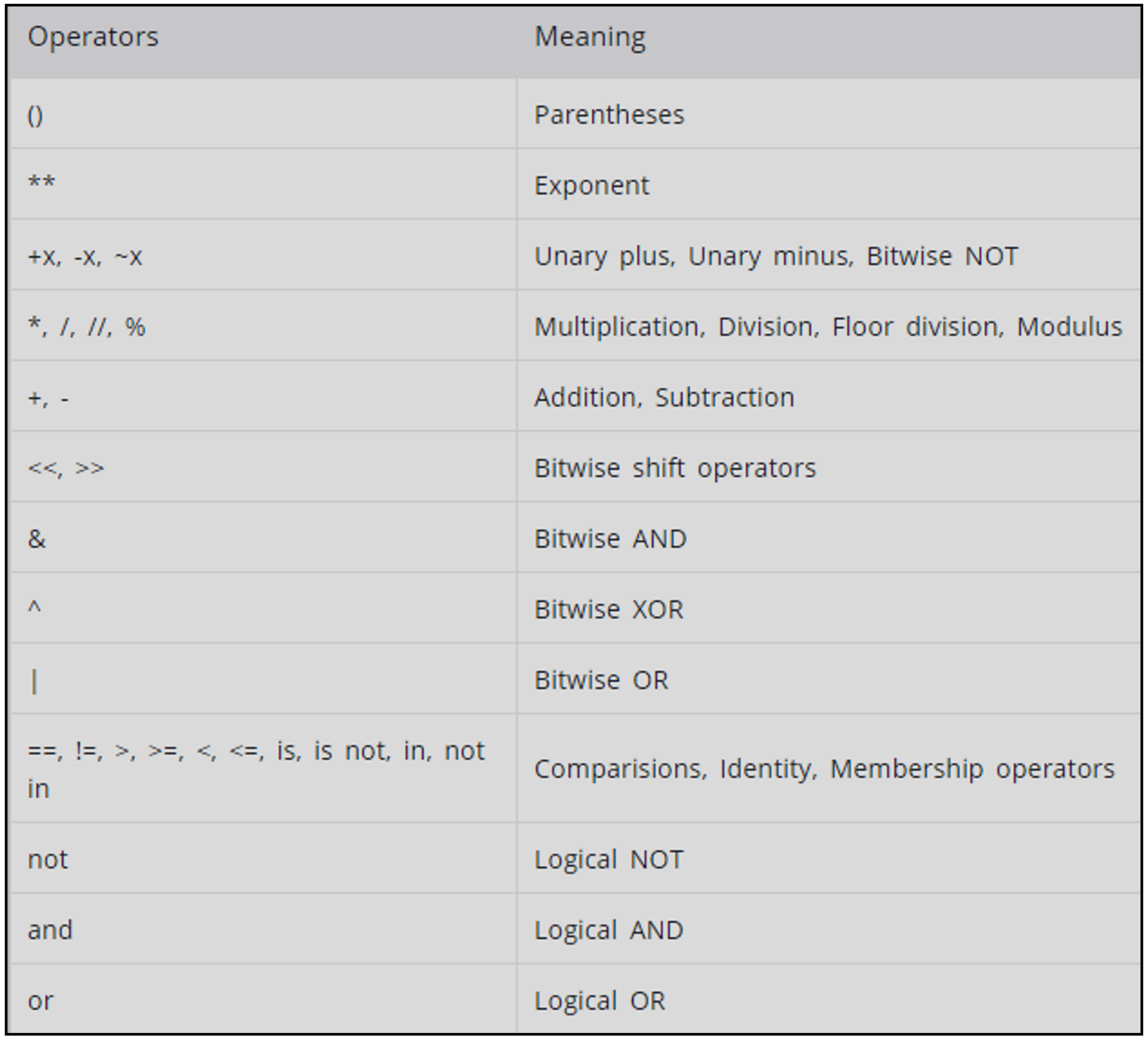
  2
- But you can change this order using parentheses ( ) as it has higher precedence.
- >>> (10-4)*2

  12

Operators and Expressions

- **Precedence of Python Operators**
- Operator precedence affects how an expression is evaluated. The operator precedence in Python are listed in the form of a table.
- **Associativity of Python Operators**
- You can see in the table that more than one operator exists in the same group. These operators have the same precedence.
- When two operators have the same precedence, associativity helps to determine which the order of operations.
- Associativity is the order in which an expression is evaluated that has multiple operator of the same precedence.
- Almost all the operators have left-to-right associativity.

Operators and Expressions



Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisions, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Highest precedence

Lowest precedence

Operators and Expressions

- For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, left one is evaluates first.
- **# Left-right associativity**

-
  ```
  >>> print(5 * 2 // 3)
  3
  ```

-
  ```
  >>> print(5 * (2 // 3))
  0
  ```
- Exponent operator ** has right-to-left associativity in Python.
- **# Right-left associativity** of ** exponent operator

-
  ```
  >>> print(2 ** 3 ** 2)
  512
  ```

-
  ```
  >>> print((2 ** 3) ** 2)
  64
  ```

Operators and Expressions

- **Non associative operators**
- Some operators like assignment operators and comparison operators do not have associativity in Python.
- For example, x < y < z neither means (x < y) < z nor x < (y < z). x < y < z is equivalent to x < y and y < z, and is evaluates from left-to-right.
- Furthermore, while chaining of assignments like x = y = z is perfectly valid, x = y += z will result into error.

THANKS…..