IPython: a very quick overview

Fernando Pérez, Brian Granger, Min Ragan-Kelley
and 80+ contributors...

UC Berkeley, Cal Poly San Luis Obispo

Oct 15, 2010
Getting all the power from interactive computing in Python

1. A better Python shell: object introspection, system access, 'magic' commands, ...
2. An embeddable interpreter: debugging, mix batch/interactive work.
3. A flexible component: base environment for systems with Python as the underlying language
4. A system for interactive control of distributed/parallel computing systems.
Some quick stats. http://www/ohloh.net/p/ipython

Ohloh Analysis Summary

- Mostly written in Python
- Mature, well-established codebase
- Increasing year-over-year development activity
- Large, active development team
- Well-commented source code

Updated 13 Oct 2010 07:10 UTC

Project Cost

This calculator estimates how much it would cost to hire a team to write this project from scratch. More »

<table>
<thead>
<tr>
<th>Include</th>
<th>Markup And Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codebase</td>
<td>77,003</td>
</tr>
<tr>
<td>Effort (est.)</td>
<td>19 Person Years</td>
</tr>
<tr>
<td>Avg. Salary</td>
<td>$ 55000</td>
</tr>
<tr>
<td></td>
<td>year</td>
</tr>
<tr>
<td></td>
<td>$ 1,041,279</td>
</tr>
</tbody>
</table>

Ratings & Reviews

Community Rating

Your Rating: ★★★★★
Click to rate this project.

Based on 68 user ratings.
**Licenses**

Ohloh searches the source code for individual license declarations. These licenses can differ from the project's official license.

- **BSD Copyright**: 257 files
- **MIT License**: 5 files
- **GNU General Public License 2**: 2 files
- **GNU Lesser General Public License v3**: 1 file
- **Apache License 2.0**: 1 file
- **Python 2.4.2 license**: 1 file

---

**Languages**

Ohloh analyzes the project source code and determines the language of each line of code, excluding comments and blanks.

- **Python**: 92%
- **XML**: 5%
- **Other**: 3%

---

**Lines of Code**

[Line of code chart with years from 2000 to 2010]
# Lines of Code By Language

<table>
<thead>
<tr>
<th>Language</th>
<th>Code Lines</th>
<th>Comment Lines</th>
<th>Comment Ratio</th>
<th>Blank Lines</th>
<th>Total Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>71,224</td>
<td>43,335</td>
<td>37.8%</td>
<td>27,576</td>
<td>142,135</td>
</tr>
<tr>
<td>XML</td>
<td>3,931</td>
<td>0</td>
<td>0.0%</td>
<td>115</td>
<td>4,046</td>
</tr>
<tr>
<td>Emacs Lisp</td>
<td>522</td>
<td>364</td>
<td>41.1%</td>
<td>89</td>
<td>975</td>
</tr>
<tr>
<td>CSS</td>
<td>498</td>
<td>11</td>
<td>2.2%</td>
<td>91</td>
<td>600</td>
</tr>
<tr>
<td>Perl</td>
<td>401</td>
<td>440</td>
<td>52.3%</td>
<td>208</td>
<td>1,049</td>
</tr>
<tr>
<td>Make</td>
<td>252</td>
<td>49</td>
<td>16.3%</td>
<td>107</td>
<td>408</td>
</tr>
<tr>
<td>shell script</td>
<td>143</td>
<td>106</td>
<td>42.6%</td>
<td>63</td>
<td>312</td>
</tr>
<tr>
<td>HTML</td>
<td>130</td>
<td>0</td>
<td>0.0%</td>
<td>65</td>
<td>195</td>
</tr>
<tr>
<td>Vim Script</td>
<td>124</td>
<td>2</td>
<td>1.6%</td>
<td>17</td>
<td>143</td>
</tr>
<tr>
<td>Objective-C</td>
<td>30</td>
<td>7</td>
<td>18.9%</td>
<td>12</td>
<td>49</td>
</tr>
</tbody>
</table>
Actively developed: Git/GitHub are amazing
## Projects using IPython

### Scientific

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PyRAF</strong></td>
<td>Space Telescope Science Institute</td>
</tr>
<tr>
<td><strong>CASA</strong></td>
<td>National Radio Astronomy Observatory</td>
</tr>
<tr>
<td><strong>Ganga</strong></td>
<td>CERN</td>
</tr>
<tr>
<td><strong>PyMAD</strong></td>
<td>neutron spectrometer, Institut Laue Langevin</td>
</tr>
<tr>
<td><strong>Sardana</strong></td>
<td>European Synchrotron Radiation Facility</td>
</tr>
<tr>
<td><strong>ASCEND</strong></td>
<td>engineering modeling (Carnegie Mellon)</td>
</tr>
<tr>
<td><strong>JModelica</strong></td>
<td>dynamical systems</td>
</tr>
<tr>
<td>Denver Aerosol Sources and Health (<strong>DASH</strong>), CU Boulder</td>
<td></td>
</tr>
<tr>
<td><strong>PyIMSL</strong> Studio, by Visual Numerics</td>
<td></td>
</tr>
<tr>
<td><strong>Trilinos</strong></td>
<td>Sandia National Lab</td>
</tr>
<tr>
<td><strong>Sage</strong></td>
<td>open source mathematics</td>
</tr>
<tr>
<td><strong>Pymerase</strong></td>
<td>microarray gene expression</td>
</tr>
</tbody>
</table>

### Web/Other

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Django</strong></td>
<td>web framework</td>
</tr>
<tr>
<td><strong>Turbo Gears</strong></td>
<td>web framework</td>
</tr>
<tr>
<td><strong>Pylons</strong></td>
<td>web framework</td>
</tr>
<tr>
<td><strong>Zope</strong> and <strong>Plone</strong> CMS</td>
<td></td>
</tr>
<tr>
<td>Axon Shell, BBC <strong>Kamaelia</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Schevo</strong> database</td>
<td></td>
</tr>
<tr>
<td><strong>Pitz</strong></td>
<td>distributed task/bug tracking</td>
</tr>
<tr>
<td><strong>iVR</strong> (interactive Virtual Reality)</td>
<td></td>
</tr>
<tr>
<td><strong>Movable Python</strong> (portable Python environment)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Matlab/IDL-like interactive use

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*math.sin(phi))
   ...:     return 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 = j0i(x) # sample J0 at all values of x
In [7]: x1 = x[::10] # subsample the original grid every 10th point
In [8]: y1 = numpy.array(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=r'$J_0(x)$');
In [11]: plot(x1,y1,ro',label=r'$J_0$');
In [12]: axhline(0,color='green',label='_nolegend_');
In [13]: title(r'Verify $J_0(x)=\frac{1}{\pi} \int_0^{\pi} \cos(x \sin \phi) d\phi$');
In [14]: xlabel('$x$');
In [15]: legend();
In [16]: imshow(numpy.random.random((32,32)))
out[16]: <matplotlib.figure.Figure instance at 0x4630042c>
Parallel work, the story so far...

- Parallel computing: fully interactive
  - development, debugging, testing, execution, monitoring,...
- Easy things should be easy, difficult things possible
- Make parallel computing collaborative
- More dynamic model for load balancing and fault tolerance
- Seamless integration with other tools: plotting/visualization, system shell.
- Also want to keep the benefits of traditional approaches:
  - Should integrate with threads/MPI if appropriate
  - Should be easy to integrate compiled code and libraries
- Support many types of parallelism
Network-aware IPython

The "IPython VM"

- Kernel Controller
  - Non-blocking
  - World-visible

Private link

Kernel Engine

MAY BLOCK!

Twisted

Networks
What does IPython offer here?

- **Easy reuse** and distribution of existing serial (‘normal’) codes.
- High-level abstractions for ‘embarrassingly parallel’ problems.
  - Direct execution of code over the network: *multiengine* interface.
  - Out-of-the box *task farming* tools: *task* interface.
- Task farming system is “low-latency” (not in the Myrinet sense...)
  - can be integrated into more complex codes.
- Implement any approach to parallelism you want:
  - Synchronous or asynchronous execution of code on nodes.
  - Task farming.
  - Traditional Message Passing (MPI).
  - Integrate hybrid codes.
Some technical notes

- Networking: Twisted
  - High-level interfaces: no need to learn Twisted.
- RPC: Twisted’s foolscap
- Security: foolscap supports SSL (pyOpenSSL) and a capabilities model.
  - Review/improvements welcome, we’re not security experts!
- MPI support is there, use mpi4py bindings.
- Integration with queuing systems, better process control coming...
“Sockets done right”

- C/C++ library
- Python bindings in Cython (Brian Granger, Min RK)
- 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
Interactive IPython on ØMQ

- Kernel raw_input
- Requests to kernel
- Kernel output broadcast
- Request/Reply direction
Rich Qt Console
Enthought: sponsorship, Evan Patterson.

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
Multiple users of one process instead of many processes for one user

These could be two different hosts on separate networks
ZeroMQ-based architecture: very solid design, polish work to do...

Release 0.11: December 2010

- 0.10.2 was out on Tuesday Oct 12.

Clients:

- Continue improving Qt console: lots and lots of ideas.
- Rich Qt “notebook” client: prototype exists (Google SoC 2010).
- Collaborative web client: prototype already started at UC Berkeley (two days ago). AJAX/HTML5.

Parallel architecture on ZeroMQ (later today - see Min’s talk)

Python3 support: experimental branch already on GitHub.

New ideas from today???
Where to next?

- **ZeroMQ-based architecture**: very solid design, polish work to do...
- **Release 0.11**: December 2010
  - 0.10.2 was out on Tuesday Oct 12.
- **Clients**:
  - Continue improving Qt console: lots and lots of ideas.
  - Rich Qt “notebook” client: prototype exists (Google SoC 2010).
  - Collaborative web client: prototype already started at UC Berkeley (two days ago). AJAX/HTML5.
- **Parallel architecture on ZeroMQ** (later today - see Min’s talk)
- **Python3 support**: experimental branch already on GitHub.

New ideas from today???