A Quick Introduction to Python

Prepared by Assil Ksiksi

March 2014

Contents

What is Python?

- Object-oriented
- Dynamically typed
- General purpose (scientific, web development, mobile app dev, etc.)

Section 0. Setting Up the Environment

1. Installing Anaconda

Anaconda is a custom installer for Python that includes the most used Python libraries. It is available for all major operating systems and works pretty much the same across them all, making troubleshooting less of a problem. Also, it includes `IPython`, outlined below in step 4.

To download Anaconda, visit its Downloads page. Scroll down a bit to see links to the installers. The installation process is quite straightforward. Do not change any of the options during the installation, except perhaps the installation directory.

To verify that Python was installed correctly, type `python --version` in the command prompt on Windows or the terminal on OS X/Linux. If you don't get an error, you're good to go.

2. Creating a Directory

Create a directory for the workshop. We'll be saving our work in this directory. An example could be `py-workshop` on the Desktop. Navigate to this directory using your prompt's `cd` command before proceeding.

3. The IPython Interpreter

IPython is a special version of Python that adds a good amount of useful features to the Python interpreter. In addition, it comes with a Notebook version that allows you to interactively run your code in a web browser. Since Python is a dynamic language, you do not need to compile your code - simply type the code in a block and hit `Enter` to view the results live.

Type `ipython notebook` in your Terminal. This will start up the IPython notebook interface through your default web browser. Create a new notebook. It will be saved in the current directory.

IPython also allows you to include images, text, and video along with your code. More details can be found in the IPython documentation.

Section 1. Variables, Types, and User Input

```python
# Setup some variables of different types
a = 5
b = 12.0
c = 'apple'
d = True
e = [1, 5.0, False, 'orange']
f = {1: 'kiwi', 'banana': 3}

# Simple operations and access
print (a + 10) ** 2
print 'Value = %f' % (b * a)
print c + ' ' + c
print d
print e[1], e[-1]
print f['banana']

# Take user input
name = raw_input('Enter your name: ')
print 'Hello, %s!' % name
```

Section 2. Flow Control and Looping

```python
# Simple if-elif-else block
if a < 5:
    print 'Less.'
elif a == 5:
    print 'Equal.'
else:
    print 'Greater.'

# Iterate over a range of numbers (1-10)
m = 10

for i in range(1, m+1):
    print i

# Iterate over a list
e.append('mango')
e.append(33.5)

for item in e:
    print e
```

```python
# Simple while loop
i = 0

while i < 5:
    print 'Iteration: %d' % i
    i += 1
```

Section 3. Functions and File I/O

```python
# Add two numbers
def add(x, y):
    return x + y

# Greeting with default name
def greeting(name='world'):
    return 'Hello, %s!' % name

# Call above functions
s = add(10, 20)
t = greeting()
u = greeting('Assil')

print s
print t
print u

# Create a new file in current directory and write 1-10 on seperate lines
f = open('nums.txt', 'w')

for i in range(1, 11):
    f.write('%d\n' % i)

f.close()

# Open above file for reading
f = open('nums.txt')

# Two ways to get contents of file
for line in f:
    print line

lines = f.readlines() # List of lines
```

Section 4. Imports and The Standard Library

For this part, you'll need to create a Python script. On Unix, simply type `touch test.py` in the terminal. On Windows, you'll need to create a new file using Windows Explorer. Open the `py-workshop` folder in Explorer, right-click, and navigate to `New > Text Document`. Rename the new document to `test.py`. Make sure the extension is not `.txt`.

Type the following into `test.py`:

```python
def add(x, y):
    return x + y

a = 15
b = 'apple'
```

Save `test.py`, and close it. Go back to your IPython notebook.

```python
# Import from the standard library
import math
from math import pi

# Import external Python script
import test

# Get variables from external script
a = test.a
b = test.b

print a, b

n = test.add(a, b)
p = math.sqrt(n)
q = math.pow(pi, 2)

print n, p, q
```

Section 5. Project

The project combines everything covered above to create a relatively useful Python application. This application will do the following:

1. Ask the user to enter the URL of a valid website.
2. Ask the user for a path to the folder to save the above website's HTML in as a `.html` file.
3. Download the contents of the website's homepage in HTML and save it.
4. Tell the user where to find the downloaded file.

Let's do the easy stuff first, and that's asking the user for the URL and location. It's good practice to write the comments for your code before writing the code itself. This helps you organize your thinking.

```python
# Prompts
url = raw_input('Enter a valid URL: ')
path = raw_input('Enter a path (with trailing slash): ')
full_path = path + 'page.html'

# Create the file
f = open(full_path, w)

# Get the webpage's contents as text (not implemented)
page = ?

# Save the file
f.write(page)
f.close()

# Give user status
print 'Done! File at: %s' % full_path
```

Now we need to figure out how to download the webpage. There's actually a module in the standard library that can do just that. It's called `urllib2`. To use it, you'll have to import it first.

Put the downloading code in its own function - call it `get_html` - so it can be re-used in other scripts.

```python
import urllib2

def get_html(url):
    # Get the page's contents
    response = urllib2.urlopen(url)

    # Get the body of the page (HTML)
    text = response.read()

    return text
```

After adding the `get_page` function and filling in the missing function call, we get the following code:

```python
import urllib2

def get_html(url):
    # Get the page's contents
    response = urllib2.urlopen(url)

    # Get the body of the page (HTML)
    text = response.read()

    return text

# Prompts
url = raw_input('Enter a valid URL: ')
path = raw_input('Enter a path (with trailing slash): ')
full_path = path + 'page.html'

# Create the file
f = open(full_path, w)

# Get the webpage's contents as text (not implemented)
page = get_page(url)

# Save the file
f.write(page)
f.close()

# Give user status
print 'Done! File at: %s' % full_path
```

How about we save the code into its own Python script? Follow the steps mentioned in the previous section to create a new Python file and then copy and paste the code into that file. Assume we named the file save_page.py.

To run it, in the ommand prompt (or terminal), type `python save_page.py`. You should be able to type the URL and path in the prompt and then see the result.