Simple tutorial on Python and IPython

Mauro Sozio

Telecom ParisTech

September 30, 2016

Python in a Nutshell

Python is a high-level, dynamically typed multiparadigm programming language Python is simple! Python code is often said to be almost like pseudocode

```
def quicksort(arr): if len(arr)<=1:
return arr
pivot = arr[len(arr) / 2]
left = [x for x in arr if x < pivot]
middle = [x for x in arr if x == pivot]
right = [x for x in arr if x > pivot]
return quicksort(left) + middle + quicksort(right)
print quicksort ([3 ,6 ,8 ,10 ,1 ,2 ,1])

# Prints "[1, 1, 2, 3, 6, 8, 10]"
```

Versions

There are two most commonly used Python versions:

- Python2.7.x (used in this tutorial!)
- Python3.x

Note: The two versions are incompatible!

Basic data types: Integer

Python has basic data types including integers, floats, booleans, and strings. They behave similarly to other programming languages.

```
x = 3
print type(x) # Prints "<type 'int'>"
print x       # Prints "3"
print x + 1   # Addition; prints "4"
print x - 1   # Subtraction; prints "2"
print x * 2   # Multiplication; prints "6"
print x ** 2  # Exponentiation; prints "9"
x += 1
print x  # Prints "4"
x *= 2
print x  # Prints "8"
y = 2.5
print type(y) # Prints "<type 'float'>"
print y, y + 1, y * 2, y ** 2 # Prints "2.5 3.5 5.0 6.25"
```

Basic data types: Boolean

```
t = True
f = False
print type(t) # Prints "<type 'bool'>"
print t and f # Logical AND; prints "False"
print t or f  # Logical OR; prints "True"
print not t   # Logical NOT; prints "False"
print t != f  # Logical XOR; prints "True"
```

Basic data types: Strings

```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print hello        # Prints "hello"
print len(hello)   # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print hw  # prints "hello world"

hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print hw12  # prints "hello world 12"

# sprintf style string formatting
print hw12 # prints "hello world 12"

# substring replacement
print hw.replace('world', 'telecom') # prints "hello telecom"

# find the index of "world"
print h.index('world') # prints "6"
```

Containers: List

Python includes several built-in container types: lists, dictionaries and sets. A list is the Python equivalent of an array.

```
xs = [3, 1, 2, 4]# Create a list
print xs, xs[2]  # Prints "[3, 1, 2, 4] 2"
print xs[-1]     # Negative indices count from the end of
                 # the list; prints "4"
print xs[1:3]    # Get a slice from index 1 to 3 (exclusive);
                 # prints "[1 , 2]"

xs[2] = 'foo'    # Lists can contain elements of different types
print xs         # Prints "[3, 1, 'foo', 4]"
xs.append('bar') # Add a new element to the end of the list
print xs         # Prints "[3, 1, 'foo', 4, 'bar']"
x = xs.pop()     # Remove and return the last element of the list
print x, xs      # Prints "bar [3, 1, 'foo', 4]"
```

Containers: Tuple

A tuple is an immutable ordered list of values. Immutable means the tuple cannot be modified once initialized.

```
t = (5, 6) # Create a tuple
print t[1] # Prints "6"
t[0] = 4   # Raise an error!
```

Tuples are very useful for storing a small amount of data.

Containers: Dictionary

A dictionary stores (key, value) pairs, similar to a Map in Java.

```
d = {}                          # Create an empty dictionary
d = {'cat': 'cute', 'dog': 'furry'}  # Create new dictionary
print d['cat']                  # Get an entry from a dictionary; prints "cute"
print 'cat' in d                # Check if a dictionary has a given key;
                                # prints "True"
d['fish'] = 'wet'               # Set an entry in a dictionary
print d['fish']                 # Prints "wet"
del d['fish']                   # Remove an element from a dictionary
```

Containers: Sets

A set is an unordered collection of distinct elements.

```
animals = {'cat', 'dog'}
print 'cat' in animals    # Check if an element is in a set; prints "True"
print 'fish' in animals   # prints "False"
animals.add('fish')       # Add an element to a set
print 'fish' in animals   # Prints "True"
print len(animals)        # Number of elements in a set; prints "3"
animals.add('cat')        # Adding an element that is already in the set doe
print len(animals)        # Prints "3"
animals.remove('cat')     # Remove an element from a set
print len(animals)        # Prints "2"
```

IF Statement

An IF statement is defined using three keywords: **if**, **elif** and **else**:

```
x=5
if x < 5:
    print 'x < 5'
elif x < 10:      # abbr. for else if
  print '5<=x<10'
else:
  print 'x>=10'

# Prints '5 <= x < 10'
```

Loops

```
sum = 0
for i in range(1, 101): # generate a sequence of [1, 100]
    sum += i
print 'the sum is %d' % sum # Prints the sum is 5050
```

Note: Indentation in Python is important! If we type

```
sum = 0
for i in range(1, 101): # generate a sequence of [1, 100]
sum += i
print 'the sum is %d' % sum # Prints the sum is 5050
```

we get an error:

```
  File "<ipython-input-3-bc6d1a109db0>", line 2
    print i
        ^
IndentationError: expected an indented block
```

Loops for Lists and Dictionaries

```
animals = ['cat', 'dog', 'monkey']
for animal in animals :
  print animal
# Prints "cat", "dog", "monkey", each on its own line.

d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
  legs = d[animal]
  print 'A %s has %d legs' % (animal, legs)
# Prints "A person has 2 legs", "A spider has 8 legs";
# "A cat has 4 legs"
```

Functions

We often define functions to take optional keyword arguments, like this:

```
def hello(name, loud=False):
  if loud:
    print 'HELLO, %s!' % name.upper()
  else:
    print 'Hello,%s' % name

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True) # Prints "HELLO, FRED!"
```

I/O operations

Creating a file object with "open(filename, mode)". Mode can be 'r' (only reading),'w' (only writing erasing existing files),'a' (opens the file for appending) 'r+' (reading and writing).

```
f = open('workfile', 'w')
print f
#Prints open file 'workfile', mode 'w' at 80a0960
```

Reading from file. The following reads a list of integer pairs.

```
for line in f:
    print line
    u,v= [ int(x) for x in line.split() ]
```

I/O operations

Writing a string:

```
f.write('This is a test\n')
```

To write something other than a string, it needs to be converted first:

```
value = ('the answer', 42)
s = str(value)
f.write(s)
```

Call "f.close()" to close it and free up any system resources used.

```
f.close()
```

Advanced Topics 1: List Comprehensions

Often, we may want to process the elements in a list (change data types, perform operations etc.) Example:

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
  squares.append(x ** 2)
print squares # Prints [0, 1, 4, 9, 16]
```

You can make this code simpler using a list comprehension:

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print squares # Prints [0, 1, 4, 9, 16]
```

Advanced Topics 1: List Comprehensions

List comprehensions can also contain conditions:

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0] 3
print even_squares # Prints "[0, 4, 16]"
```

Advanced Topics 2: Classes

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
            print 'HELLO, %s!' % self.name.upper()
        else:
            print 'Hello, %s' % self.name

g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

Programming in Python

There are many ways to develop python programs:

- Use the python interactive shell: type python to start the shell
- Use any text editor (e.g.,vim, SublimeText) for coding and run programs in console with " python test.py"
- Use all-in-one IDEs: PyCharm, Visual Studio. Easy for debugging.

Introducing IPython (now called Jupiter)

[1] IPython is (mainly) a powerful interactive shell for Python. It has a powerful build-in notebook feature:

- lets you write and execute Python code in your web browser.
- supports syntax highlighting and tab completion
- merge Python codes with notes and equations to make nice notes.
- Very popular and widely used in scientific computing.

[1] `http://ipython.org`

Installing IPython

Installing and running IPython is easy. From the command line, the following will install IPython:

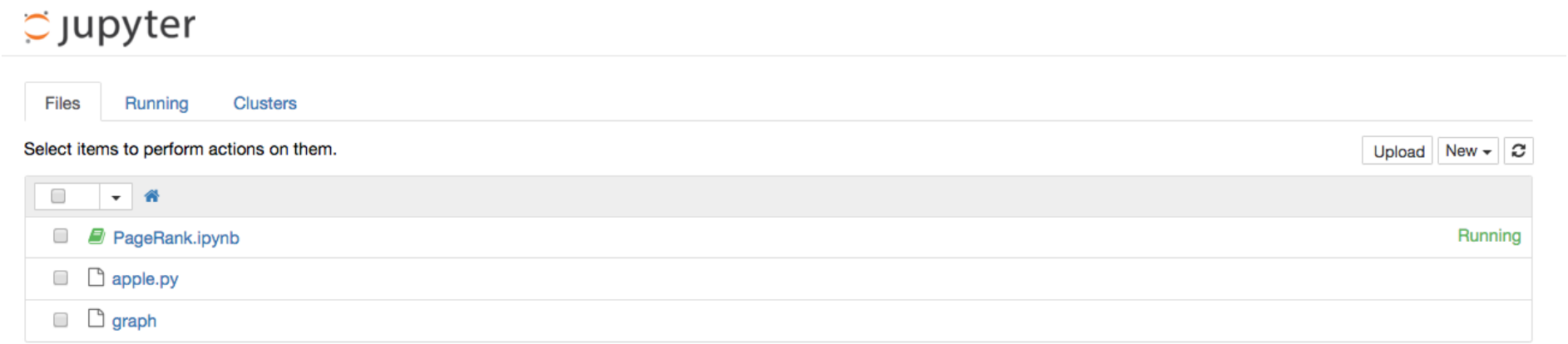
```
pip install "ipython[notebook]"
```

Once you have IPython installed, start it with this command:

```
ipython notebook
```

An IPython notebook server will startup on your computer (typically `http://localhost:8888`).

IPython Notebook Startup



Click New → Notebook (Python 2) to create a new Python notebook.

IPython Notebook Basics

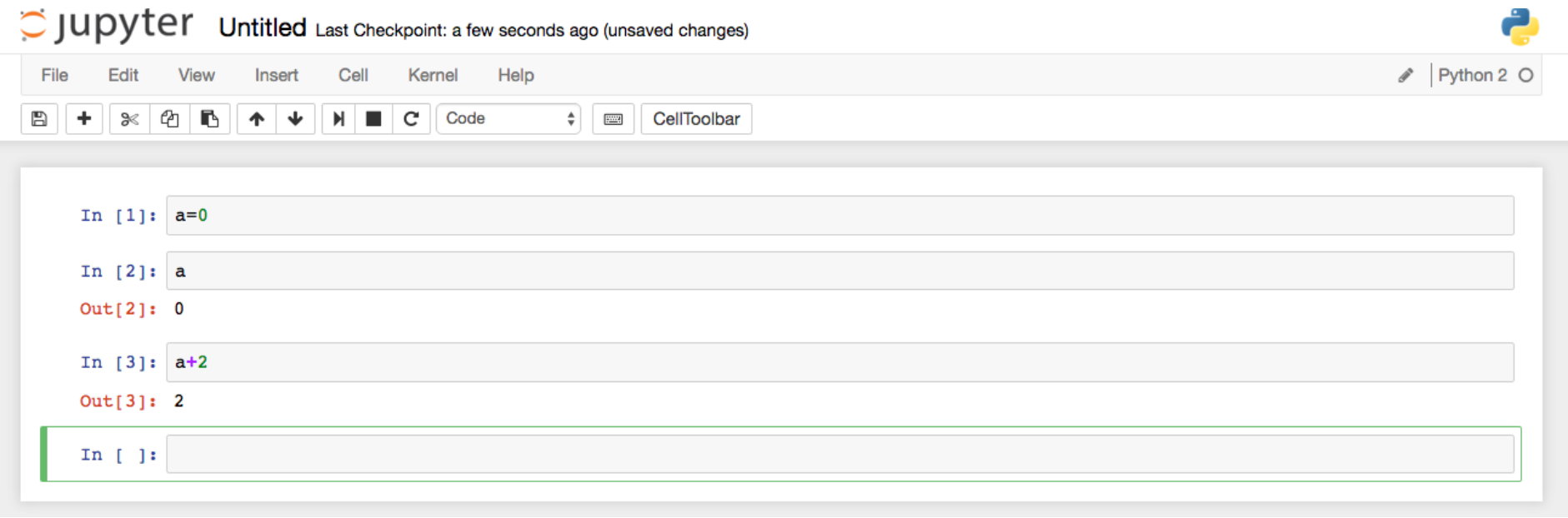
An IPython notebook is made up of a number of cells. Each cell can contain either Python code, markdown text, or latex equations. You can execute a cell by clicking on it and pressing **Shift-Enter**.

```
for u in S:
    r[u]=1.0/len(S)
for i in range(1,10):
    print r
    r = multMr(M,r)
    for u in r:
        r[u]=r[u]*beta+ (1-beta)/len(S)
```

```
{1: 0.16666666666666666, 4: 0.16666666666666666, 6: 0.16666666666666666, 8: 0.16666666666666666, 11: 0.16666666666666
666, 12: 0.16666666666666666}
{1: 0.08333333333333333, 4: 0.16666666666666666, 6: 0.16666666666666666, 8: 0.3333333333333333, 11: 0.083333333333333
33, 12: 0.16666666666666666}
{1: 0.16666666666666666, 4: 0.08333333333333333, 6: 0.16666666666666666, 8: 0.3333333333333333, 11: 0.16666666666666
66, 12: 0.08333333333333333}
{1: 0.16666666666666666, 4: 0.16666666666666666, 6: 0.08333333333333333, 8: 0.25, 11: 0.16666666666666666, 12: 0.1666
6666666666666}
{1: 0.125, 4: 0.16666666666666666, 6: 0.16666666666666666, 8: 0.25, 11: 0.125, 12: 0.16666666666666666}
{1: 0.125, 4: 0.125, 6: 0.16666666666666666, 8: 0.3333333333333333, 11: 0.125, 12: 0.125}
{1: 0.16666666666666666, 4: 0.125, 6: 0.125, 8: 0.29166666666666663, 11: 0.16666666666666666, 12: 0.125}
{1: 0.14583333333333331, 4: 0.16666666666666666, 6: 0.125, 8: 0.25, 11: 0.14583333333333331, 12: 0.16666666666666666}
{1: 0.125, 4: 0.14583333333333331, 6: 0.16666666666666666, 8: 0.29166666666666663, 11: 0.125, 12: 0.145833333333333
1}
```

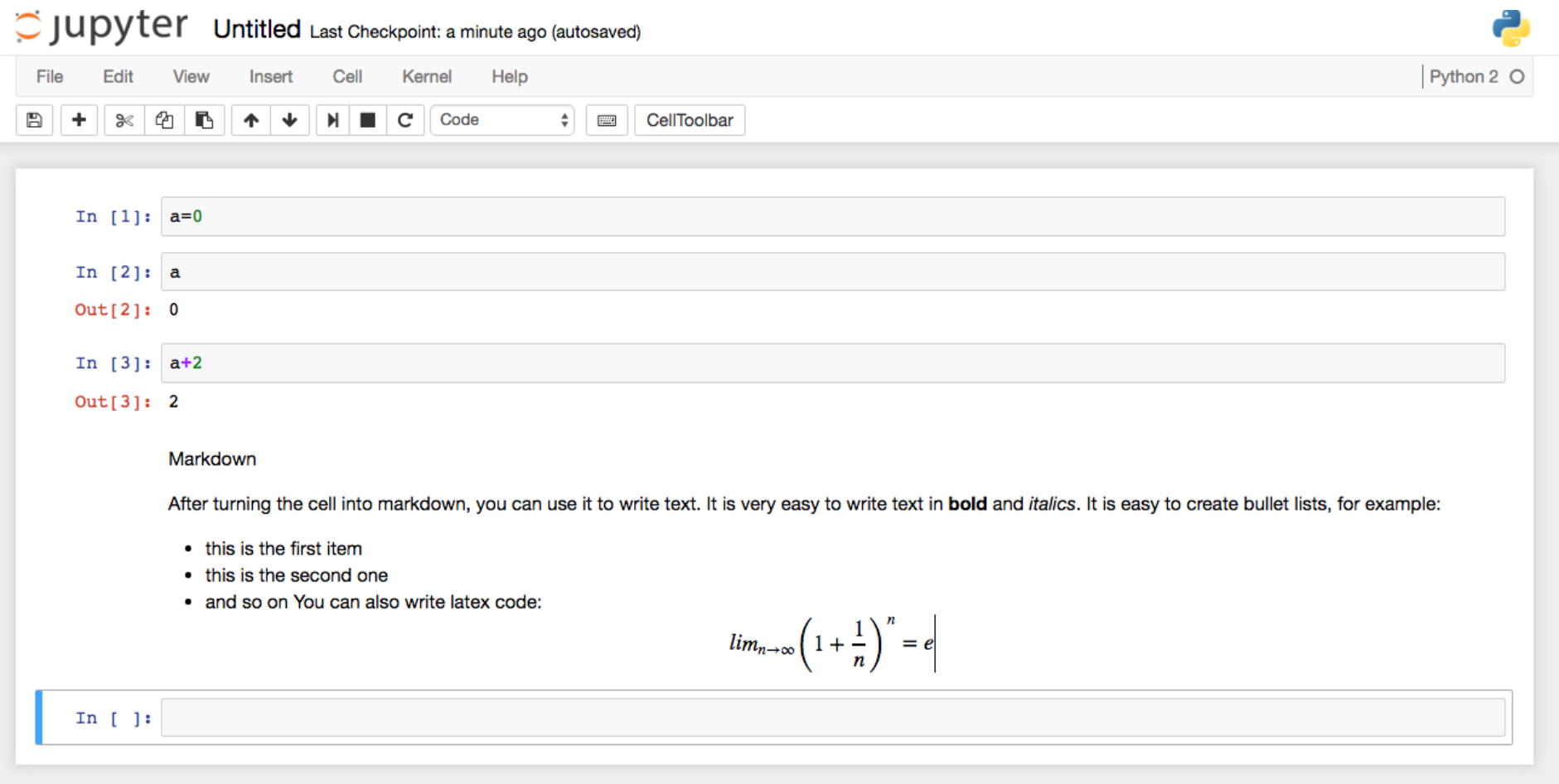
IPython Notebook Basics

Global variables are shared between cells. Executing the second cell thus gives the following result:



IPython Notebook Basics

An IPython cell also supports Markdown text. It is easy to write in bold or italic fonts or to write Latex equations. Switch cell format to **Markdown**.



IPython Notebook Basics

- Remember to save your file!
- Check out come nice IPython notebooks at `http://nbviewer.jupyter.org`

Python Tutorials

- http://cs231n.github.io/python-numpy-tutorial/
- OfficialPythontutorial https://docs.python.org/2/tutorial/
- Python online doc https://www.python.org/doc/
- IPython online doc http://ipython.org/documentation.html

Acknowledgements

Most of the material has been adapted from the Website `http://cs231n.github.io/python-numpy-tutorial/` and other Web tutorials with the help of Pengcheng Yin.