Ten years of (interactive) scientific Python

Fernando Pérez
http://fperez.org
Fernando.Perez@berkeley.edu

Helen Wills Neuroscience Institute, UC Berkeley

EuroScipy 2011,
Ecole Normale Supérieure, Paris.
August 28, 2011
Supernova PTF11kyl:
Event of a Generation found on Tuesday
Most nearby Type Ia supernova in > 25 years
Soon visible with binoculars
Astronomy Challenge:
Quickly find new events in millions of (mostly bogus) candidates/night

Palomar Transient Factory (2009-now)

Solution:
Train a machine to mimic expert labels

- Python Google App Engine used for training
- scipy for expert voting
- Jython to run machine-learning on new data

→ 1000 to 1 compression allows PTF to focus on most probable candidates

Bloom et al. 2011
Outline

1. Personal view of scientific computing
2. IPython
3. Rethinking interactivity
4. The IPython Notebook
5. Parallelism?
6. Trouble in paradise
Outline

1. Personal view of scientific computing
2. IPython
3. Rethinking interactivity
4. The IPython Notebook
5. Parallelism?
6. Trouble in paradise
Science and computing: a changing landscape

- **High level** systems: beyond C and Fortran
- The data avalanche
- Massive resources at low cost
- Internet: enabler of Open Source Software
  - development model akin to scientific traditions
  - viable alternatives to proprietary software
Computing is not the ’third branch’ of science...

It is now the backbone of theory and experiment!
Hardware floating point

Extended precision floating point

Arbitrary precision integers

Rationals

Interval arithmetic

Symbolic manipulation

FORTRAN

Extended precision floating point

Text processing

Graphical user interfaces

Web interfaces

Hardware control

Multi-language integration

Data formats: HDF5, XML, ...

Databases
Me? No real training in computing

- **High school in Colombia, the 80’s**
  - TI-99/4A, 16KB Basic, home Sony tape recorder.
  - A few home tutoring lessons on ’structured programming’, promptly forgotten.
    - Never did anything interesting.

- **College in Colombia, EE:**
  - One semester of Pascal; my only formal computer course ever.
Switch to physics, plot fractals in TurboPascal

- Program on paper, use mom’s office PC on weekends.
- Debug on paper. **Think a lot away from the screen.**
VGA graphics!
Binary compiled in 1991/92, screenshot taken May 2011!

Fractales Lineales 1.2

Estamos graficando las siguientes
  Transformaciones Afines:

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>s</th>
<th>θ</th>
<th>φ</th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0.05</td>
<td>0.60</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>T2</td>
<td>0.05</td>
<td>-0.50</td>
<td>0.0</td>
<td>0.0</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>T3</td>
<td>0.60</td>
<td>0.50</td>
<td>40.3</td>
<td>40.1</td>
<td>0.00</td>
<td>0.60</td>
</tr>
<tr>
<td>T4</td>
<td>0.50</td>
<td>0.45</td>
<td>19.9</td>
<td>19.7</td>
<td>0.00</td>
<td>1.10</td>
</tr>
<tr>
<td>T5</td>
<td>0.50</td>
<td>0.53</td>
<td>-30.2</td>
<td>-31.9</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>T6</td>
<td>0.55</td>
<td>0.40</td>
<td>-39.8</td>
<td>-40.0</td>
<td>0.00</td>
<td>0.70</td>
</tr>
</tbody>
</table>

Pulse cualquier tecla para
detener la graficacion.

Hemos realizado 46697 Iteraciones

Pulse C para continuar,
I para imprimir el fractal,
o <Esc> para salir.
Clueless about the internet (which I unplugged)

Image credit: Carlos Latuff
A learning moment...

Image credit: crazytales562 on Flickr
High-level tools are important

- **Undergraduate thesis**: the *electrostatic unrestricted 3-body problem.*
  - Maple $\Rightarrow$ C $\Rightarrow$ Gnuplot.
  - Code generation as a natural part of the problem.
  - Multi-language integration.

- **Teach computational physics** for undergrads: C/Gnuplot on VAX talking to a 486PC running Linux.
  - A complete disaster.
  - Never again! Need different/better tools.

- **Thesis at CU Boulder**: *lattice QCD* (numerical Quantum Chromodynamics)
  - Large open source C package (MILC),
  - Custom C code, Mathematica, IDL, bash, sed, awk, Perl, Gnuplot. (*)

- End of my PhD: **Python**.
  - **Big revelation**: all of (*) $\Rightarrow$ One language!!!
Postdoc work: Adaptive, multiwavelet PDE tools
Gregory Beylkin, Vani Cheruvu, FP. Applied Math, CU Boulder.

- **Fast** application of integral kernels. (Partial Differential Equations)
- **Jump** from 1 to 3 dimensions: *extremely unusual*
- Complex algorithm: *beyond numerics.*
- **Performance**: to NumPy, F2PY, weave.
  - Dynamically generated C++ sources: *code as a run-time resource.*

$$N_{nod} = 10, \epsilon = 1.0 \times 10^{-10}, N_{blocks} = 445$$
Outline

1. Personal view of scientific computing
2. IPython
3. Rethinking interactivity
4. The IPython Notebook
5. Parallelism?
6. Trouble in paradise
Why IPython?

(something other than “I’d rather not finish my dissertation”)
Why IPython?

(something other than “I’d rather not finish my dissertation”)
I is for interactive...

In scientific computing, we typically don’t know what we’re doing.

Scientific computing ⇔ *Exploratory* computing
I is for interactive...

In scientific computing, we typically don’t know what we’re doing.

Scientific computing ⇔ Exploratory computing
I is for interactive...

In scientific computing, we typically don’t know what we’re doing.

Scientific computing ⇔ Exploratory computing
Interactive systems

Features

- **Execute/explore cycle instead of edit/compile/run**
- **Rich Libraries**
- **Plotting and data visualization**

Examples

- **Mathematica/Maple:** symbolic, now numerics...
- **IDL/Matlab:** numerics, image processing, ...
- **Unix system shell:** file management/text processing.
Interactive systems

Features

- **Execute/explore cycle instead of edit/compile/run**
- **Rich Libraries**
- **Plotting and data visualization**

Examples

- **Mathematica/Maple**: symbolic, now numerics...
- **IDL/Matlab**: numerics, image processing, ...
- **Unix system shell**: file management/text processing.
Python: an excellent *base* for an interactive scientific system

- **Dynamic** code evaluation
- No variable declarations
- **Powerful** introspection
- Very **regular** object model
- Excellent **string** processing
Python: an excellent *base* for an interactive scientific system

- **Dynamic** code evaluation
- No variable declarations
- Powerful **introspection**
- Very **regular** object model
- Excellent **string** processing
I said a base...
Mmh, introspection?

dreamweaver[~] $ python
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> ls
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ls' is not defined
>>> os?
  File "<stdin>", line 1
    os?
      ^
SyntaxError: invalid syntax
>>>
Basic comforts?

dreamweaver[~] > python
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> ls
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ls' is not defined
>>> os?
  File "<stdin>", line 1
  os?
  ^
SyntaxError: invalid syntax
>>> execfile('~/scratch/err.py')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: '~/scratch/err.py'
>>>
Useful error info

Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
[ GCC 4.4.5 ] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> ls
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ls' is not defined
>>> os?
  File "<stdin>", line 1
    os?
    ^
SyntaxError: invalid syntax
>>> execfile('~/scratch/err.py')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: '~/scratch/err.py'
>>> execfile('/home/fperez/scratch/err.py')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "~/home/fperez/scratch/err.py", line 9, in <module>
      foo33
NameError: name 'foo33' is not defined

>>>
We can do better...
My files, thankyouverymuch

```
dreamweaver[~] > ipython
Python 2.6.6 (r266:84292, Sep 15 2010, 16:22:56)
Type "copyright", "credits" or "license" for more information.

IPython 0.11.dev -- An enhanced Interactive Python.
?    -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]: ls ~/scratch/er*py
/home/fperez/scratch/err25.py   /home/fperez/scratch/error.py*
/home/fperez/scratch/err_comps.py  /home/fperez/scratch/err.py

In [2]: 
```
Some object details?

Type: module
Base Class: <type 'module'>
String Form: <module 'os' from '/usr/lib/python2.6/os.pyc'>
Namespace: Interactive
File: /usr/lib/python2.6/os.py

Docstring:

OS routines for Mac, NT, or Posix depending on what system we're on.

This exports:
- all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.
- os.path is one of the modules posixpath, or ntpath
- os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'
- os.getcwd is a string representing the current directory ('.' or ':')
- os.pardir is a string representing the parent directory ('..' or ':')
- os.sep is the (or a most common) pathname separator ('/' or ':' or '\\')
- os.extsep is the extension separator ('.' or '/')
- os.altsep is the alternate pathname separator (None or '/')
- os.pathsep is the component separator used in $PATH etc
- os.linesep is the line separator in text files ('' or '
' or '
')
- os.defpath is the default search path for executables
- os.devnull is the file path of the null device ('/dev/null', etc.)

Programs that import and use 'os' stand a better chance of being
Utilities needed to emulate Python's interactive interpreter.

# Inspired by similar code by Jeff Epler and Fredrik Lundh.

```
import sys
import traceback
from codeop import CommandCompiler, compile_command

__all__ = ["InteractiveInterpreter", "InteractiveConsole", "interact", "compile_command"]

def softspace(file, newvalue):
    oldvalue = 0
    try:
        oldvalue = file.softspace
    except AttributeError:
        pass
    try:
        file.softspace = newvalue
```

When things go wrong

```
In [13]: run ~/scratch/error
reps: 5

ValueError: shape mismatch: objects cannot be broadcast to a single shape
```
ipython --pylab

In [1]: import math, numpy
In [2]: from scipy.integrate import quad
In [3]: from scipy.special import j0
In [4]: def j0i(x):
   ...:     """Integral form of J_0(x)"""
   ...:     def integrand(phi):
   ...:         return math.cos(x*phi)*quad(integrand,0,math.pi)[0]
   ...:
In [5]: x = numpy.linspace(0,20,200) # sample grid: 200 points between 0 and 20
In [6]: y = j0(x) # sample J_0 at all values of x
In [7]: x1 = x[::10] # subsample the original grid every 10th point
In [8]: y1 = map(j0i,x1) # evaluate the integral form at all points in x1
In [9]: # Make a plot with these values (the ; suppresses output)
In [10]: plot(x,y,\label={"J_0(x)\});
In [11]: plot(x1,y1,'ro',\label={"\int_0^\infty \cos(x \sin \phi) d\phi\})
In [12]: axhline(0,\color={green},\label={nolegend})
In [13]: title(r'\textit{Verify J_0(x) = \frac{1}{2} \int_0^\infty \cos(x \sin \phi) d\phi}\}
In [14]: xlabel('x$');
In [15]: legend();
In [16]: matshow(numpy.random.random((32,32)))
Out[16]: <matplotlib.figure.Figure instance at 0x4630042c>
Embedded widget
How did we get here?
A brief history of IPython

October/November 2001: “just a little afternoon hack“

- My own $PYTHONSTARTUP hack:
  - ipython-0.0.1.py: 259 lines.
  - Mathematica’s In [N]: prompts and %N results cache.
- IPP (Interactive Python Prompt) by Janko Hauser (Oceanography)
- LazyPython by Nathan Gray (CS Caltech)

2002: Ignore John Hunter’s Gnuplot support patches

- ... let there be matplotlib
- and the world rejoiced.
- (actually finish my PhD!)
How did we get here?
A brief history of IPython

- **October/November 2001:** “just a little afternoon hack“
  - My own `$PYTHONSTARTUP` hack:
    - `ipython-0.0.1.py`: 259 lines.
    - Mathematica’s `In [N]:` prompts and `%N` results cache.
  - IPP (Interactive Python Prompt) by Janko Hauser (Oceanography)
  - LazyPython by Nathan Gray (CS Caltech)

- **2002: Ignore John Hunter’s Gnuplot support patches**
  - ... let there be `matplotlib`
  - and the world rejoiced.
  - (actually finish my PhD!)
2004: Brian Granger, Min Ragan-Kelley
Physics @ U. Santa Clara

- Killer team for parallel/networking ideas.
- Min’s senior undergrad thesis (Physics @ U. Santa Clara, CA):
  - first parallel tools, Twisted-based

2005: Ville Vainio

- Core maintenance while we worked on machinery for parallelism.

2008: Gaël Varoquaux/Enthought: ipythonx

- WX widget: standalone and embeddable (MayaVi)

2008: Laurent Dufréchou: ipython-wx

- WX app: standalone rich console
2004: Brian Granger, Min Ragan-Kelley
Physics @ U. Santa Clara
- Killer team for parallel/networking ideas.
- Min’s senior undergrad thesis (Physics @ U. Santa Clara, CA):
  - first parallel tools, Twisted-based

2005: Ville Vainio
- Core maintenance while we worked on machinery for parallelism.

2008: Gaël Varoquaux/Enthought: ipythonx
- WX widget: standalone and embeddable (MayaVi)

2008: Laurent Dufréchou: ipython-wx
- WX app: standalone rich console
2004: Brian Granger, Min Ragan-Kelley
Physics @ U. Santa Clara
  - Killer team for parallel/networking ideas.
  - Min’s senior undergrad thesis (Physics @ U. Santa Clara, CA):
    - first parallel tools, Twisted-based

2005: Ville Vainio
  - Core maintenance while we worked on machinery for parallelism.

2008: Gaël Varoquaux/Enthought: ipythonx
  - WX widget: standalone and embeddable (MayaVi)

2008: Laurent Dufréchou: ipython-wx
  - WX app: standalone rich console
2004: Brian Granger, Min Ragan-Kelley
Physics @ U. Santa Clara
  - Killer team for parallel/networking ideas.
  - Min’s senior undergrad thesis (Physics @ U. Santa Clara, CA):
    - first parallel tools, Twisted-based

2005: Ville Vainio
  - Core maintenance while we worked on machinery for parallelism.

2008: Gaël Varoquaux/Enthought: ipythonx
  - WX widget: standalone and embeddable (MayaVi)

2008: Laurent Dufréchou: ipython-wx
  - WX app: standalone rich console
Outline

1. Personal view of scientific computing
2. IPython
3. Rethinking interactivity
4. The IPython Notebook
5. Parallelism?
6. Trouble in paradise
More complex interactive uses?

- Kernel
- Client - Terminal
- Client - Qt
- Client - ...

Client: monitor email, publish, ...
A messaging protocol

Direct communication

- Execute code ('eval')
- Object information
- Complete
- History
- Connect

Broadcasting

- Functional execution:
  - Python inputs
  - Python outputs
  - Python errors

- Side effects:
  - Streams (stdout, stderr, etc)
  - Display data: plots, other payloads
A messaging protocol

Direct communication
- Execute code (‘eval’)
- Object information
- Complete
- History
- Connect

Broadcasting
- Functional execution:
  - Python inputs
  - Python outputs
  - Python errors
- Side effects:
  - Streams (stdout, stderr, etc)
  - Display data: plots, other payloads
ØMQ - “Sockets done right”
http://zeromq.org

The socket library that acts as a concurrency framework

- Pure C++ library.
- Python bindings in Cython (Brian Granger, Min RK). Python 2.5-3.2.
- Python bindings run messaging in native threads - no GIL
- Abstractions are at the message delivery level, not the raw-bytes level.
- Socket types encapsulate messaging patterns.
- Open source (LGPL), actively developed.
ØMQ: Messaging patterns

**Figure 1** - Request-Reply

- **Client**
  - REQ
  - "Hello"
  - "World"
  - REP
  - **Server**

**Figure 4** - Publish-Subscribe

- **Publisher**
  - PUB
  - bind
  - updates

- **Subscriber**
  - connect
  - updates

Image credit: official ØMQ documentation
A bit more history: 2010

- **February:** discover ØMQ (ZeroMQ)
  - A remarkable networking library in C++
  - *Sockets done right.*
  - 3 days later, Brian has Cython bindings going on github!

- **March:** proof of concept of shell over ØMQ
  - 2-day sprint with Brian
  - Simple, clean, performant. We’re thrilled.

- **May:** switch from Bazaar/Launchpad to Git/Github
  - Learn Git *NOW.*
  - Github rocks. Seriously.

- **Google Summer of Code:** 2 real prototypes on ØMQ
  - Omar Zapata (U. de Antioquia - Colombia, CS): terminal.
  - Gerardo Gutiérrez (U. de Antioquia - Colombia, Physics): Qt notebook.
A bit more history: 2010

- **February: discover ØMQ (ZeroMQ)**
  - A remarkable networking library in C++
  - **Sockets done right.**
  - 3 days later, Brian has Cython bindings going on github!

- **March: proof of concept of shell over ØMQ**
  - 2-day sprint with Brian
  - Simple, clean, performant. We’re thrilled.

- **May: switch from Bazaar/Launchpad to Git/Github**
  - Learn Git **NOW**.
  - Github rocks. Seriously.

- **Google Summer of Code: 2 real prototypes on ØMQ**
  - Omar Zapata (U. de Antioquia - Colombia, CS): terminal.
  - Gerardo Gutiérrez (U. de Antioquia - Colombia, Physics): Qt notebook.
A bit more history: 2010

- February: discover ØMQ (ZeroMQ)
  - A remarkable networking library in C++
  - Sockets done right.
  - 3 days later, Brian has Cython bindings going on github!

- March: proof of concept of shell over ØMQ
  - 2-day sprint with Brian
  - Simple, clean, performant. We’re thrilled.

- May: switch from Bazaar/Launchpad to Git/Github
  - Learn Git NOW.
  - Github rocks. Seriously.

- Google Summer of Code: 2 real prototypes on ØMQ
  - Omar Zapata (U. de Antioquia - Colombia, CS): terminal.
  - Gerardo Gutiérrez (U. de Antioquia - Colombia, Physics): Qt notebook.
A bit more history: 2010

- **February:** discover ØMQ (ZeroMQ)
  - A remarkable networking library in C++
  - Sockets done right.
  - 3 days later, Brian has Cython bindings going on github!

- **March:** proof of concept of shell over ØMQ
  - 2-day sprint with Brian
  - Simple, clean, performant. We’re thrilled.

- **May:** switch from Bazaar/Launchpad to Git/Github
  - Learn Git **NOW**.
  - Github rocks. Seriously.

- **Google Summer of Code:** 2 real prototypes on ØMQ
  - Omar Zapata (U. de Antioquia - Colombia, CS): terminal.
  - Gerardo Gutiérrez (U. de Antioquia - Colombia, Physics): Qt notebook.
Rich Qt Console
  - Evan Patterson (Caltech physics)

Full kernel over ØMQ
  - Brian Granger (now at Cal Poly San Luis Obispo physics) and FP.

Implement parallel machinery over ØMQ
  - Min Ragan-Kelley (now at UC Berkeley, plasma physics)
  - (btw, this is insane – though par for the course for Min)

HTML/JavaScript frontend: the web notebook was coming
  - James Gao (UC Berkeley neuroscience)

Python 3 port of core terminal code
  - Thomas Kluyver
Release 0.11
- Qt console
- ZeroMQ parallel code
- Removed all Twisted code
- The start of the path to IPython 1.0

Develop the web notebook
- Summer: Brian codes like mad
- Built on the same architecture

Full Python 3 support
An architecture for interactivity with ØMQ
Client: a rich Qt Console
Enthought: sponsorship, Evan Patterson, Robert Kern.

Feels like a console, runs like a GUI
- Inline and floating images
- Syntax highlighting, full multiline editing
- Session saving
  - HTML (with PNG or SVG)
  - PDF/printing
- Help viewer
- %magics, !system access, IPython...
- Detach/reattach support
Simple spectral analysis

An illustration of the Discrete Fourier Transform

\[ X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \ldots, N - 1 \]

using windowing, to reveal the frequency content of a sound signal.
We begin by loading a datafile using SciPy's audio file support:

```python
In [1]: from scipy.io import wavfile
rate, x = wavfile.read('/home/fperez/teach/py4science/book/examples/test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's built-in specgram routine:

```python
In [3]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```

---

Client: a web notebook
Brian Granger, James Gao.
### Other projects using IPython

#### Scientific
- **Sage**: open source mathematics.
- **PyRAF**: Space Telescope Science Institute
- **CASA**: Nat. Radio Astronomy Observatory
- **Ganga**: CERN
- **PyMAD**: neutron spectrom., Laue Langevin
- **Sardana**: European Synchrotron Radiation
- **ASCEND**: eng. modeling (Carnegie Mellon).
- **JModelica**: dynamical systems.
- **DASH**: Denver Aerosol Sources and Health.
- **PyIMSL** Studio, by Visual Numerics.
- **Trilinos**: Sandia National Lab.
- **Pymerase**: microarray gene expression.
- **DoD**: baseline configuration.
- **Mayavi**: 3d visualization, Enthought.
- **NiPype**: computational pipelines, MIT.
- **QSnake**: source-distro for scientific comp.

#### Web/Other
- **Visual Studio 2010**: MS.
- **Django** web.
- **Turbo Gears** web.
- **Pylons** web framework
- **Zope** and **Plone** CMS.
- **Axon Shell**, BBC **Kamaelia**.
- **Schevo** database.
- **Pitz**: distributed task/bug tracking.
- **iVR** (interactive Virtual Reality).
- **Movable Python** (portable Python environment).
- ...
(Incomplete) Cast of Characters

- **Brian Granger** - Physics, Cal State San Luis Obispo
- **Min Ragan-Kelley** - UC Berkeley
- **Thomas Kluyver** - U. Sheffield
- **Evan Patterson** - Caltech/Enthought
- **Robert Kern** - Enthought
- **Jörgen Stenarson** - SP Technical Research Institute of Sweden.
- Ondrej Certik - Physics, U Nevada Reno
- Darren Dale - Cornell
- Laurent Dufréchou - France
- James Gao - UC Berkeley
- **Satra Ghosh** - MIT Neuroscience
- Paul Ivanov - UC Berkeley
- Justin Riley - MIT
- Thomas Spura - Fedora project
- Ville Vainio - CS, Tampere University of Technology, Finland
- **Stefan van der Walt** - Applied Math, U. Stellenbosch, South Africa
- **Gaël Varoquaux** - Neurospin (Orsay, France)
- Mark Voorhies - UC San Francisco
- Many more! (~60 commit authors)
Support

Enthought, Austin, TX: Lots!

Tech-X Corporation, Boulder, CO: Parallel/notebook (previous versions)

Microsoft: WinHPC support, Visual Studio and Azure integration

NIH: via NiPy grant

NSF: via Sage compmath grant

Hands-on
Outline

1. Personal view of scientific computing
2. IPython
3. Rethinking interactivity
4. The IPython Notebook
5. Parallelism?
6. Trouble in paradise
We’re very excited

- Capturing exploratory workflows (for self or others)
  - Reproducibility in research
- Interactive documents
  - Mathematica Manipulate, Sage @interact.
- Teaching
  - Tutorials and documentation
- Publication: Papers and books
- A rich document format and API
- Very long list of ideas...

We really want community buy-in, hence critical feedback!
We’re very excited

- Capturing **exploratory workflows** (for self or others)
  - Reproducibility in research
- **Interactive documents**
  - Mathematica **Manipulate**, Sage **@interact**.
- **Teaching**
  - Tutorials and documentation
- **Publication**: Papers and books
- A rich document **format and API**
- **Very long list of ideas...**

We really want community buy-in, hence critical feedback!
Outline

1. Personal view of scientific computing
2. IPython
3. Rethinking interactivity
4. The IPython Notebook
5. Parallelism?
6. Trouble in paradise
The infamous Global Interpreter Lock in CPython

Only one thread can modify Python state/variables at a time

- Historical reasons, simplicity of implementation
- All attempts at removing it have failed
- Threads useless for CPU-bound problems.

The best possible reference on the GIL: David Beazley's work

http://www.dabeaz.com/GIL
The infamous Global Interpreter Lock in CPython

Only one thread can modify Python state/variables at a time

- Historical reasons, simplicity of implementation
- All attempts at removing it have failed
- Threads useless for CPU-bound problems.

The best possible reference on the GIL: David Beazley’s work

http://www.dabeaz.com/GIL
Core idea: manage a namespace

- Read: take user input.
- Eval: execute code.
- Print: provide output.
- Add support for data transfer...

...and interactive and parallel work start looking very similar.

These steps can happen in multiple processes:

- Read: user environment
- Eval: kernel process
- Print: user environment
Core idea: manage a namespace

- Read: take user input.
- Eval: execute code.
- Print: provide output.
- Add support for data transfer...

...and interactive and parallel work start looking very similar.

These steps can happen in multiple processes:

- Read: user environment
- Eval: kernel process
- Print: user environment
IPython: a REPL (Read/Eval/Print Loop)

Core idea: manage a namespace

- **Read**: take user input.
- **Eval**: execute code.
- **Print**: provide output.
- **Add support for data transfer...**

...and interactive and parallel work start looking very similar.

These steps can happen in multiple processes:

- **Read**: user environment
- **Eval**: kernel process
- **Print**: user environment
IPython for parallel computing
Min Ragan-Kelley, Brian Granger
A few simple concepts

- The **client**: lightweight handle on all engines of a cluster
- The **views**: “slice” the client with specific execution semantics
  - **DirectView**: direct execution on *all* engines (blocking or not)
  - **LoadBalancedView**: run on *any one* engine.
- **x.apply()**: highly functional API.
- **AsyncResult**: similar to the one in *multiprocessing*.
- The **controller**: manage all activity in a cluster (accessed via *clients*)
Multiple usage patterns

- **Direct** interface: explicit (and flexible) control of where things run.
  - Choice of blocking behavior up to the user.
- **Task** interface: load-balanced (with flexible scheduling policies)
- **Data** push/pull, scatter/gather.
- **Decorators** that encapsulate many common patterns
- **Informative** exception propagation
- **Explicit** node-to-node communication:
  - MPI-style problems
  - … without all the pain of MPI.
Hands-on
Neat trick: DAG dependencies

A simple DAG example

In [2]: G = random_dag(32, 128)
In [3]: jobs = {}

# in reality, each job would presumably be different
# randomwait is just a function that sleeps for a random interval
In [4]: for node in G:
   ...:     jobs[node] = randomwait

In [5]: c = client.Client()

In [6]: results = {}

In [7]: for node in G.topological_sort():
   ...:     # get list of AsyncResult objects from nodes
   ...:     # leading into this one as dependencies
   ...:     deps = [ results[n] for n in G.predecessors(node) ]
   ...:     # submit and store AsyncResult object
   ...:     results[node] = client.apply(jobs[node], after=deps, block=False)

In [8]: [ r.get() for r in results.values() ]
DAG dependencies - validation
Outline

1. Personal view of scientific computing
2. IPython
3. Rethinking interactivity
4. The IPython Notebook
5. Parallelism?
6. Trouble in paradise
Commit rates since Jan-2010

- cython
- ipython
- matplotlib
- mayavi
- numpy
- scipy
- sympy
We have problems

Python in science is a great success story, but...

The entire edifice rests on (often the spare time of) ~30 people!

- A backbone of top-heavy projects
- Formal funding hard to come by
  - Glimmers of hope, e.g. Ed Seidel @ NSF.
- Software work in science can be career suicide.

Not a sustainable strategy for the long term
We have problems

Python in science is a great success story, but...

The entire edifice rests on (often the spare time of) ~30 people!

- A backbone of top-heavy projects
- Formal funding hard to come by
  - Glimmers of hope, e.g. Ed Seidel @ NSF.
- Software work in science can be career suicide.

Not a sustainable strategy for the long term
We have problems

Python in science is a great success story, but...

The entire edifice rests on (often the spare time of) ~30 people!

- A backbone of top-heavy projects
- Formal funding hard to come by
  - Glimmers of hope, e.g. Ed Seidel @ NSF.
- Software work in science can be career suicide.

Not a sustainable strategy for the long term
How can you help?

- **Teach a class?**
  - Real world homework projects!

- **Tenure/grant panel?**
  - Don’t kill candidates who contribute software
  - Do kill proposals who pay lip service to real development work

- **Have your own project?**
  - Put it up on Github, BSD/MIT license it

- **Just join in!**
  - Work on whatever project interests you.
Plenty of interesting problems

- **IPython**
  - Reproducible research, interactive documents, parallel APIs, MPI for the rest of us, from notebook to paper, ...

- **Numpy**
  - multidimensional data descriptions, array VMs, database interactions, jagged arrays, streaming access, ...

- **Matplotlib**
  - grammar of graphics, streaming data, Javascript interactivity, beyond the Matlab API, ...

- **Scipy**
  - Sparse tensors, adaptive functional interpolators, statistics (with sk.learn, statsmodels), code generation models, ...

- **Python itself**
  - new operators (PEP 225), decimals and rationals, docstrings, the packaging hell, ...
The incentive structure of academia fits poorly the widely distributed, rapid-fire collaborative dynamics of open source development.
Aside - BSD/MIT licenses?

- The **GPL**: a good tool for applications (Linux kernel)
- Bad for libraries:
  - Infinite leverage
- A spirit of **reciprocity**
  - You are building on millions of lines of BSD-type code
  - GPL code is worse than closed source as a contribution to those foundations
- We have been **well served** by permissive licensing
  - 10 years of productive engagement with industry
- Nobody is going to get rich by 'stealing' your code
  - Building a successful business takes MUCH more than putting a price sticker on a piece of code.
In closing

- The **language** lured me in...
- But I stayed for the **community**!
  - Real **friendships** and incredible people
  - A culture of generous and mutual cross-project **collaboration**
  - But we have a **ton of work** to do!
- The tools we need **must be built by scientists**.

  **Lots of space for truly innovative thinking, and Python is an expressive tool for the exercise.**
Thank you!

Questions?