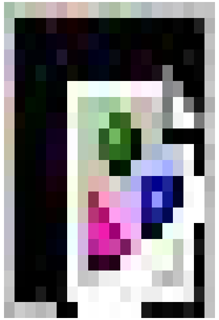
Python

Introduction

Python Facts, Orientation



- An interpreted language developed in the late 1980s by Guido Van Rossum
- Has developed into a widely used programming language for
  - Web programming (Django)
  - Scripting
  - Data Science/Scientific computing
- To use, you need to install a python interpreter
  - Recommended: Miniconda (see course website)
- The textbook author has put all the code examples on Github (see course website for link)
- We will be using Python 3

Getting Started

- Install the Miniconda package
- Install the IPython (Jupyter) Notebook package
- `[~]$ conda install ipython-notebook`
- Type “ipython” at the command line
- Type `import this`

HOw to Run Python Code

- The Python interpreter
  - Type `python` at the command prompt
  - A new prompt with `>>>` will appear
  - Type commands
- The IPython interpreter: a Python interpreter with more functionality
  - Type `ipython` at command prompt
  - A new prompt with `ln[1]:` will appear
  - Type commands
- Python scripts
  - Type `python my_script.py` to run at command line
- Jupyter Notebooks
  - Type `jupyter notebook` at command line to open your local server

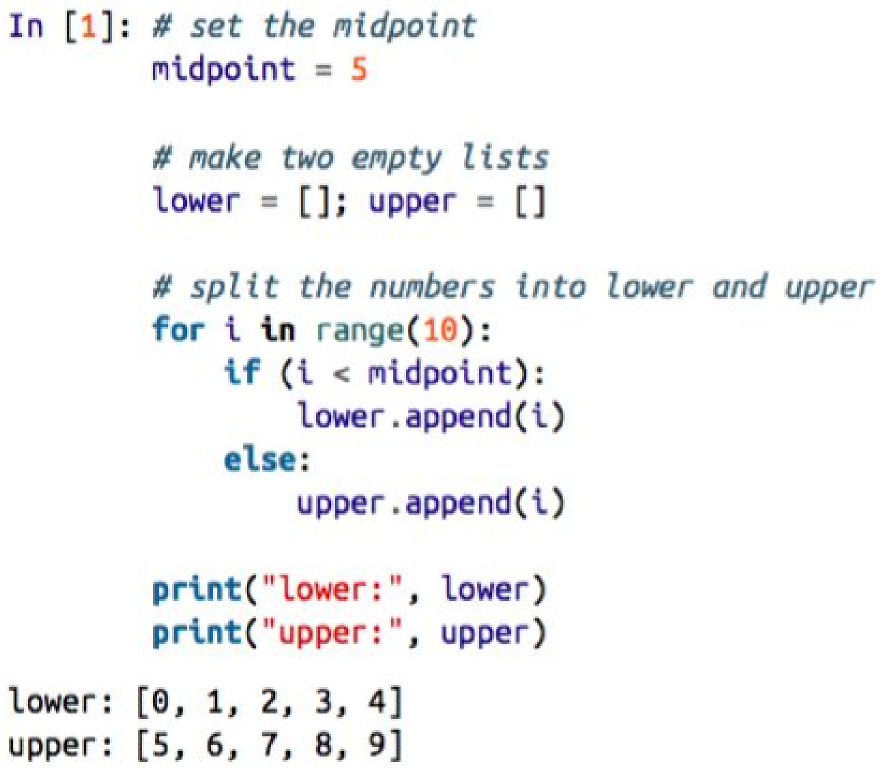
A quick tour of Python syntax

Use : and indentation to indicate code blocks

# starts comments

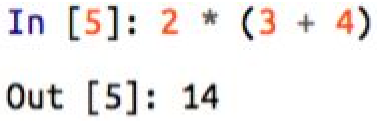
End statements with new line, or ;

White space within lines is ignored

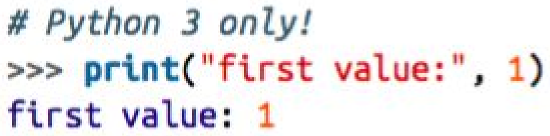


```
In [1]: # set the midpoint
        midpoint = 5

        # make two empty lists
        lower = []; upper = []

        # split the numbers into lower and upper
        for i in range(10):
            if (i < midpoint):
                lower.append(i)
            else:
                upper.append(i)

        print("lower:", lower)
        print("upper:", upper)

lower: [0, 1, 2, 3, 4]
upper: [5, 6, 7, 8, 9]
```

- Parentheses are used for grouping or calling functions
- Python 3 printing
- Python style guide: https://www.python.org/dev/peps/pep-0008/



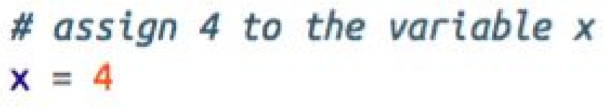
```
In [5]: 2 * (3 + 4)

Out [5]: 14
```



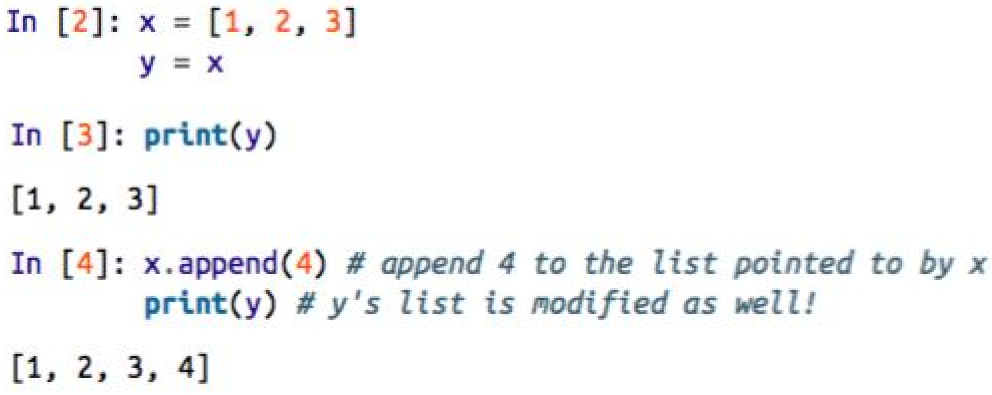
```
# Python 3 only!
>>> print("first value:", 1)
first value: 1
```

Basic Python semantics

- Python variables: just assign a value to a variable name
    - Python variables are pointers, not containers
    - Python, like JavaScript is dynamically typed
- If two variables point to a mutable object, changes using one variable affect the other

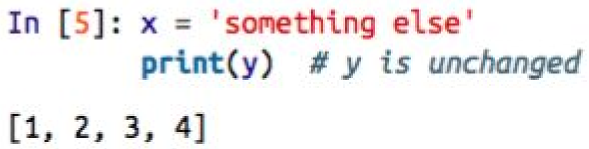


```
# assign 4 to the variable x
x = 4
```

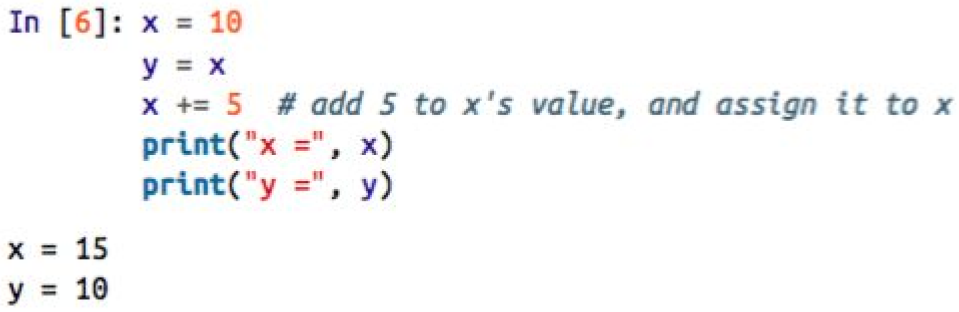


```
In [2]: x = [1, 2, 3]
        y = x

In [3]: print(y)

[1, 2, 3]

In [4]: x.append(4) # append 4 to the list pointed to by x
        print(y) # y's list is modified as well!

[1, 2, 3, 4]
```

- Changing assignments don’t affect underlying objects
- Simple types are immutable, “changing” them simply replaces an earlier value with a new one



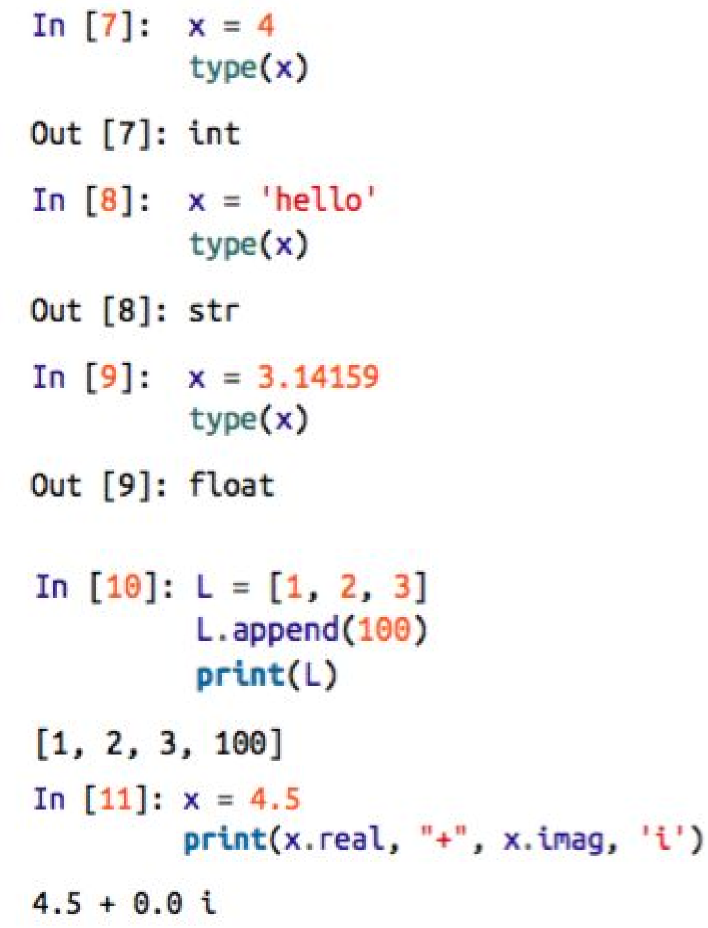
```
In [5]: x = 'something else'
        print(y)  # y is unchanged

[1, 2, 3, 4]
```



```
In [6]: x = 10
        y = x
        x += 5  # add 5 to x's value, and assign it to x
        print("x =", x)
        print("y =", y)

x = 15
y = 10
```

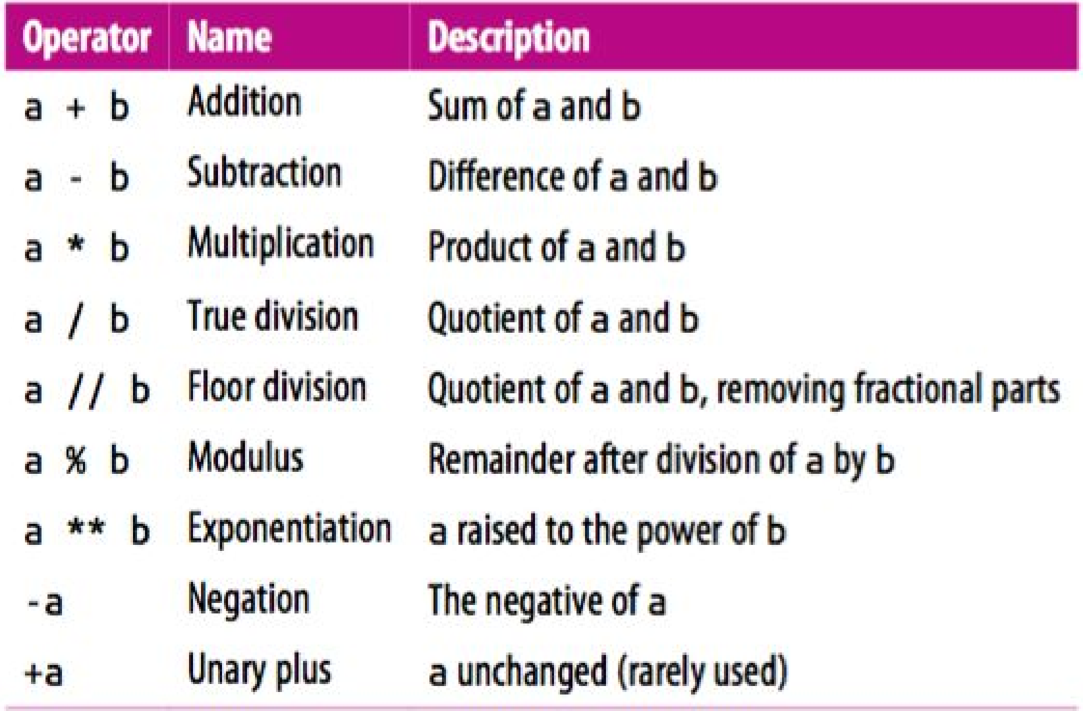
Everything is an object

- Python has types, but the types are linked to the objects themselves, not the variables
- Objects are entities that contain “metadata”
  - These include *attributes* and *methods*
  - Even primitive types have attributes
  - Even the attributes themselves are objects



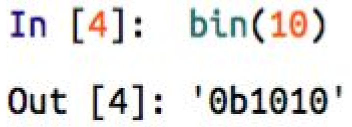
```
In [7]:  x = 4
         type(x)

Out [7]: int

In [8]:  x = 'hello'
         type(x)

Out [8]: str

In [9]:  x = 3.14159
         type(x)

Out [9]: float


In [10]: L = [1, 2, 3]
         L.append(100)
         print(L)

[1, 2, 3, 100]
In [11]: x = 4.5
         print(x.real, "+", x.imag, 'i')

4.5 + 0.0 i
```

Python math Operators

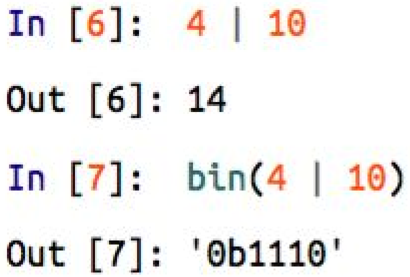


Operator	Name	Description
a + b	Addition	Sum of a and b
a - b	Subtraction	Difference of a and b
a * b	Multiplication	Product of a and b
a / b	True division	Quotient of a and b
a // b	Floor division	Quotient of a and b, removing fractional parts
a % b	Modulus	Remainder after division of a by b
a ** b	Exponentiation	a raised to the power of b
-a	Negation	The negative of a
+a	Unary plus	a unchanged (rarely used)

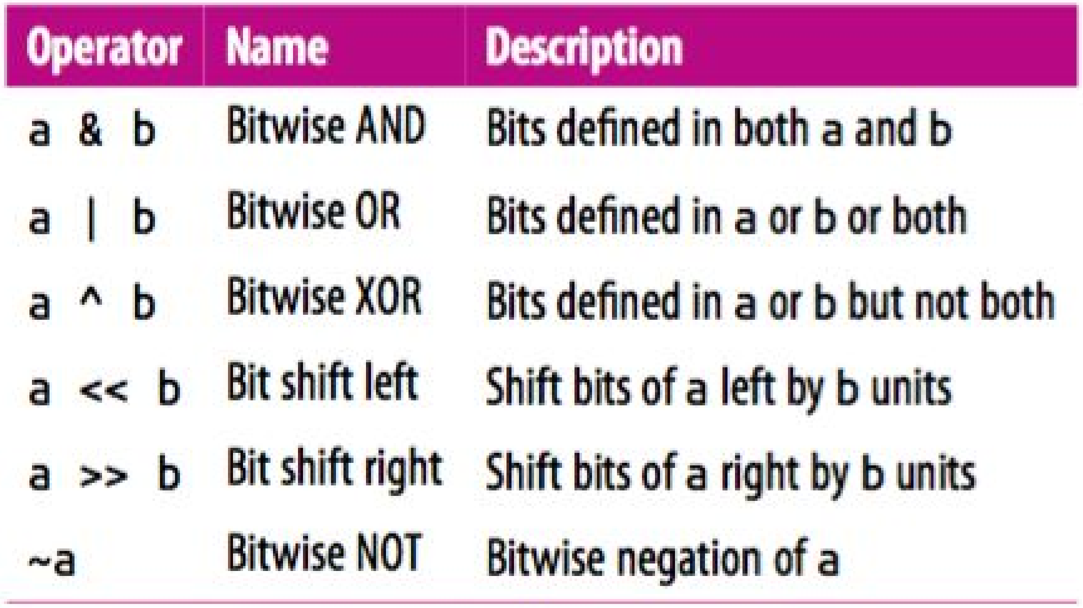
Bitwise operators



```
In [4]:  bin(10)

Out [4]: '0b1010'
```

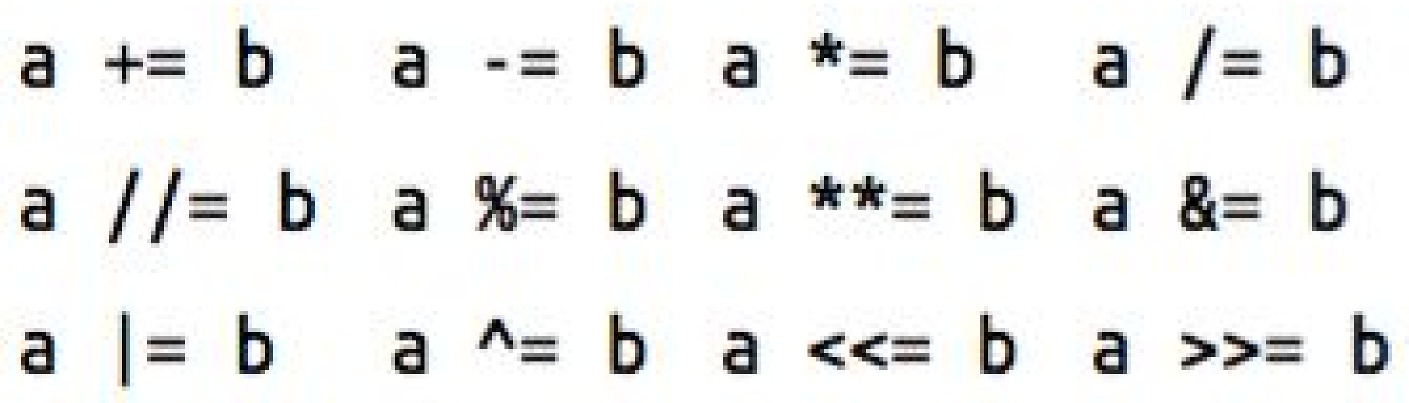


```
In [6]:  4 | 10

Out [6]: 14

In [7]:  bin(4 | 10)

Out [7]: '0b1110'
```



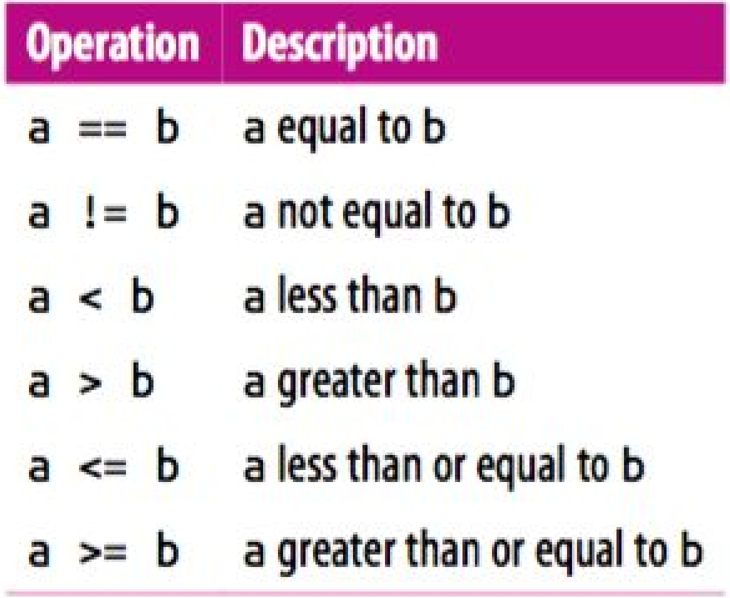
Operator	Name	Description
a & b	Bitwise AND	Bits defined in both a and b
a | b	Bitwise OR	Bits defined in a or b or both
a ^ b	Bitwise XOR	Bits defined in a or b but not both
a << b	Bit shift left	Shift bits of a left by b units
a >> b	Bit shift right	Shift bits of a right by b units
~a	Bitwise NOT	Bitwise negation of a

Augmented assignment operators



```
a  +=  b    a  -=  b   a  *=  b    a  /=  b
a  //=  b  a  %=  b   a  **=  b  a  &=  b
a  |=  b    a  ^=  b   a  <<=  b  a  >>=  b
```

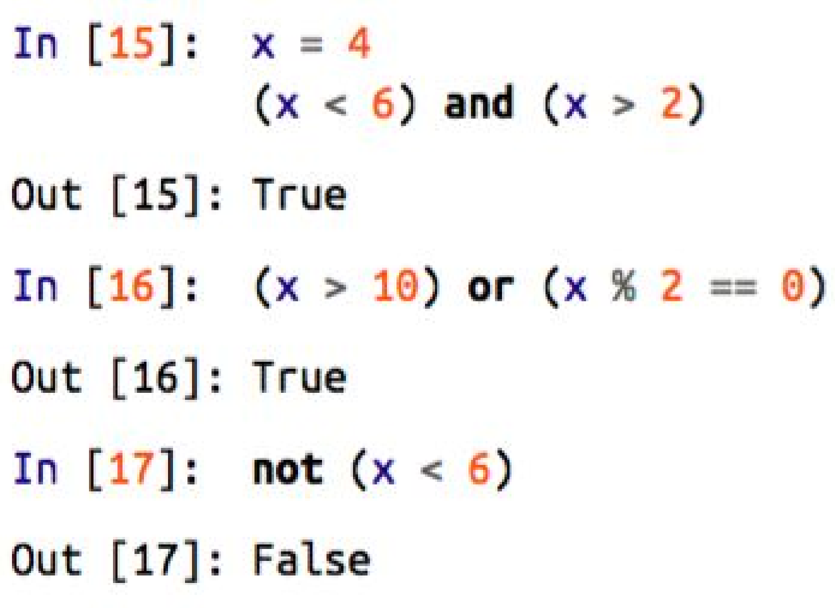
Comparison operators



Operation	Description
a == b	a equal to b
a != b	a not equal to b
a < b	a less than b
a > b	a greater than b
a <= b	a less than or equal to b
a >= b	a greater than or equal to b

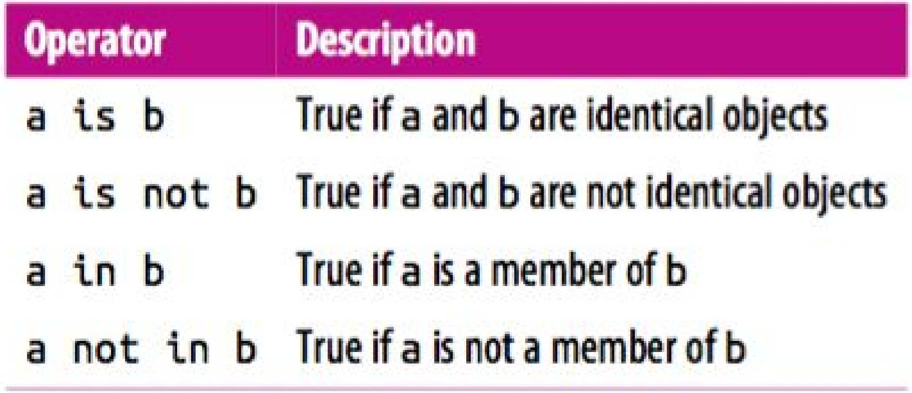
Boolean operations

- The boolean values are True and False (capitalized)
- The operators are `and`, `or` and `not`

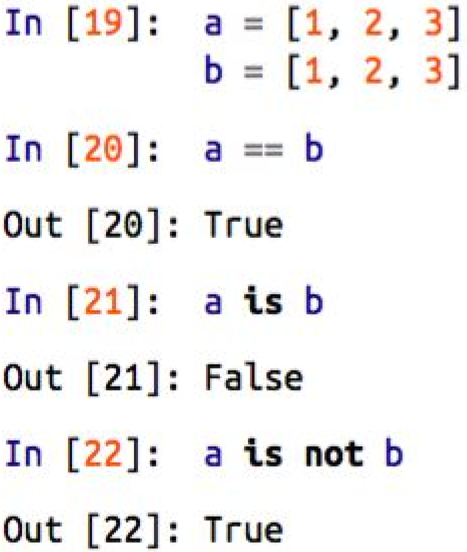


```
In [15]:  x = 4
          (x < 6) and (x > 2)

Out [15]: True

In [16]:  (x > 10) or (x % 2 == 0)

Out [16]: True

In [17]:  not (x < 6)

Out [17]: False
```

Identity operators



Operator	Description
`a is b`	True if a and b are identical objects
`a is not b`	True if a and b are not identical objects
`a in b`	True if a is a member of b
`a not in b`	True if a is not a member of b



```
In [19]:  a = [1, 2, 3]
          b = [1, 2, 3]

In [20]:  a == b

Out [20]: True

In [21]:  a is b

Out [21]: False

In [22]:  a is not b

Out [22]: True
```