Python
IPython

Stéphane Vialette

LIGM, Université Paris-Est Marne-la-Vallée

October 5, 2011

Outline

Outline

Functions

Description

The goal of IPython is to create a comprehensive environment for interactive and exploratory computing. To support this goal, IPython has two main components:

- An enhanced interactive Python shell.
- An architecture for interactive parallel computing.

All of IPython is open source (released under the revised BSD license).

Highlights

Tab completion

TAB-completion, especially for attributes, is a convenient way to explore the structure of any object youre dealing with. Simply type `object_name.<TAB>` and a list of the objects attributes will be printed.

Tab completion also works on file and directory names, which combined with IPythons alias system allows you to do from within IPython many of the things you normally would need the system shell for.

Highlights

Explore your objects

Typing `object_name?` will print all sorts of details about any object, including docstrings, function definition lines (for call arguments) and constructor details for classes.

The magic commands `%pdoc`, `%pdef`, `%psource` and `%pfile` will respectively print the docstring, function definition line, full source code and the complete file for any object (when they can be found). If automagic is on (it is by default), you dont need to type the % explicitly.

Highlights

The %run magic command

The %run magic command allows you to run any python script and load all of its data directly into the interactive namespace.

Since the file is re-read from disk each time, changes you make to it are reflected immediately (in contrast to the behavior of import).

%run also has special flags for timing the execution of your scripts (-t) and for executing them under the control of either Pythons pdb debugger (-d) or profiler (-p).

Highlights

Use the output cache

All output results are automatically stored in a global dictionary named `Out` and variables named `_1`, `_2`, etc. alias them.

For example, the result of input line 4 is available either as `Out[4]` or as `_4`.

Additionally, three variables named `_`, `__` and `___` are always kept updated with the for the last three results. This allows you to recall any previous result and further use it for new calculations.

Highlights

Input cache

A similar system exists for caching input.

All input is stored in a global list called `In`, so you can re-execute lines 22 through 28 plus line 34 by typing `exec In[22:29]+In[34]` (using Python slicing notation).

If you need to execute the same set of lines often, you can assign them to a macro with the `%macro` function.

Outline

IPython in action

Running IPython

```
$ ipython

Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]:
```

IPython in action

Exploring objects

```
In [1]: l?
Object ‘l‘ not found.

In [2]: l = [1, 2, 3]

In [3]: l?
list
Base Class:<type ’list’>
String Form:[1, 2, 3]
Namespace: Interactive
Length:    3
Docstring: list() -> new empty list
           list(iterable) -> new list initialized from iterable’s items

In [4]:
```

IPython in action

In

```
In [1]: print 'statement 1'
statement 1

In [2]: print 'statement 2'
statement 2

In [3]: print 'statement 3'
statement 3

In [4]: print 'statement 4'
statement 4

In [5]: print In[0]


In [6]: print In[1]
print 'statement 1'

In [7]:
```

IPython in action

In

```
In [6]: exec In[1]
statement 1

In [7]: exec In[1] + In[3:5]
statement 1
statement 3
statement 4

In [8]: exec In[1:5]
statement 1
statement 2
statement 3
statement 4

In [9]: print In[6]
exec In[1]

In [10]: exec In[6]
statement 1
```

IPython in action

Store

```
In [1]: x = 1

In [2]: x+1
Out[2]: 2

In [3]: x+2
Out[3]: 3

In [4]: x+3
Out[4]: 4

In [5]: _, __, ___
Out[5]: (4, 3, 2)

In [6]: _, __, ___
Out[6]: ((4, 3, 2), 4, 3)

In [7]: _, __, ___
Out[7]: (((4, 3, 2), 4, 3), (4, 3, 2), 4)
```

IPython in action

Store

```
In [1]: x = 1

In [2]: _
Out[2]: ''

In [3]: x+1
Out[3]: 2

In [4]: 1 + _
Out[4]: 3

In [5]: _ + __
Out[5]: 5

In [6]: _ + __ + ___
Out[6]: 10

In [7]: _, __, ___
Out[7]: (10, 5, 3)
```

IPython in action

Macro

```
In [1]: print 'statement 1'
statement 1

In [2]: print 'statement 2'
statement 2

In [3]: print 'statement 3'
statement 3

In [4]: %macro mymacro 1-3
Macro `mymacro` created. To execute, type its name (without quotes).
Macro contents:
print 'statement 1'
print 'statement 2'
print 'statement 3'
```

IPython in action

Macro

```
In [5]: mymacro
------> mymacro()
statement 1
statement 2
statement 3

In [9]: mysecondmacro = mymacro

In [10]: mysecondmacro
-------> mysecondmacro()
statement 1
statement 2
statement 3

In [14]: type(mymacro)
Out[14]: <type 'instance'>

In [15]:
```