Essential Data Skills for Business Analytics

Lecture 2: Variables, expressions, statements

Decision, Operations & Information Technologies
Robert H. Smith School of Business
Spring, 2020

Constants

- Fixed values such as numbers, letters, and strings are called "constants" because their value does not change.
- Numeric constants are as you expect
- String constants use single quotes (') or double quotes (")

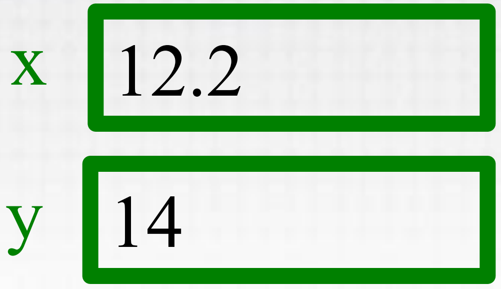
```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Variables

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable "name".
- Programmers need to choose the names of the variables.
- You can change the contents of a variable in a later statement.

```
x = 12.2
y = 14
```

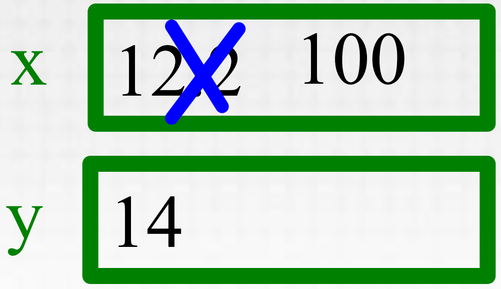


Variables

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable "name".
- Programmers need to choose the names of the variables.
- You can change the contents of a variable in a later statement.

```
x = 12.2
y = 14
x = 100
```



Variable name rules

1. Must start with a letter or underscore _
2. Only consist of letters, numbers, and underscores
3. Case sensitive
   - Different: smith Smith SMITH SmiTH
4. You can not use reserved words as variable names

   smith $smiths smith23 _smith _23_

   23smith smith.23 a+b smiTH -smith

Variable name rules

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

1. Must start with a letter or underscore _
2. Only consist of letters, numbers, and underscores
3. Case sensitive
   - ❑ Different: smith Smith SMITH SmiTH
4. You can not use reserved words as variable names

   smith $smiths smith23 _smith _23_

   23smith smith.23 a+b smiTH -smith

Green: Good var names **Red: Bad var names**

Reserved words

and del for is raise assert elif from
lambda return break else global not
try class except if or while continue
exec import pass yield def finally in
print as with

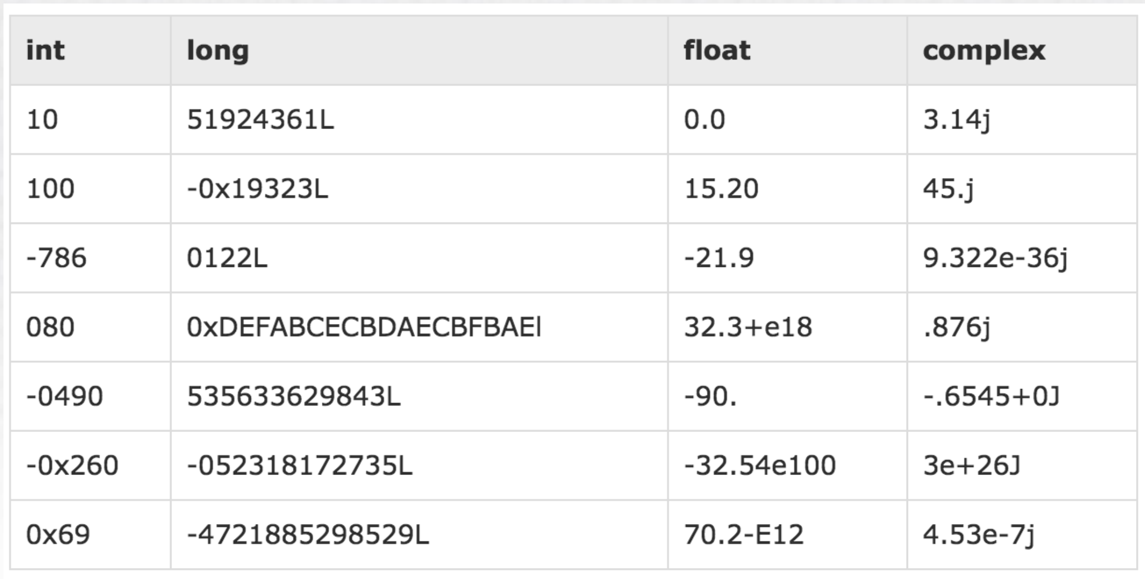
```
>>> 32smith = ‘hello world’
SyntaxError: invalid syntax

>>> class = 27
SyntaxError: invalid syntax
```

Variable type

- The programmer (and the interpreter) can identify the type of a variable.
- You do not need to explicitly define or declare the type of a variable.
  - ❑int x = 3   (for most other languages)
  - ❑x = 3 (for Python)
- Python has five standard data types
  - ❑Numbers
  - ❑String
  - ❑List
  - ❑Tuple
  - ❑Dictionary

Numbers

- Python supports four different numerical types:
  - ❑int (signed integers)
  - ❑long (long integers, they can also be represented in octal and hexadecimal)
  - ❑float (floating point real values)
  - ❑complex (complex numbers)



int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Strings

- Strings in Python are identified as a contiguous set of characters represented in the single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string.

```
#!/usr/bin/python

str = 'Hello World!'

print(str)          # Prints complete string
print(str[0])       # Prints first character of the string
print(str[2:5])     # Prints characters starting from 3rd to 5th
print(str[2:])      # Prints string starting from 3rd character
```

Output:
```
Hello World!
H
llo
llo World!
```

Lists

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets([]).
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list.

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list)          # Prints complete list
print(list[0])       # Prints first element of the list
print(list[1:3])          # Prints elements starting from 2nd till 3rd
print(list[2:])           # Prints elements starting from 3rd element
```

Output:

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
```

Tuples

- A tuple is another sequence data type that is similar to the list.
- A tuple contains items separated by commas and enclosed within parentheses ().
- Difference between lists and tuples:
    - ❑Elements and size in lists can be changed, while tuples can not

```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print(tuple)          # Prints complete list
print(tuple[0])       # Prints first element of the list
print(tuple[1:3])     # Prints elements starting from 2nd till 3rd
print(tuple[2:])      # Prints elements starting from 3rd element
```

Output:

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
```

Dictionary

- Consists of a number of key-value pairs.
- It is enclosed by curly braces ({ })

- We will see more details about dictionary later.

Get variable type

- If you are not sure what type a variable has, the interpreter can tell you by using **type**.
  - ❑>>> type('Hello, world!')

    <type 'str'>
  - ❑>>> type(17)

    <type 'int'>
  - ❑>>> type(3.2)

    <type 'float'>
  - ❑>>> type('4.7')

    <type 'str'>

Type matters

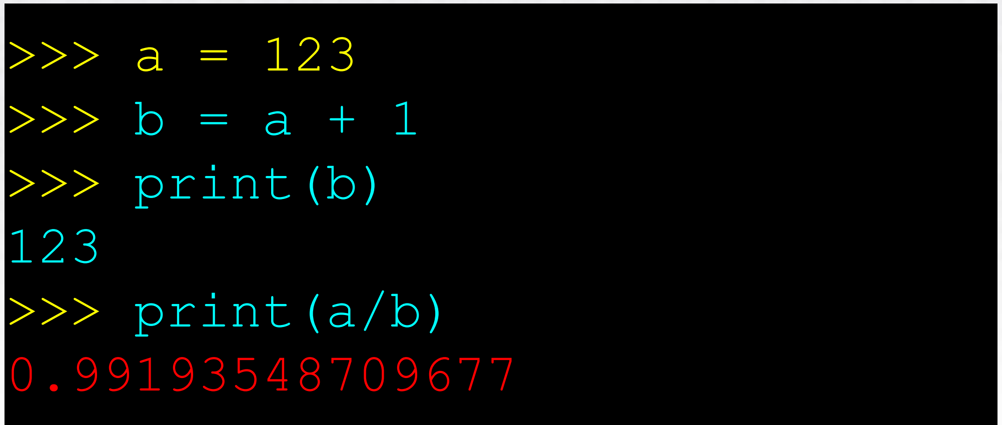
- Python knows what “type” everything is.

- Some operations are prohibited.
    - ❑You can not add a string to an integer.

```
>>> a = '123'
>>> b = a + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and
'int' objects
```

Type matters

- Operations on the same type operands will lead to results with the same type
  - ❑int + int => int
  - ❑int – int => int
  - ❑int * int => int
  - ❑int / int => int (Python 2) float (Python 3)



```
>>> a = 123
>>> b = a + 1
>>> print(b)
123
>>> print(a/b)
0.99193548709677
```

Type conversions

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- When you put an integer and floating point in an expression, the integer is **implicitly** converted to a float.
- You can control this with the built-in functions int() and float()

```
>>> print(float(99) / 100)
0.99
>>> i = 42
>>> type(i)
<type 'int'>
>>> f = float(i)
>>> print (f)
42.0
>>> type(f)
<type 'float'>
>>> print (1 + 2 * float(3) / 4 - 5)
-2.5
```

String conversions

- You can use int() and float() to convert between strings and integers/floating points.
- You will get an error if the string does not contain numeric characters.

```
>>> sval = '123'
>>> type(sval)
<type 'str'>
>>> print (sval + 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int'
>>> ival = int(sval)
>>> type(ival)
<type 'int'>
>>> print (ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

User input

- We can ask Python to pause and read data from the keyboard using input() function.
- The input() function returns a string.

```
>>> name = input('What's your name? ')
What's your name? KZ # You type KZ using keyboard
>>> print ('Welcome', name)
Welcome KZ
```

Converting user input

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- If we want to read a number from the keyboard, we must convert it from a string to a number using a type conversion function [int() or float()].

```
>>> inp = input('Europe floor? ')
Europe floor? 2 # 2 is the input from keyboard
>>> usf = int(inp) + 1
>>> print ('US floor', usf)
US floor 3
```

The assignment statement

- An assignment statement creates new variables and gives them values.
- An assignment statement consists of an expression on the right-hand side and a variable to store the result.
  - ❑>>> message = “hello, world”
  - ❑>>> x = 17
  - ❑>>> pi = 3.14159

Expressions

- An expression is a combination of values, variables, and operators. Not every expression contains all of these elements.
- If you type an expression on the command line (after >>>), the interpreter evaluates it and displays the result.
  - ❑>>> 1+2*7

          15
  - ❑>>>‘Hello’

          Hello

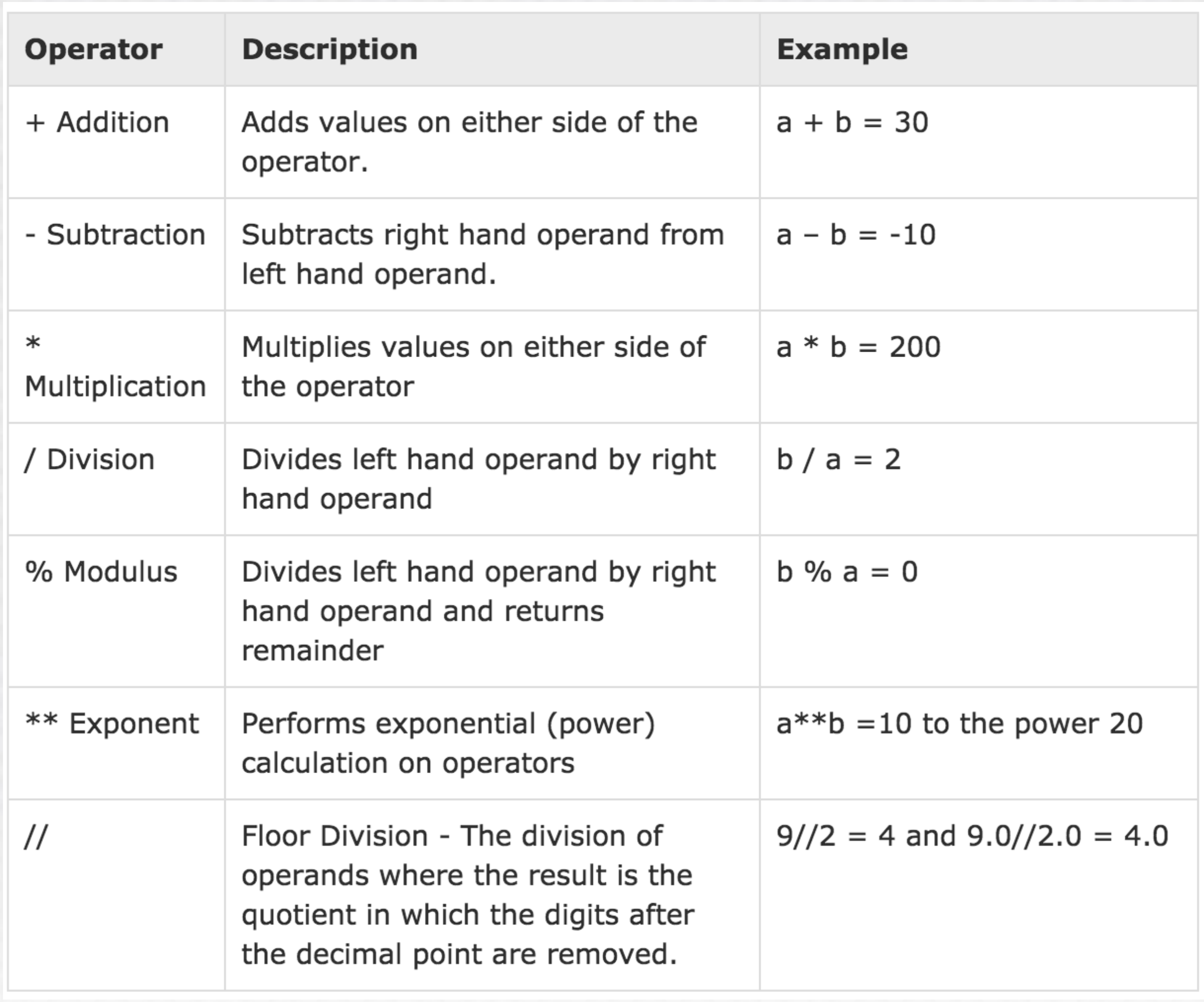
Operators

- Operators are special symbols that represent computations like addition and multiplication.
- The values the operator uses are called operands.
- When a variable name appears in the place of an operand, it is replaced with its value before the operation is performed.
- Types of operator
  - ❑ Arithmetic operators
  - ❑ Comparison (Relational) operators
  - ❑ Assignment operators
  - ❑ Logical operators
  - ❑ Bitwise operators
  - ❑ Membership operators
  - ❑ Identity operators

Arithmetic operators

UNIVERSITY OF MARYLAND
ROBERT H. SMITH
SCHOOL OF BUSINESS

Assume variable a holds 10 and variable b holds 20.



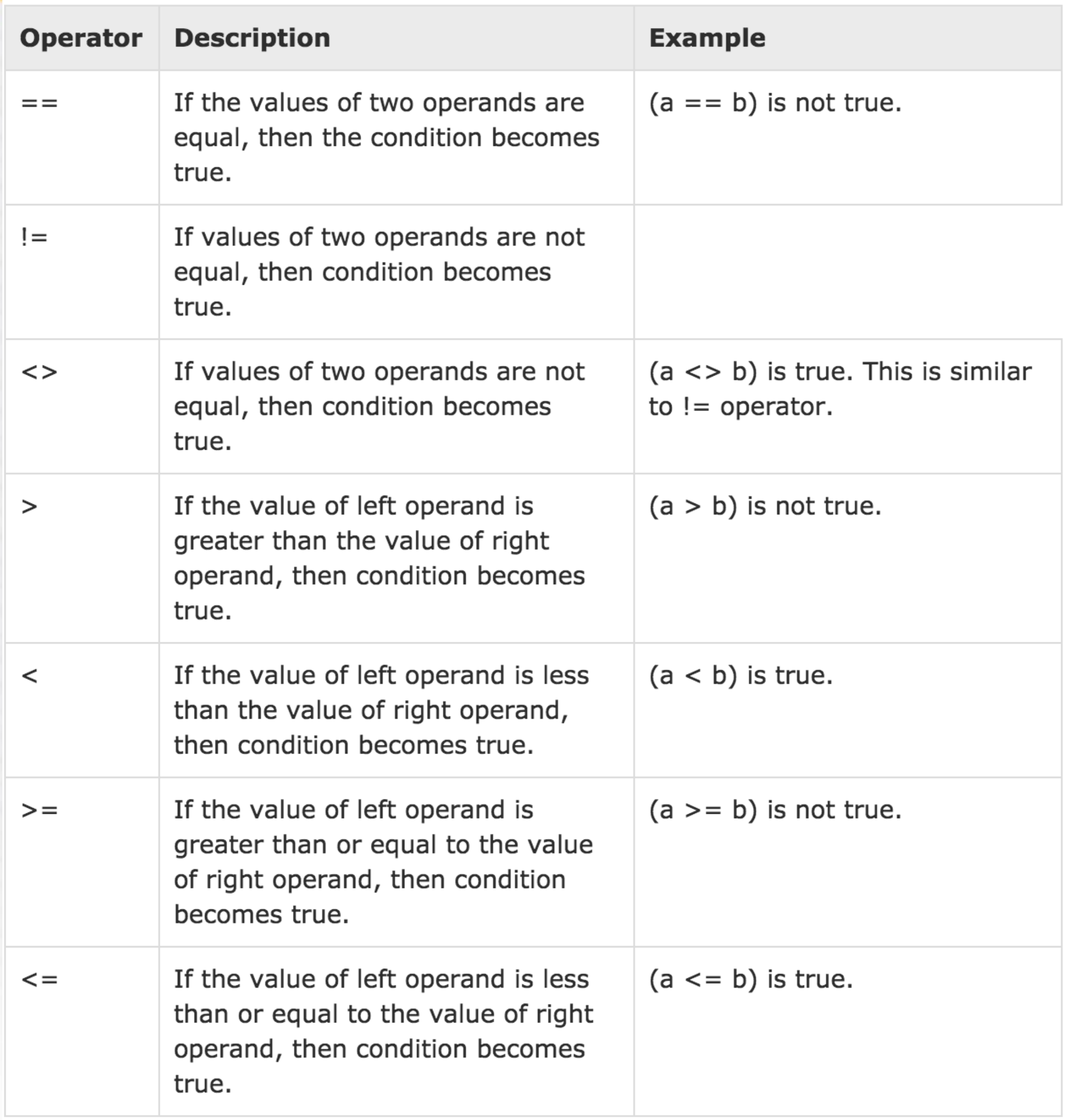
Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 = 4 and 9.0//2.0 = 4.0

Comparison operators

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Compare values on either sides of them and decide the relation among them.

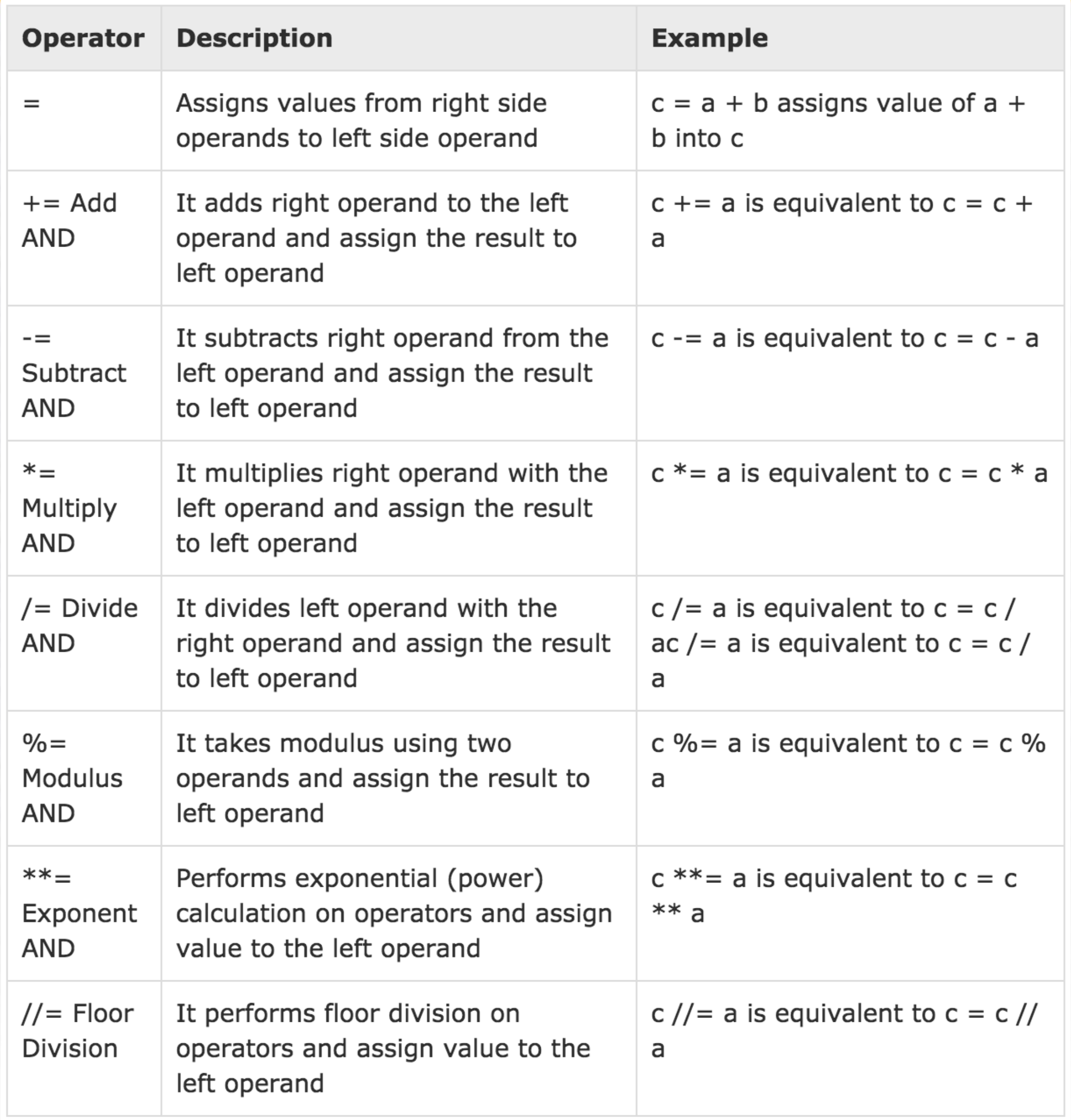
Assume variable a holds 10 and variable b holds 20.



Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Assignment operators

Assume variable a holds 10 and variable b holds 20.



Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Bitwise operators

Assume variable a holds 60 and variable b holds 13.

Binary representation:

a = 0011 1100

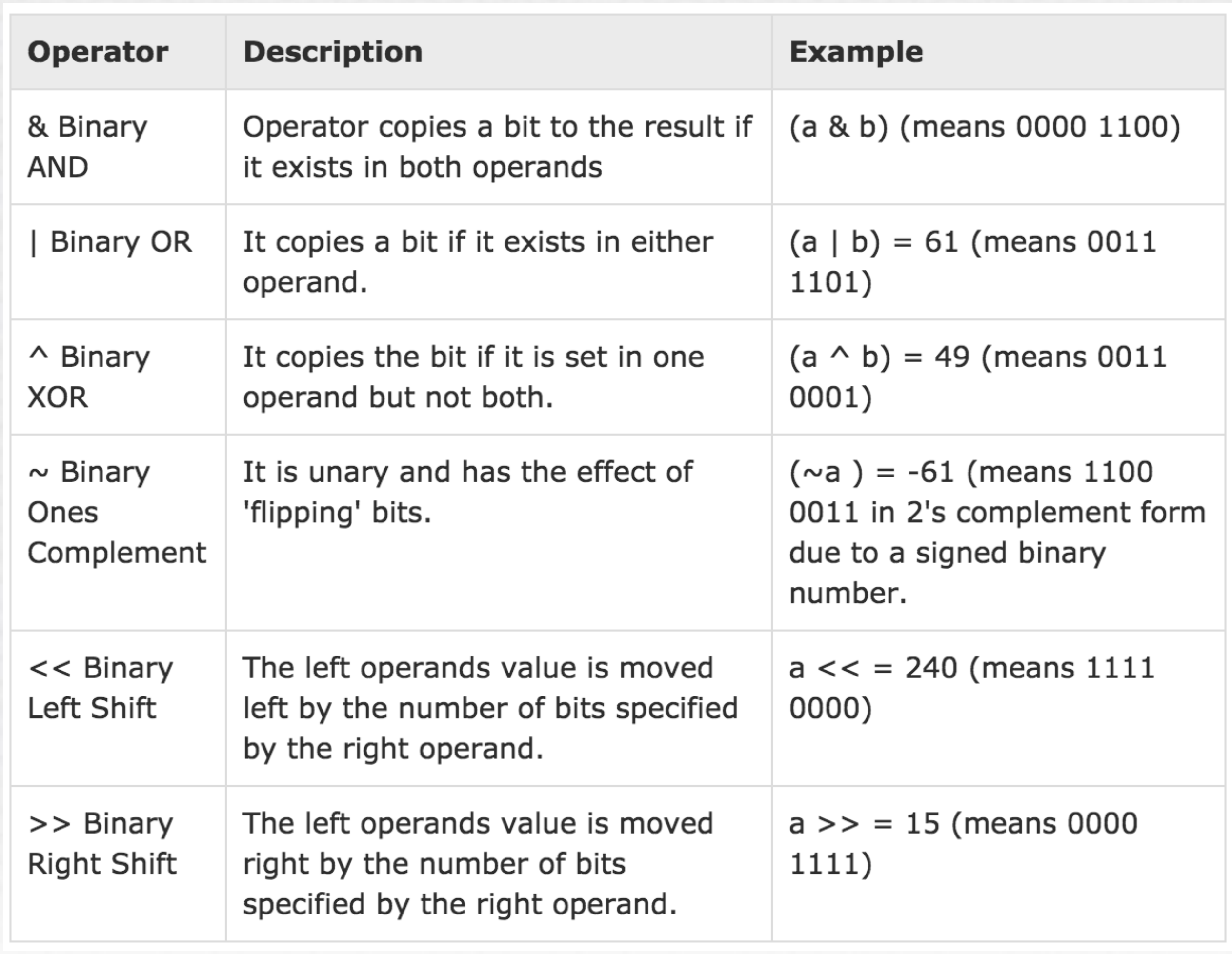
b = 0000 1101

a & b = 0000 1100

a | b = 0011 1101

a ^ b = 0011 0001

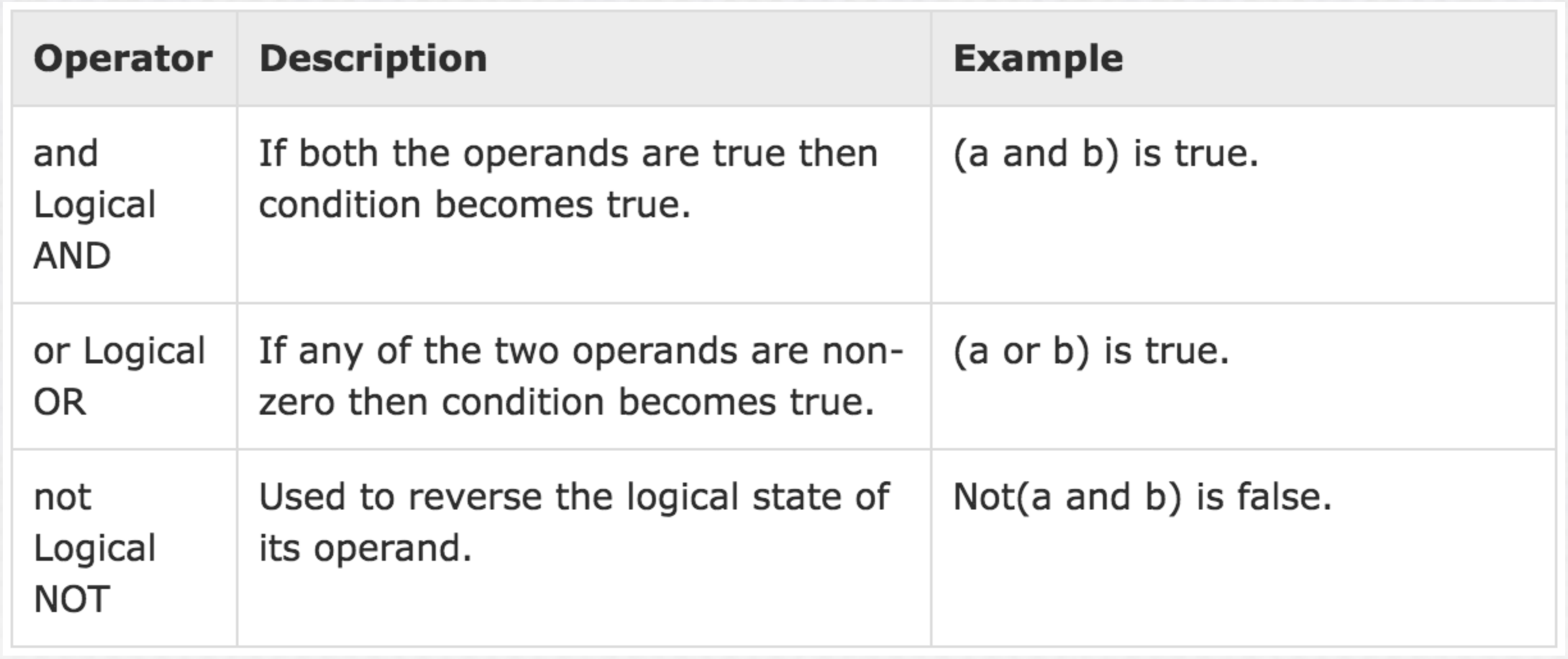
~a = 1100 0011



Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> = 15 (means 0000 1111)

Logical operators

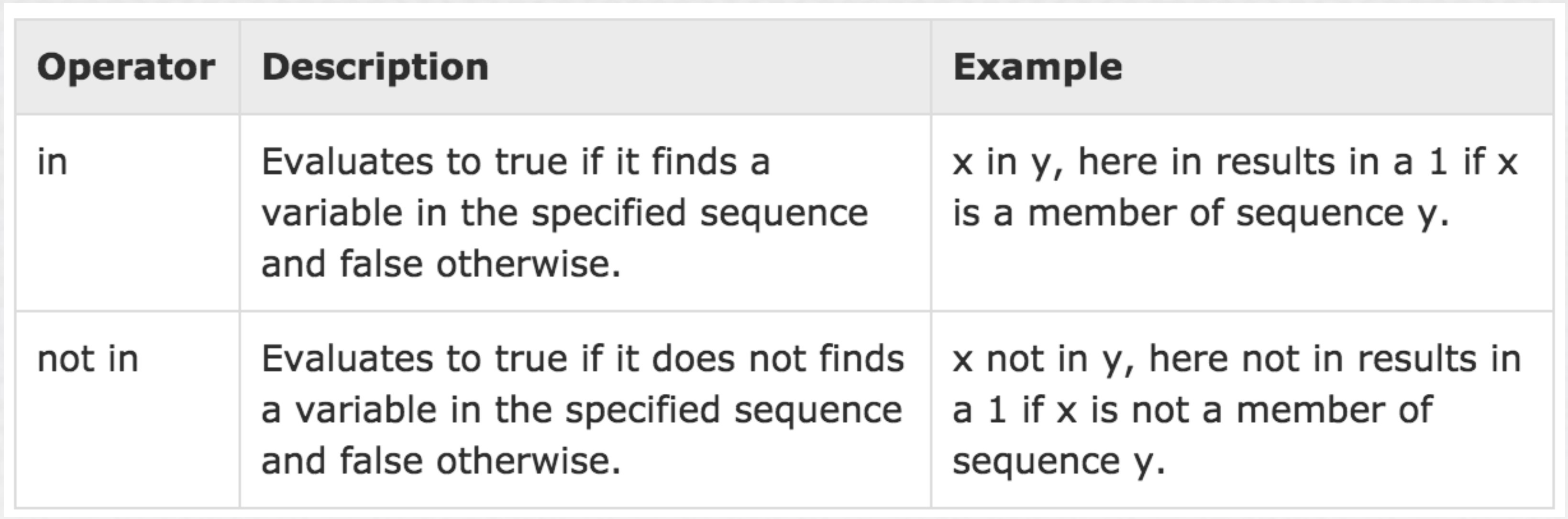
Assume variable a holds True and variable b holds False.



Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Membership operators

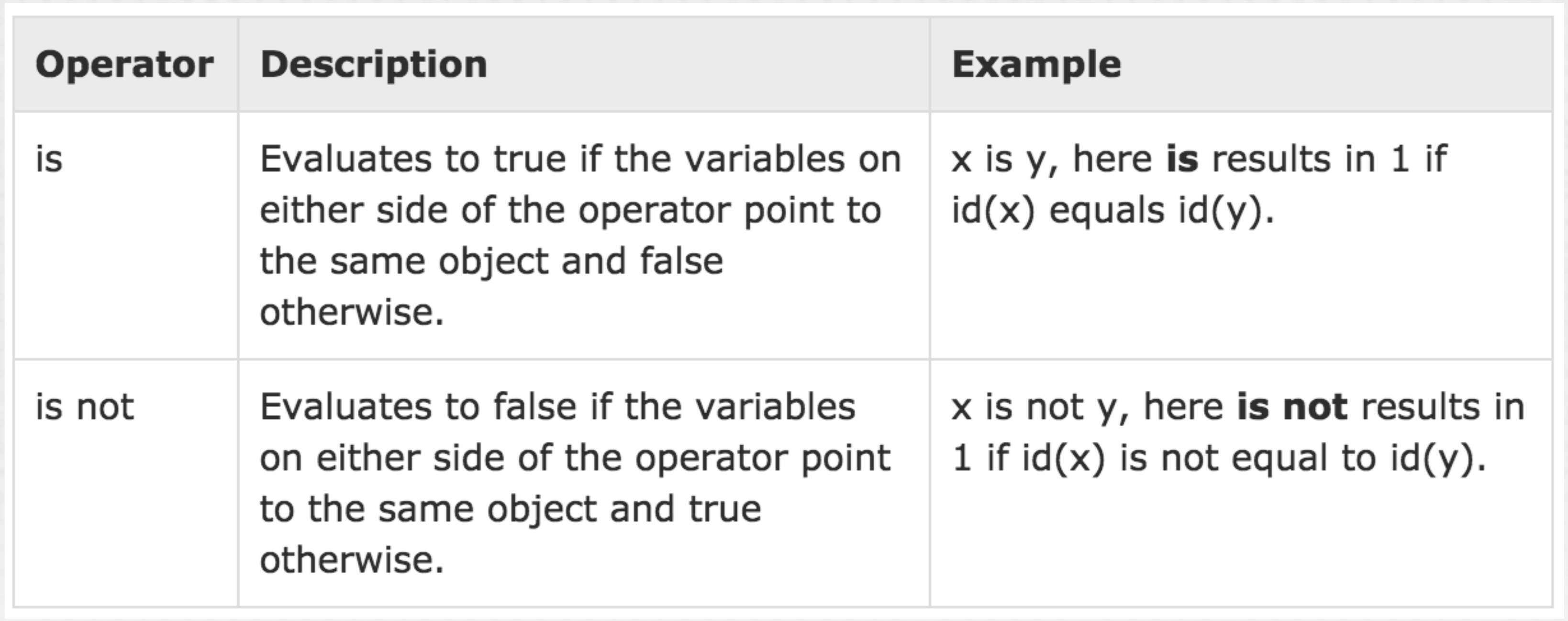
Test for memberships in a sequence, such as strings, lists, or tuples.



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 finds 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.

Identity operators

Compare the memory location of two objects.

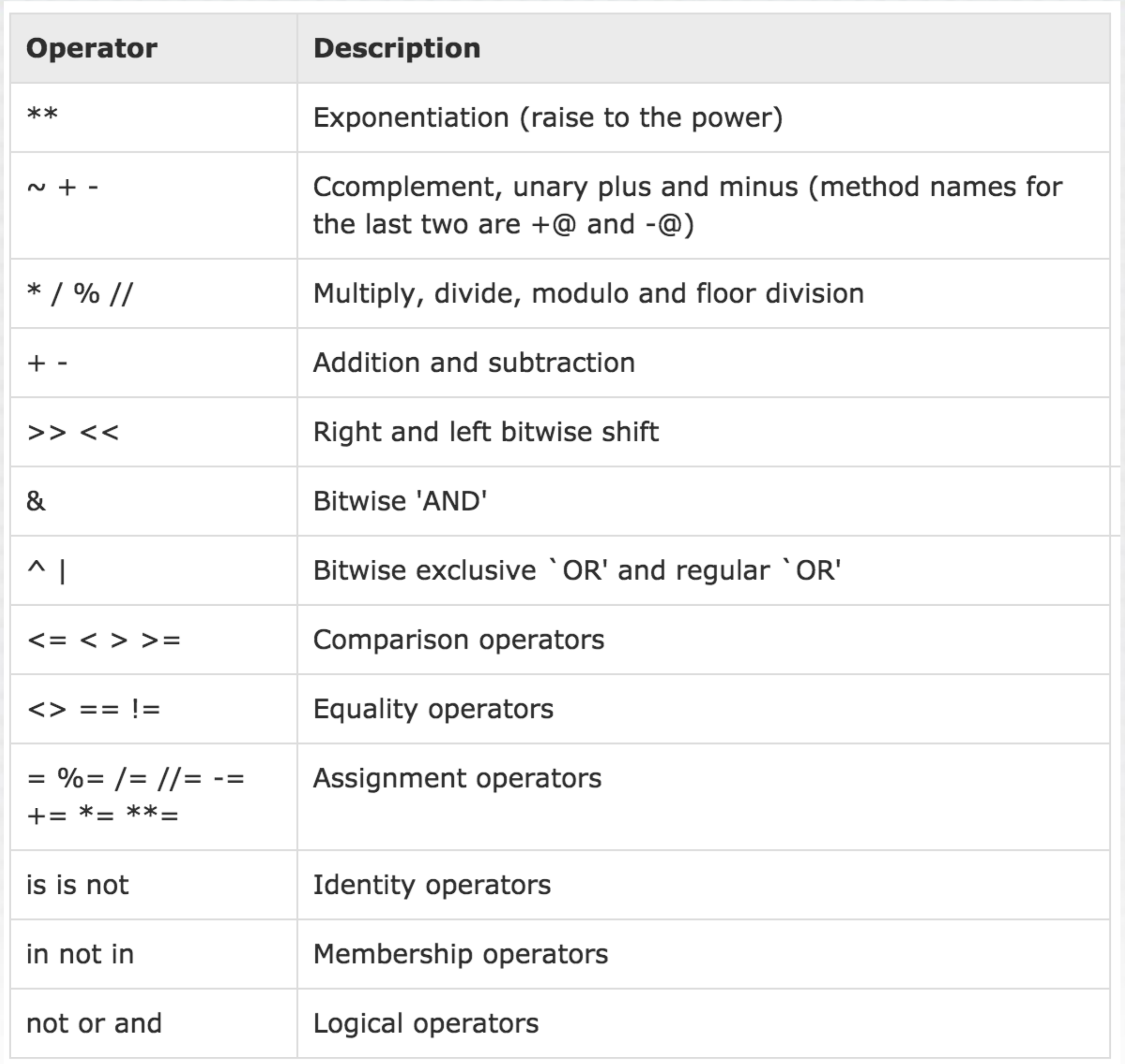


Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and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

Operator precedence

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.
  - ❑ Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want.
  - ❑ Exponentiation has the next highest precedence.
  - ❑ Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence
  - ❑ Operators with the same precedence are evaluated from left to right.

Operator precedence



Operator	Description
**	Exponentiation (raise to the power)
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

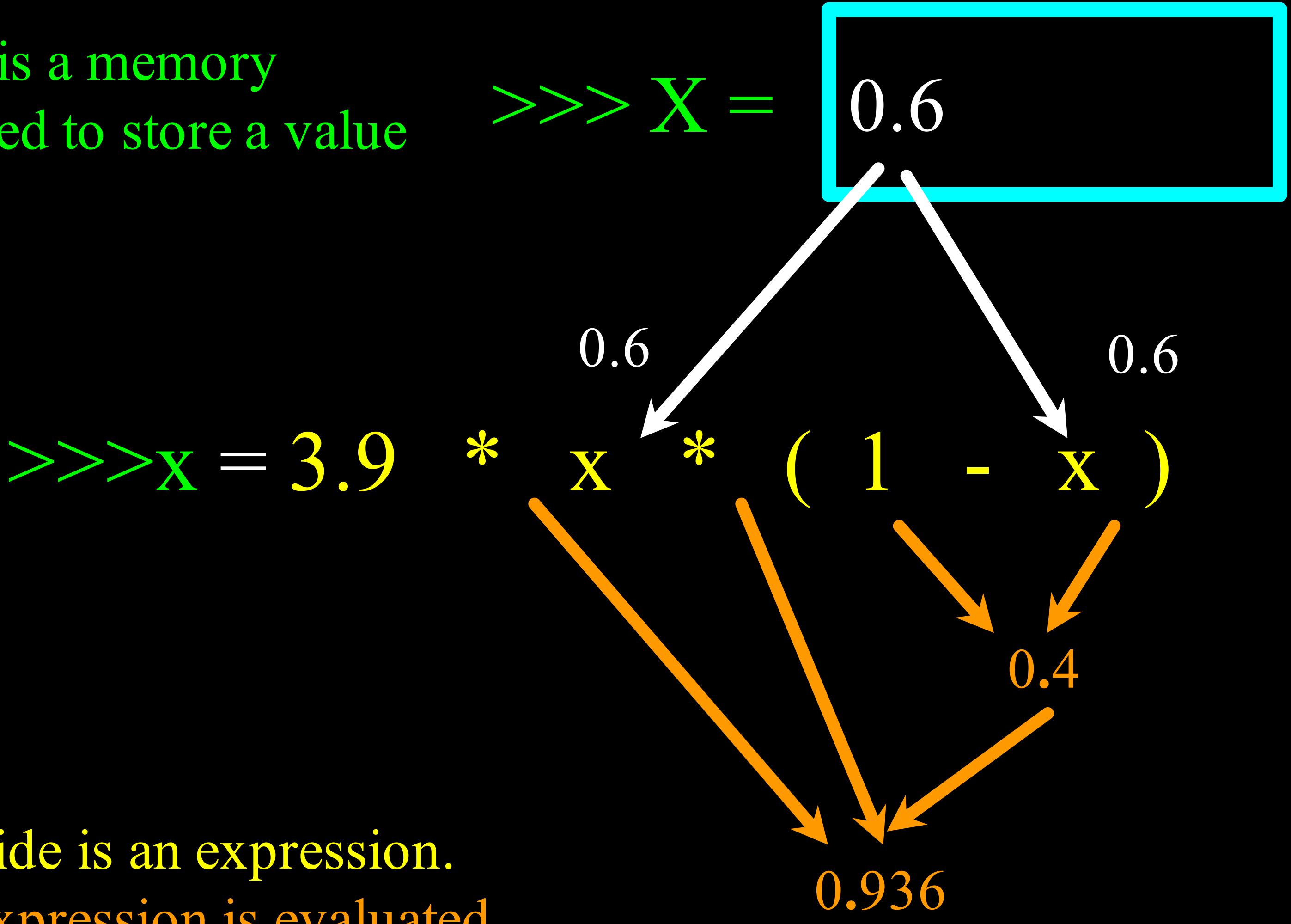
Assignment statement

- An assignment statement consists of an expression on the right-hand side and a variable to store the result.

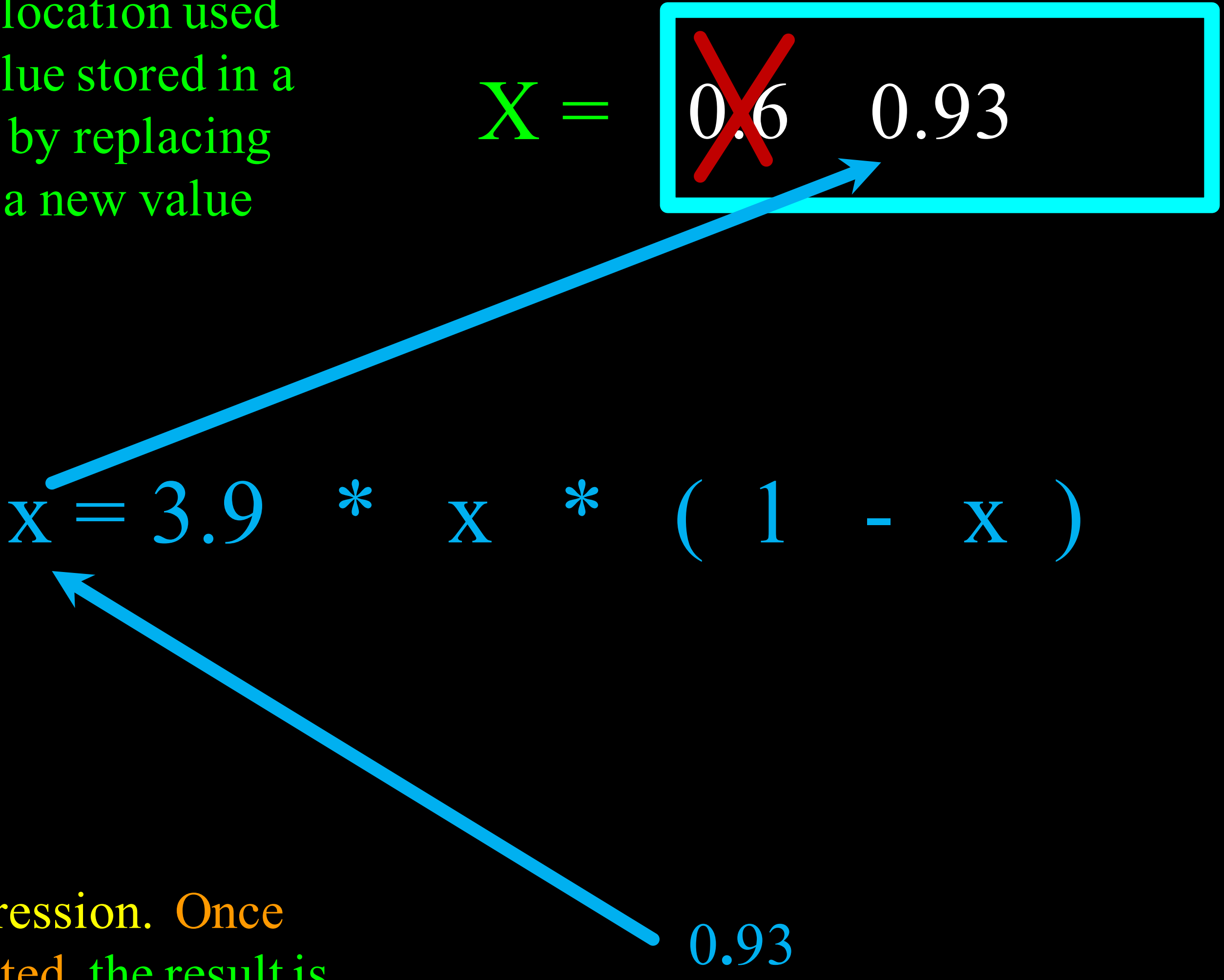
X = 3.9 * X * (1 - X)

A variable is a memory location used to store a value (0.6)

The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) x.



A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.93).



The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e., x).

Integer division is weird (Python 2.*)

- Integer division truncates.
  - >>> print 10 / 2
    5
  - >>> print 9 / 2
    4
- Floating point division produces floating point numbers.
  - >>> 10.0 / 2.0
    5.0
  - >>> 9.0 / 2.0
    4.5

This changes in Python 3.0

Mixing integer and floating

- When you perform an operation where one operand is an integer and the other one is a floating point, the result is a floating point.
  - ❑
    ```
    >>> print 10.0 / 2
        5.0
    ```
  - ❑
    ```
    >>> print 9 / 2.0
        4.5
    ```

Statements

- Simple statements: executed sequentially and do not affect the flow of control.
  - ❑Print statement
  - ❑Assignment statement
  - ❑And many others…

```
| expression_stmt
| assert_stmt
| assignment_stmt
| augmented_assignment_stmt
| pass_stmt
| del_stmt
| return_stmt
| yield_stmt
| raise_stmt
| break_stmt
| continue_stmt
| import_stmt
| global_stmt
| nonlocal_stmt
```

- Compound statements: may affect the sequence of execution.