

Python and scientific computation

LinuxDays 2008

19. February 2008

Georges Schutz

Table of Contents

Preamble.....	2
An Introduction to Python for scripting.....	2
Principal Python concepts.....	2
Python as a calculator.....	2
IPython the preferable interactive shell.....	3
Python scripts.....	5
Defining functions.....	6
Debugging source code.....	7
Exercise.....	9
Some words about object orientation.....	9
IDE Eclipse.....	9
Numeric and scientific extensions.....	10
SciPy & NumPy.....	10
PyLab & Matplotlib.....	10
Applications an Examples.....	10
Useful references.....	10

Preamble

This document is written to provide a handout to the participants of the tutorial “Python and scientific computing” at the LinuxDays 2008. In the same time the content of this document should provide enough background information for an potential user to set up python with all its necessary extensions in order to use it for scientific computing.

In this paper we will suppose that Linux is used as OS (Kubuntu 7.10 distribution) even if python exists for other platforms. All proposed examples will be possible on other OS with only slight modifications. Additionally we will use the integrated development environment Eclipse with the PyDev extension for convenience.

The content of this document is based on different sources that are listed in the references part at the end of the document.

An Introduction to Python for scripting

This first chapter will provide an overview of Python as an environment for scripting and interactive programming.

Principal Python concepts

Python is on the one hand an interpreter that you can use like a interactive shell on the other hand it is a modern multi-platform object oriented programming language. Python is widely used in different context going from shell scripting over web-application to complete standalone programs.

The scope of this document is restricted to the use of python in the scientific computing. As this is a large domain only comment tasks will be treated but note that python gains more and more interest in all scientific domains.

Python as a calculator

We will start with python as an interactive shell for doing simple calculations in order to familiarize with the environment.

Open a shell on your system (K – System – Konsole) and lunch python

```
$ python [ENTER]
Python 2.5.1 (r251:54863, Oct 5 2007, 13:36:32)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

With this you are in the python interpreter (the interactive python shell) and you can type any python command.

You can simply write some basic arithmetic calculations like:

```
>>> (1+3)*3/(12-2*3) [ENTER]
2
>>> (1+3)**2 / ((2-7)* -2 *(1-3**3)) # integer calculation
-1
>>> (1.+3)**2 / ((2-7)* -2 *(1-3**3)) # float calculation
-0.061538461538461542
>>>
```

Without going into details here, note that you'll have to be careful concerning the implicit used calculation depending on the involved value types.

Of course there are a lot of other possibilities to use python but as the workshop focus on scientific use of python it is important to have an environment supporting rapid prototyping and testing of calculations and methodologies we will first have a look at a more comfortable shell for python.

```
>>> [Ctrl+d] # exit python
```

IPython the preferable interactive shell

Skipping the details, IPython is an extension of the default python shell with a lot of useful features like [2]:

- Completion in the local namespace,
- Numbered input/output prompts with command history
- User-extensible 'magic' commands
- Complete system shell access
- Session logging and restoring
- Easy debugger access
- Profiler support
- ...

Before using the IPython extension it has to be installed. On the workshop systems this was already done but on Debian based systems like Ubuntu/Kubuntu this is a simple task:

```
& sudo aptitude install ipython
```

To launch IPython now simply run the following command on the shell:

```

& ipython
Python 2.5.1 (r251:54863, Oct  5 2007, 13:36:32)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.1 -- An enhanced Interactive Python.
?      -> Introduction to IPython's features.
%magic -> Information about IPython's 'magic' % functions.
help   -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.

In [1]:

```

You are informed about some basic elements of IPython and IPython is now waiting for interactive input. Test some simple calculations in order to see ipython in action.

```

In [1]: a = 10

In [2]: b = 20

In [3]: x = a + b

In [4]: x
Out[4]: 30

In [5]:

```

Note: all commands are numbered and can be called using this number.

The magic command %hist shows the current command history:

```

In [5]: %hist
1: a = 10
2: b = 20
3: x = a + b
4: x
5: _ip.magic("%hist ")

```

In this example 3 variables “a”, “b” and “x” are created. The command who shows the user defined variables.

```

In [6]: who
a      b      x

In [7]: del( a, b )

In [8]: who
x

```

To recall one or more commands use the input history list “In”

```
In [9]: exec In[1:3]
```

```
In [10]: who
```

```
a      b      x
```

It is also possible to build macros out of yet executed commands with the magic command “%macro name line-numbers”

```
In[11]: %macro 'BuildVars' 1-3 4
```

```
Macro `BuildVars` created. To execute, type its name (without quotes).
```

```
Macro contents:
```

```
a = 10
```

```
b = 20
```

```
x = a + b
```

```
x
```

```
In [12]: BuildVars
```

```
Out[13]: Executing Macro...
```

```
Out[13]: 30
```

There are other interesting possibilities in IPython using magic commands but they will not be analyzed in this workshop, see the IPython help for further details.

But one magic command is very important and will be analyzed in detail later.

Python scripts

A python script is principally python source file where different python instructions are written. Such a script can be interpreted by python.

```
##This is a simple python script
print "Some variables are initialized"
a = 10
b = 20
print "a: %s, b: %s; a+b=%s" % (a,b,a+b)
```

To run this file as a python script give the filename to python

```
$ python simplePythonscript.py
Some variables are initialized
a: 10, b: 20; a+b=30
$
```

In IPython a script can be run using the magic “%run” command

```
In [1]: run simplePythonscript.py
Some variables are initialized
a: 10, b: 20; a+b=30

In [2]: who
a      b

In [3]:
```

Defining functions

As in any other programming language it is possible to write a functions that can be called later to provide a specific functionality.

It is possible to define a function directly in the python interpreter or in a python script.

In IPython

```
In [1]: def addplus1(a,b):
...:     d = a + b + 1
...:     return d
...:

In [2]: addplus1(10,20)
Out[2]: 31
```

This can off cause also be written to a python source file

```
In [3]: %save 'functions' 1 2
The following commands were written to file `functions.py`:
def addplus1(a,b):
    d = a + b + 1
    return d

addplus1(10,20)
```

Open the file functions.py in an editor (K->Utilities->kate) and modify it as follows:

```
def addplus1(a,b):
    d = a + b + 1
    return d

def addminus1(a,b):
    return a + b -1

a = 10
b = 20
c = -10
print addplus1(a,b)
print addminus1(b,c)
```

Debugging source code

To execute this script in a debug mode we will use the IPython environment again.

```
%run -d functions
Breakpoint 1 at
  /mnt/usb1/LinuxDays2008/ScientificPython/src/functions.py:1
NOTE: Enter 'c' at the ipdb> prompt to start your script.
> <string>(1)<module>()

ipdb> c
> /mnt/usb1/LinuxDays2008/ScientificPython/src/functions.py(1)<module>()
1----> 1 def addplus1(a,b):
        2     d = a + b + 1
        3     return d

ipdb>
```

You are now at the IPython debugger prompt and can interact with the debugger:

- c to continue running the script (first time c starts the script)
- s step: steps through the code with entering the functions
- n next: steps through the code without going into called functions
- q quit: stops the debugger
- h help: help for the debug mode.

In the debug mode it is possible to inspect variables and execute commands but you can not modify existing variables. You can add code components or ignore code components.

You can set and manage breakpoints ...

See the help and the documentation for further information.

Now debug through the script to get some notions of the debugging possibilities. With "c" (continue) we started the debugging of the script and

are now at the first line of the script.

Note that the debugger always gives some neighborhood information of the current position.

With “n” (next) we will jump to the next executable line. You should note that the content of the function was not executed. The only thing that was done for now is that python now knows the definition of a function called `addplus1` with its synopsis and its location. This is introduced with the “def” keyword.

The next step will be the first instruction where the variable “a” will be created. With the python command “`dir()`” we can get all currently defined code elements.

```
ipdb> dir()
Out[1]:
['__builtins__',
 '__file__',
 '__name__',
 '__nonzero__',
 'addminus1',
 'addplus1']
ipdb> pinfo addminus1
Type:          function
Base Class:    <type 'function'>
String Form:   <function addminus1 at 0xb78b53e4>
Namespace:    Locals
File:
  /mnt/usb1/LinuxDays2008/ScientificPython/src/functions.py
Definition:    addminus1(a, b)
Source:
def addminus1(a,b):
    return a + b -1
ipdb>
```

The command “pinfo” gives detailed information of each object in the python environment. The command “`dir()`” or “pinfo” are examples of python commands that can be lunched during the debugging process.

After another 3 steps the variables “a”, “b” and “c” are defined in the environment. Check this with “`dir()`” and “pinfo” if you want.

The command “pp” means pretty print and gives you a readable output of the content of a variable.

```
ipdb> pp a
10
```

The debugger will in the next step call of the function `addplus1(a,b)`. With the command “n” this statement will be completely executed without going into the detailed execution of the function.

Use “s” (step) in order to force the debugger to look into the function.

```

ipdb> s
--Call--
> /mnt/usb1/LinuxDays2008/ScientificPython/src/functions.py(1)addplus1()
1---> 1 def addplus1(a,b):
      2     d = a + b + 1
      3     return d

```

The debugger informs you that a object call was done and gives the detailed location of the function.

If you now execute the `dir()` command you can see that the context of the debugger has changed and only “a” and “b”, the two parameters of the function are known. Even if they have the same name as the previous variables they are different objects (copies in this case).

Step further until the return instruction will be the next step with which the function context will be lost except of the returned result.

We will not go into further details here but keep in mind that the possibility of debugging a code is one of the most important elements of an environment used in the context of rapid prototyping and interactive coding. And with IPython this environment is ready for that.

With “c” (continue) you can run the rest of the code until the end of the script.

Exercise

Use the help integrated into the debugger to find out how to set a breakpoint to the `addminus1` function in order to rapidly run until this position without having to step through the whole code.

Some words about object orientation

Python is a complete object oriented programming language. This property should not be ignored because this is one of the reasons of the attractiveness of the language and with this spread out use of the language in nearly all IT domains.

But before going further with this subject we will have a look at an other well known integrated developing environment, Eclipse [5] and its python development extension.

IDE Eclipse

Once Eclipse and the PyDev extension is installed (the installation steps are not topic of this workshop, see other HowTo's you can find on the WEB) you can launch Eclipse by running the command “eclipse” on the shell or navigate through the K-Menu – Development - Eclipse that will also launch the IDE. Note that this is a general purpose IDE written in Java and principally designed as an IDE for the Java language. But in the meantime a lot of extensions exist for other programming languages like C/C++ and Python for example.

The IDE is a complete development environment and as such some how more complex than the simple IPython environment. In this workshop we will not go into details of using all features of Eclipse but we will need to go through some project setup in order to be able to use the advantages of the IDE.

- Set PyDev properties, to point to the possible python executable
- Define a PyDev project, Create new project
- Build a new python package
- Build a new python module
- Define a new Class object with some methods

Numeric and scientific extensions

In the previous chapter, we looked principally at the python as an environment for interactive scripting and programming. This chapter will focus on the extensions that exists for the use of python in scientific computing.

This part is very well introduced by a presentation realized by Eric Jones of Enthought and Travis Oliphant of the Brigham Young University [7].

SciPy & NumPy

These extensions defines some elements very useful for the scientific use like multidimensional arrays and the operations like matrix calculations ...

PyLab & Matplotlib

These extensions brings an easy to use plotting functionalities to python an essential element for visualizing data.

Applications an Examples

During the Workshop some examples that are provided with the extensions will be analyzed in order to get more familiar with some of the extensions.

Useful references

Getting more information

[1] Python <http://www.python.org>

[2] Python IDEs

<http://wiki.python.org/moin/IntegratedDevelopmentEnvironments>

[3] IPython <http://ipython.scipy.org>

[4] IPython local help </usr/share/doc/ipython/manual/manual.html>

[5] Eclipse: <http://www.eclipse.org>

[6] PyDev eclipse extension:

[7] Eric Jones and Travis Oliphant, "Introduction to scientific computing with Python" October 2004.

[Creative Commons Attribution-NonCommercial-ShareAlike 2.0 License]

This pages are licensed under a Creative Commons License.