

# IPython

Getting the most out of working interactively in Python

<http://ipython.scipy.org>

**Brian E. Granger**  
**Tech-X**

bgranger@txcorp.com

**Fernando Pérez**

**Applied Math, U. Colorado**

fperez@colorado.edu



**CU BOULDER**

PyCon'07

Addison, Texas; Feb. 24 2007

# Acknowledgements

---

- **Nathan Gray and Janko Hauser**: authors of the code IPython started from.
- **IPython community**: too many people to list, but a few must be mentioned
  - Ville Vainio (trunk)
  - Jörgen Stenarson (pyreadline for win32)
  - Walter Dörwald (ipipe system)
  - Benjamin Ragan-Kelley (distributed/parallel support, see next talk)
- **Enthought (Eric Jones)**: hosting IPython, the SciPy package, ...
- **\$\$\$**:
  - Tech-X Corporation (B. Granger)
  - US Department of Energy and Oak Ridge National Laboratory (F. Pérez)

# IPython's goals

---

The **interactive prompt**: one of Python's greatest strengths.

**But**: it feels like a half-implemented idea (vs. the Unix shell, or Mathematica's prompt)

In its simplest form, IPython is a BSD-licensed Python shell replacement.

In broader terms, it tries to be:

1. **A better Python shell**: object introspection, system access, 'magic' command system for adding functionality when working interactively, ...
2. **An embeddable interpreter**: useful for debugging and for mixing batch-processing with interactive work.
3. **A flexible component**: you can use it as the base environment for other systems with Python as the underlying language. It is very configurable in this direction.
4. **A system for interactive control of distributed/parallel computing systems**: next talk.
5. **An interactive component** we can plug into GUIs, browser-based shells, etc.

## Where we are today

---

- Enthought hosts IPython (<http://ipython.scipy.org>), with proper SVN, mailing lists, Trac and Moin sites, etc.
- A crude LOC count (IPython currently has zero extension code):

```
# In trunk:
find | grep 'py$' | egrep -v '/build/|\\.svn|external' | xargs wc -l
28761 total

# In the saw branch (current development, see the next talk):
find | grep 'py$' | egrep -v '/build/|\\.svn|external' | xargs wc -l
15150 total
```

- Packaged by all the major Linux distros and Fink. We ship Win32 installer. egg OK.
- Widely used, stable. A number of projects offer it as a shell, sometimes with extensive customizations: SAGE, Django, TurboGears, PyRAF from the Hubble Telescope, CASA from NRAO, Ganga from CERN, ...
- Works with: Python 2.3, 2.4 or 2.5.

# Design ideas

---

*A good shell is a necessity for a solid, pleasant (scientific) computing environment*

Some key ideas underlying IPython's design:

- **Every keystroke counts:** it's an interactive shell, after all.
- **Meta-control:** the 'magic' functions control IPython itself while it runs.
- **System-level access:** direct access to files, commands, etc.
- **Pleasant development:**
  - Object introspection: TAB-completion, '?', '??', '%p\*' functions.
  - Better tracebacks: colored, longer and with data details.
  - %run: `execfile()` on steroids.
  - Profiler: quick and easy profile access via %prun.
  - Debugger: automatic pdb triggering on exceptions.
- **Adaptability:** be easily extensible and customizable for specific problem domains.

# Embedding IPython into other programs

---

You can call IPython as a Python shell inside your own programs.

The resulting shell opens within the surrounding local/global namespaces.

Great for:

- Debugging: print variables, execute code, plot things right at the trouble spot.
- Providing interactive abilities for your programs (very useful for data analysis).

It's as simple as:

```
from IPython.Shell import IPShellEmbed
ipshell = IPShellEmbed()
... Your code here ...
ipshell() ← Opens IPython in your program at this point
... More code ...
ipshell() ← It can be called multiple times
```

# An extensible system

---

- Plain Python customisation is clunky: `$PYTHONSTARTUP`.
- IPython has extensive customization options in `~/.ipython/ipythonrc`
- Configuration 'profiles':

```
$ ipython -p scipy ← Load ipythonrc-scipy config
```

These configuration files can include others: a base config for most options, plus specific settings for particular uses:

`ipythonrc`  $\subset$  `ipythonrc-math`  $\subset$  `ipythonrc-scipy`  
(base config)            (calculator)            (full SciPy)

- Extensible input syntax. You can define filters that preprocess user input before execution (try `ipython -p tutorial`). Very useful to make tools tailored for special application domains.
- Other parts are also customizable (magics, prompts, object info, ...)

# Basic interactive features

---

- TAB-completion in the local namespace and filesystem (via readline). Extensible.
- Numbered prompts with command history, searching and caching:
  - **Output:** stored in the global `Out` and `_N` (`Out[4]==_4`). For convenience, `_`, `--` and `---` keep the last three results.
  - **Input:** stored in the global `In`. Re-execute code with `'exec In[22:29]+In[34]'`.
  - `%macro`: `'%macro mm 22:29 24'` → type `'mm'` to execute.
  - `%hist` shows previous input history.
  - `Ctrl-p/n` or `↑` and `↓`: search previous/next match in history.
- Automatic indentation of typed text (toggle with `%autoindent`).
- `%edit`: direct access to your `$EDITOR`. A poor-man's multi-line editing capabilities, without the complexity (for developers) of a curses interface. IPython can also be used as the Python shell in (X)Emacs.
- Session logging and restoring (`%logstart`, `%logon/off`, `%runlog`).

# Interactive Demos

(which will only cover a few features, there's plenty more, see the docs!)

# Random goodies we won't have time to talk about

---

These tools may be useful to your projects and everyday use of Python:

- ✓ [demo](#): The demos presented here use the `IPython.demo` module for interactively presenting Python scripts to an audience.
- ✓ [irunner](#): pexpect wrapper to run a file containing input for any interactive system with a recognizable (by a regexp) prompt.
- ✓ [pycolor](#): read your source nicely highlighted at the terminal.
- ✓ [ipdoctest](#): easily generate doctest files (for non-docstring uses), merge them into a standard unittest suite, use IPython syntax in them. In the saw development branch.
- ✓ [ultraTB/CrashHandler](#): nicer tracebacks for your code, enormously detailed post-mortem reports (many bugs caused by user code fixed just with these tracebacks).

**Thanks! Any Questions?**