

# IPython

## Components for Interactive Scientific Computing

IP[y]:

Fernando Pérez  
Brian E. Granger

`Fernando.Perez@berkeley.edu`

`bgranger@calpoly.edu`

Helen Wills Neuroscience Institute, U.C. Berkeley  
Physics, California Polytechnic State University, San Luis Obispo

SIAM CSE 09, Miami  
March 5, 2009

# Outline

- 1 Scientific Computing
  - Inherently exploratory
  - Python?
- 2 IPython: Interactive Python
- 3 Users

# Outline

- 1 Scientific Computing
  - Inherently exploratory
  - Python?
- 2 IPython: Interactive Python
- 3 Users

# Outline

- 1 Scientific Computing
  - Inherently exploratory
  - Python?
- 2 IPython: Interactive Python
- 3 Users

# Outline

- 1 Scientific Computing
  - Inherently exploratory
  - Python?
- 2 IPython: Interactive Python
- 3 Users

# Scientific computing

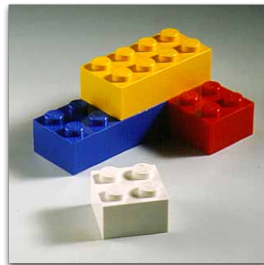
The computer as a microscope

- Problem's definition evolves as we understand it.
- No 'requirements' to build an application against.
- Mathematica, Maple, Matlab, IDL, etc.
  - All have an interactive environment.

## Applications



## Languages



# Python?

- **Primitive** interactive interpreter provided.
- Very good **introspection**.
  - All entities can be inspected at **runtime** by the language itself.
- **Dynamic reloading** of (most) code.
- Builtin documentation for objects (**docstrings**).
- Live and post-mortem interactive **debugging**.

**Note:** **Applications** can also be built (easier than in C/Fortran).

# Outline

- 1 Scientific Computing
  - Inherently exploratory
  - Python?
- 2 IPython: Interactive Python
- 3 Users



# What is IPython?

- 1 **A better Python shell**: object introspection, system access, 'magic' command system, ...
- 2 **An embeddable interpreter**: mix batch and interactive work.
- 3 **A flexible component**: tweak to taste for your project.
- 4 **An interactive component** plug into GUIs, browsers, etc.
- 5 **High level distributed/parallel computing**: next talk.

# Quick overview

## History

- Started in late 2001, ‘just one afternoon’...
- It seems to have filled a need.
- Organic community uptake, mostly ‘spare time’.
- Messy code, good functionality
- Many contributions from outsiders

## Tools

- Project hosting (site, wiki, mailing lists):  
<http://ipython.scipy.org> (courtesy of [Enthought](#))
- Code hosting and bug tracking:  
<http://launchpad.net/ipython> ([Canonical](#))
- Distributed Version control: [Bazaar](#).

# Status

Code structure (as of early 2009):

Language	LOC	Proportion
Python	37323	99.07%
Lisp	262	0.70%
Sh	51	0.14%
Objective C	37	0.10%

Data generated using David A. Wheeler's 'SLOCCount'

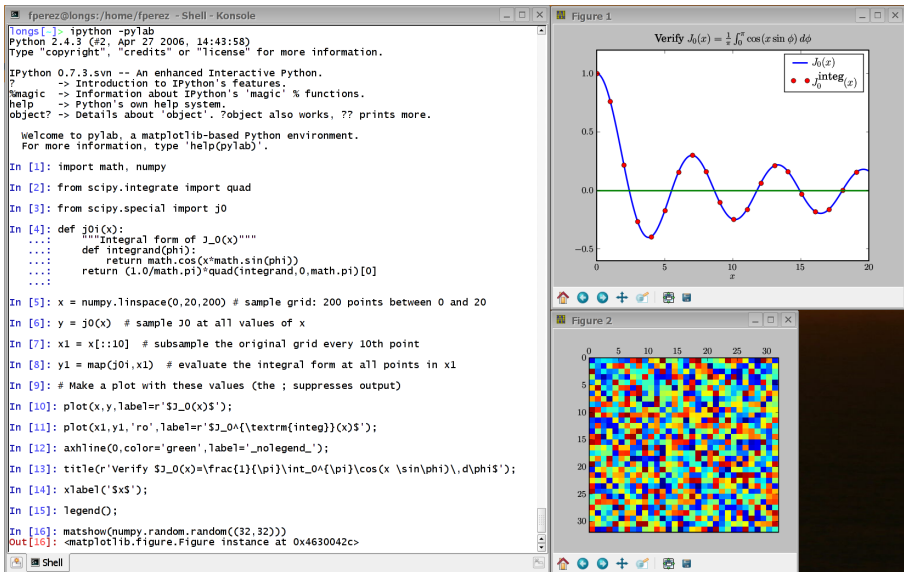
- Mailing lists:
  - Users: ~400 members
  - Developers: ~170 members
- Available on all Linux distributions. We provide OS X and Windows installers.

# Cast of Characters

- **Brian Granger** - Physics, Cal State San Luis Obispo
- Ville Vainio - CS, Tampere University of Technology, Finland
- Min Ragan-Kelley - Applied science, UC Berkeley
- Gael Varoquaux - Neurospin (Orsay, France)
- Robert Kern - Enthought
- Stefan van der Walt - Applied Math, U. Stellenbosch, South Africa
- Barry Wark - Neuroscience, U. Washington.
- Jorgen Stenarson - Sweden
- **Ondrej Certik** - Physics, U. Nevada Reno
- Laurent Dufrechou - France
- Vivian De Smedt - Belgium
- Darren Dale - Astronomy, Cornell
- *Many more I am forgetting...*

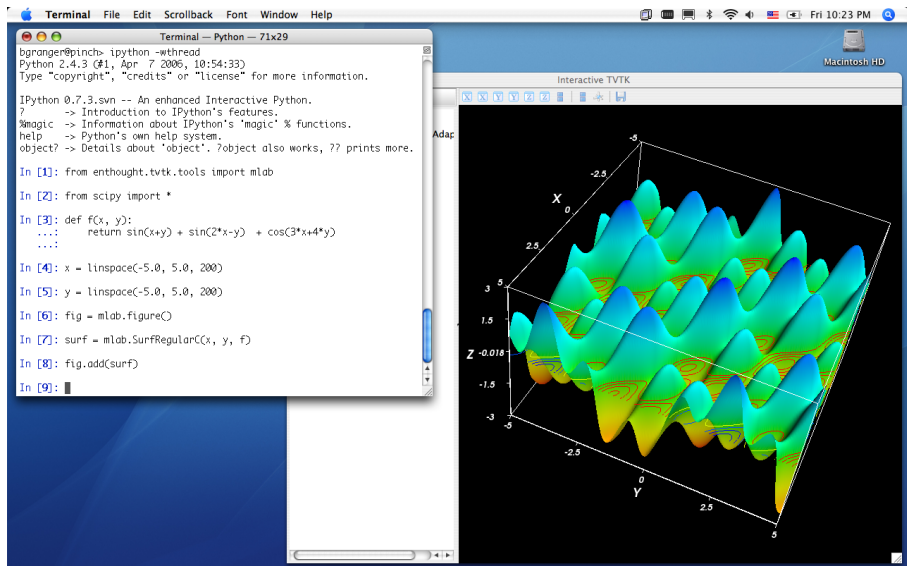
# Matlab-like interactive usage

Matplotlib: high quality plotting. Scipy: numerical algorithms.



# Multiple GUI systems

## VTK with WxWidgets

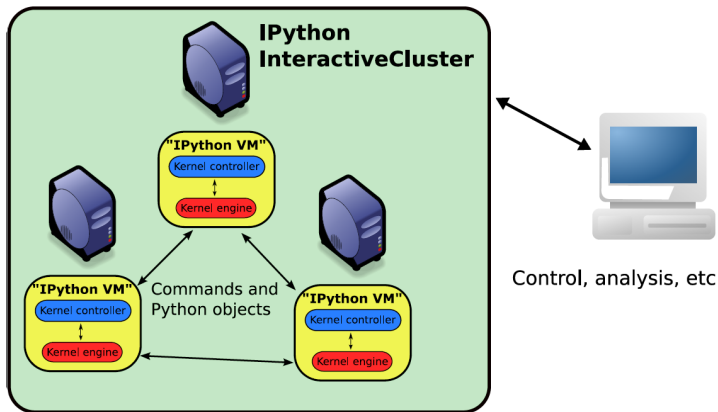


# Components

- Interactive app - terminal
- Embeddable shell (dbg, exploration, control)
- Widget (Wx, GTK, Tk, Qt)
- Network-aware computational engine
  - Interactive distributed and parallel computing
  - High-level interfaces for this
  - Complement, not replace MPI
- Node for visual-programming driven HPC - IPVision

# IPython for distributed and parallel computing

- Think of Python as 'the CPU'.
- IPython abstracts them over the network.
- Use interactively or not.





# Vision: visual programming

Michel Sanner, Scripps Institute, La Jolla.

The screenshot displays the VIPER (Visual Programming Environment) interface. The top menu bar includes File, Edit, and Networks. Below it, a toolbar contains buttons for Load Network, Save Network, and Run Network. The main workspace is divided into four panels: Input, Filter, Output, and Mapper. The Input panel contains Generic, Iterate, Python Eval, and Color Chooser. The Filter panel contains choose geom and Pass. The Output panel contains Print, get length, and Viewer. The Mapper panel contains Color, IndexedPolyLines, IndexedPolygons, and Instances. A yellow callout box points to the Viewer node, stating: "OpenGL based 3D visualization application input geometries; type: geom output1: lastId; type: None output2: viewer; type: Viewer Instance".

Below the main workspace, there are three network diagrams. The first network on the left shows a sequence of nodes: Read Image (lena.jpg), Scale, Show Image, Color Chooser, MyMacro, and Split (1-4). The second network in the center shows a Python Eval node (range(10)) connected to a Show Data node, which is connected to an Introspect node. The third network on the right shows a Python Eval node (range(10)) connected to an Introspect node, which is connected to a Print node.

Several inspection windows are open. The "node Python Eval, port command" window shows the data output of Port 0: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]. The "Introspect node Python Eval, node" window shows the internal state of the Python Eval node, including attributes like \_\_class\_\_, condition, deselectOptions, dirty, distScaleFactor, dynamicComputeFunction, editFuncWindowHeight, editFuncWindowWidth, editor, firstInstanciation, frozen, funcEditorDialog, highlightOptions, iconTag, id, innerBox, and inputPorts. The "Introspect node Python Eval, node" window also shows the source code of the Python Eval node, which is a class definition for VPER.stlib.PythonEval.

# Vision

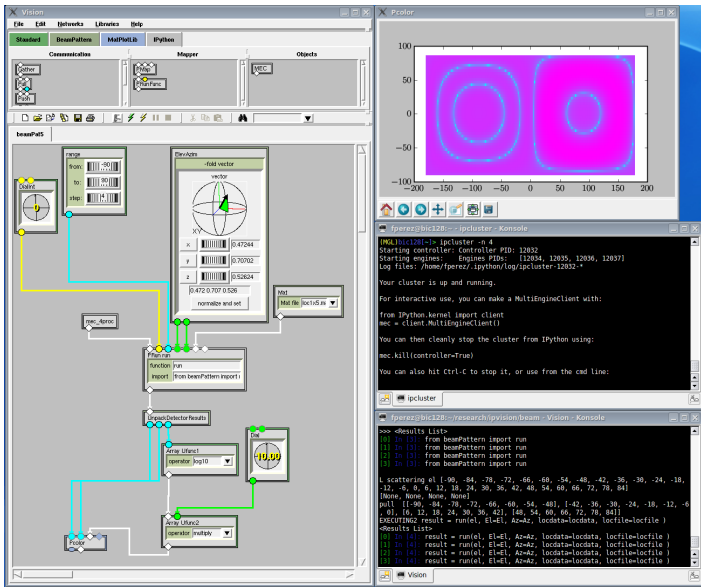
A generic framework for visual programming

The screenshot displays the ViPER (Visual Interactive Programming Environment) interface, a generic framework for visual programming. The interface is organized into several panels and windows:

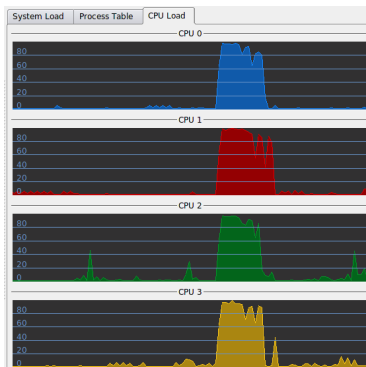
- Top Panel:** Contains menu options (File, Edit, Networks) and tabs for different modules: Std, User, Standard, MolKit, SymServer, and ImageLib. Below these are buttons for Load Network, Save Network, and Run Network.
- Mapper Panel:** Divided into four sections: mapper (Scale, Img To Heightfield), Input (Read Image, grab Image), output (Show Image), and filter (Filter Image, Invert Image).
- Network 0:** The main workspace for building a visual programming network. It shows a complex flow of data and operations:
  - Input:** Read Molecule from pdb, Assign Radii (with a checkbox for "scaled radii"), grab Image, and Scale.
  - Processing:** Select Atoms, Extract Atom Property radius, MSMS, Indexer-Polygons, Array UtilFunc2 (set to "multiply"), and CPK.
  - Output/Filtering:** Filter Image (set to "CONTOUR"), Show Image, and Color.
  - Control:** A Dial control is set to 9.55, connected to the Extract Atom Property radius node.
  - Visualization:** A Viewer window displays a 3D molecular model with colored spheres and a semi-transparent surface. A Color Map window shows a color gradient from blue to red, with Min: 0.0 and Max: 235.0.
- Viewer:** A window showing a 3D molecular model with colored spheres and a semi-transparent surface.

# IPVision: distributed computing

With: Michel Sanner, Jose Unpingco, Ananth Devulapalli [Ohio Supercomputing Center/OSU], Min Ragan-Kelley [UC Berkeley]



- Easy access to distributed computing for non-experts
- Vision allows any python library to be accessed (after wrapping)
- Users get to use all their cores...



# Part of an ecosystem

Rapid flow of developments across open source projects

- [Matplotlib](#): interactive plotting and GUI support.
- [Numpy](#): sharing code for testing support.
- [Nose](#): automatic testing system
- [Sphinx](#): high quality documentation.

# Outline

- 1 Scientific Computing
  - Inherently exploratory
  - Python?
- 2 IPython: Interactive Python
- 3 Users

# Sympy: symbolic computing

Ondrej Certik & team, U. Nevada Reno

```
maqroll[siam]> isympy
Python 2.5.2 console for SymPy 0.6.3

These commands were executed:
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z = symbols('xyz')
>>> k, m, n = symbols('kmn', integer=True)
>>> f, g, h = map(Function, 'fgh')

Documentation can be found at http://sympy.org/

In [1]: integrate(x)
Out[1]:

$$\frac{x^2}{2}$$


In [2]: integrate(exp(x**2))
Out[2]:

$$\frac{-i \cdot \sqrt{\pi} \cdot \operatorname{erf}(i \cdot x)}{2}$$


In [3]: integrate(exp(x**3))
Out[3]:

$$\int e^{\binom{3}{x}} dx$$


In [4]: gamma(pi)
Out[4]:  $\Gamma(\pi)$ 

In [5]: □
```

# BCPy2000: Brain-Computer Interface systems

## Introspection of a multi-module real-time system

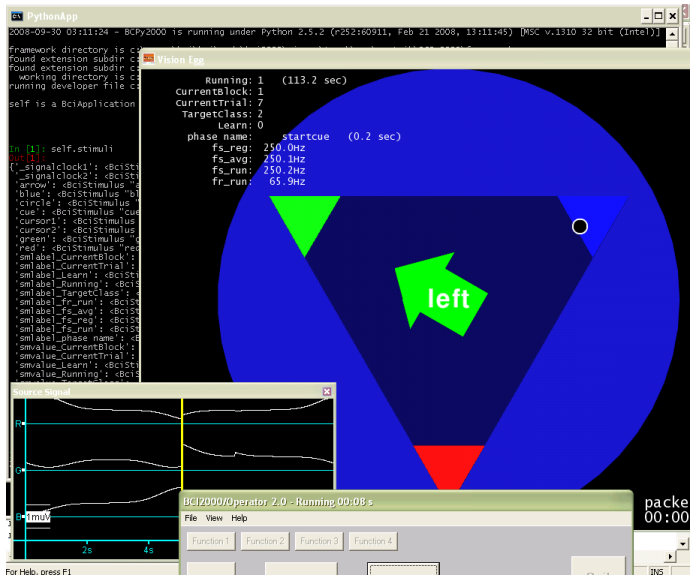


Figure credit: BCPy2000 team.



# NS-3 PyViz: discrete event simulation

## GTK widget inside the application

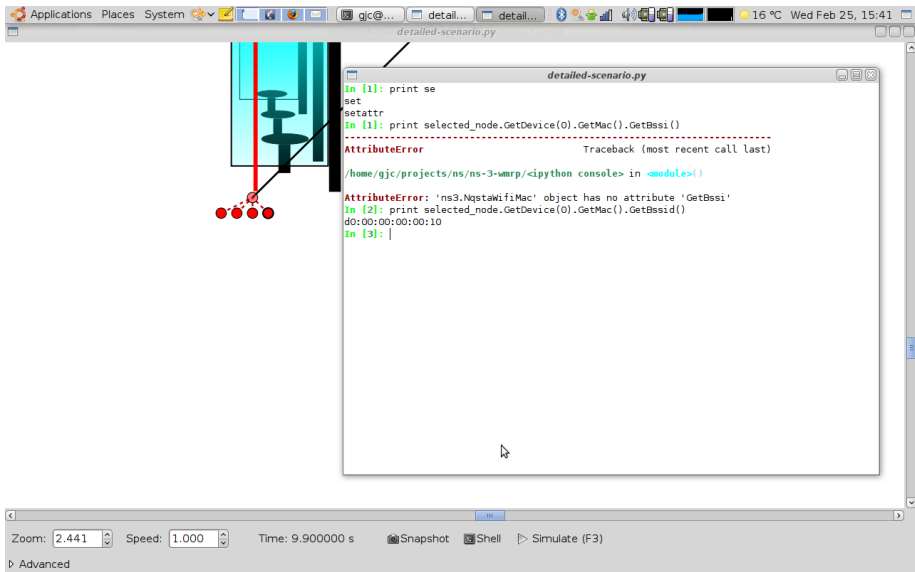
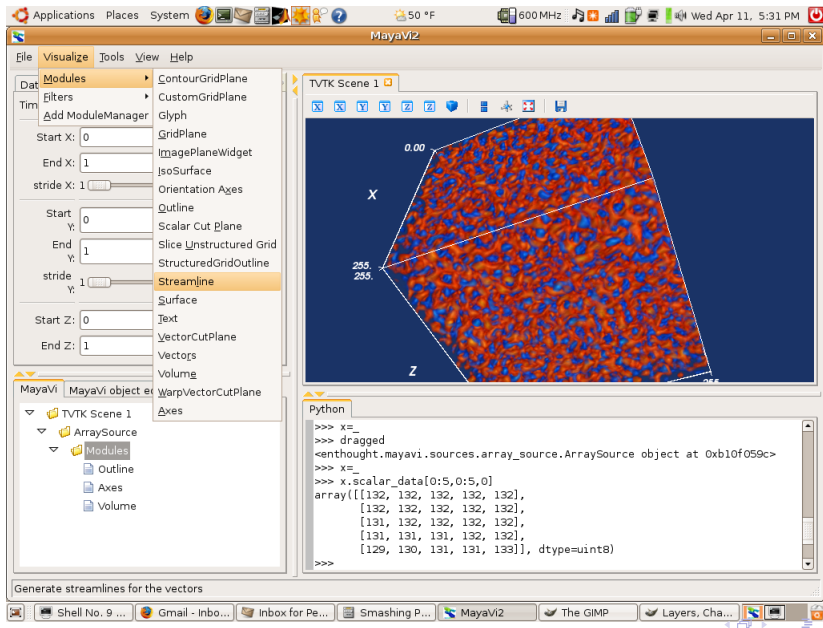


Figure credit: Gustavo Carneiro, NS-3 PyViz author.

# MayaVi 2: VTK for the rest of us

## WxWidgets embedded shell



# And the obligatory one: the iPhone



Figure credit: Josh Bloom, UC Berkeley Astronomy.

- [Sage](#): open source mathematics
  - Customized IPython instance (command line).
  - Input preprocessing to extend the language 'on the fly'
- [Django](#), [TurboGears](#), [Zope](#): web frameworks.
- [PyRAF](#) from the Hubble Telescope.
- [CASA](#) from NRAO.
- [Ganga](#) from CERN.
- [Amazon cloud](#): run your own IPython instances
  - as many as you want, without licensing restrictions...
- More...

# Wrapup

- Interactive, exploratory workflows are natural in scientific computing
- Python is a great language for this approach
- IPython has many tools to support it
- Use it in your projects...
  - let us know of problems
  - [join in](#) to make it better
  - **open source, by scientists, for scientists!**

---

IP[y]:

---

<http://ipython.scipy.org>

**Thanks!**  
**Any questions?**