
Spyder Documentation

Release 3

Pierre Raybaut

Aug 31, 2017

Contents

1 Overview	3
2 Installation	5
2.1 Installing on Windows Vista/7/8/10	5
2.2 Installing on MacOS X	6
2.3 Installing on Linux	7
2.4 Installing or running directly from source	7
2.5 Installing the development version	9
2.6 Help and support	9
3 Command line options	11
4 Editor	13
4.1 How to define a code cell	15
5 IPython Console	17
5.1 Reloading modules: the User Module Reloader (UMR)	19
6 Debugging	21
6.1 Debugging with pdb	21
7 Console	23
8 Variable Explorer	25
8.1 Supported types	28
9 Help	31
10 Projects	35
10.1 Version Control Integration	37
11 Static code analysis	39
12 File Explorer	41
13 History log	45
14 Find in files	47

15 Online help	49
16 Internal Console	51

Spyder is the Scientific PYthon Development EnviRonment:

- a powerful interactive development environment for the Python language with advanced editing, interactive testing, debugging and introspection features
- and a numerical computing environment thanks to the support of *IPython* (enhanced interactive Python interpreter) and popular Python libraries such as *NumPy* (linear algebra), *SciPy* (signal and image processing) or *matplotlib* (interactive 2D/3D plotting).

Spyder may also be used as a library providing powerful console-related widgets for your PyQt-based applications – for example, it may be used to integrate a debugging console directly in the layout of your graphical user interface.

Spyder websites:

- Downloads, bug reports and feature requests: <https://github.com/spyder-ide/spyder>
- Discussions: <http://groups.google.com/group/spyderlib>

Contents:

Spyder is a Python development environment with the following key features:

Key features:

- general features:
 - MATLAB-like PYTHONPATH management dialog box (works with all consoles)
 - Windows only: current user environment variables editor
 - direct links to documentation (Python, Matplotlib, !NumPy, !Scipy, etc.)
 - direct link to Python(x,y) launcher
 - direct links to !QtDesigner, !QtLinguist and !QtAssistant (Qt documentation)
- *preferences* dialog box:
 - keyboard shortcuts
 - syntax coloring schemes (source editor, history log, help)
 - console: background color (black/white), automatic code completion, etc.
 - and a lot more...
- *Editor*:
 - syntax coloring (Python, C/C++, Fortran)
 - *breakpoints* and *conditional breakpoints* (debugger: *pdb*)
 - run or debug Python scripts (see console features)
 - *run configuration* dialog box:
 - * working directory
 - * command line options
 - * run in a new Python interpreter or in an existing Python interpreter or IPython client
 - * Python interpreter command line options

- *code outline explorer*: functions, classes, if/else/try/... statements
- *powerful code introspection features* (powered by *rope*):
 - * *code completion*
 - * *calltips*
 - * *go-to-definition*: go to object (any symbol: function, class, attribute, etc.) definition by pressing Ctrl+Left mouse click on word or Ctrl+G (default shortcut)
- *occurrence highlighting*
- typing helpers (optional):
 - * automatically insert closing parentheses, braces and brackets
 - * automatically unindent after ‘else’, ‘elif’, ‘finally’, etc.
- *to-do* lists (TODO, FIXME, XXX)
- errors/warnings (real-time *code analysis* provided by *pyflakes*)
- integrated static code analysis (using *pylint*)
- direct link to *winpdb* external debugger
- *Console*:
 - all consoles are executed in a separate process
 - *code completion/calltips* and automatic link to help (see below)
 - open Python interpreters or basic terminal command windows
 - run Python scripts (see source editor features)
 - *variable explorer*:
 - * *GUI-based editors* for a lot of data types (numbers, strings, lists, arrays, dictionaries, ...)
 - * *import/export data* from/to a lot of file types (text files, !NumPy files, MATLAB files)
 - * multiple array/list/dict editor instances at once, thus allowing to compare variable contents
 - * data visualization
- *History log*
- *Help*:
 - provide documentation or source code on any Python object (class, function, module, ...)
 - documentation may be displayed as an html page thanks to the rich text mode (powered by *sphinx*)
- *Online help*: automatically generated html documentation on installed Python modules
- *Find in files*: find string occurrences in a directory, a mercurial repository or directly in PYTHONPATH (support for regular expressions and included/excluded string lists)
- *File Explorer*
- *Projects*

Spyder may also be used as a PyQt5 or PyQt4 extension library (module ‘spyder’). For example, the Python interactive shell widget used in Spyder may be embedded in your own PyQt5 or PyQt4 application.

Spyder is quite easy to install on Windows, Linux and MacOS X. Just the read the following instructions with care.

Installing on Windows Vista/7/8/10

The easy way

Spyder is already included in these *Python Scientific Distributions*:

1. Anaconda
2. WinPython
3. Python(x,y)

You can start using it immediately after installing one of them (you only need to install one!).

The hard way

If you want to install Spyder directly, you need to follow these steps:

1. Install the essential requirements:
 - The Python programming language
 - PyQt5 (recommended) or PyQt4
2. Install Spyder and its dependencies by running this command:

```
pip install spyder
```

Updating Spyder

You can update Spyder by:

- Updating Anaconda, WinPython or Python(x,y).
- Or using this command (in case you *don't* use any of those scientific distributions):

```
pip install --upgrade spyder
```

Note: This command will also update all Spyder dependencies

Installing on MacOS X

The easy way

Thanks to the Spyder team and [Continuum](#), you have two alternatives:

1. Use the [Anaconda](#) Python distribution.
2. Use our DMG installers, which can be found [here](#).

Note: The minimal version to run our DMG's is Mavericks (10.9) since Spyder 2.3.5. Previous versions work on Lion (10.7) or higher.

The hard way

Thanks to the *MacPorts* project, Spyder can be installed using its `port` package manager. There are [several versions](#) available from which you can choose from.

Warning: It is known that the MacPorts version of Spyder is raising this error: `ValueError: unknown locale: UTF-8`, which doesn't let it start correctly.

To fix it you will have to set these environment variables in your `~/.profile` (or `~/.bashrc`) manually:

```
export LANG=en_US.UTF-8
export LC_ALL=en_US.UTF-8
```

Installing on Linux

Please refer to the *Requirements* section to see what other packages you might need.

1. Ubuntu:

- Using the official package manager: `sudo apt-get install spyder`.

Note: This package could be slightly outdated. If you find that is the case, please use the Debian package mentioned below.

- Using the `pip` package manager:
 - Installing: `sudo pip install spyder`
 - Updating: `sudo pip install -U spyder`

2. Debian Unstable:

Using the package manager: `sudo apt-get install spyder`

The Spyder's official Debian package is available [here](#)

3. Other Distributions

Spyder is also available in other GNU/Linux distributions, like

- [Archlinux](#)
- [Fedora](#)
- [Gentoo](#)
- [openSUSE](#)
- [Mageia](#)

Please refer to your distribution's documentation to learn how to install it there.

Installing or running directly from source

Requirements

The requirements to run Spyder are:

- Python 2.7 or ≥ 3.3
- PyQt5 ≥ 5.2 or PyQt4 $\geq 4.6.0$ (PyQt5 is recommended).
- Qtconsole $\geq 4.2.0$ – for an enhanced Python interpreter.
- Rope $\geq 0.9.4$ and Jedi $\geq 0.9.0$ – for code completion, go-to-definition and calltips on the Editor.
- Pyflakes – for real-time code analysis.
- Sphinx – for the Help pane rich text mode and to get our documentation.
- Pygments ≥ 2.0 – for syntax highlighting and code completion in the Editor of all file types it supports.

- `Pylint` – for static code analysis.
- `Pycodestyle` – for style analysis.
- `Psutil` – for memory/CPU usage in the status bar.
- `Nbconvert` – to manipulate Jupyter notebooks on the Editor.
- `Qtawesome` $\geq 0.4.1$ – for an icon theme based on FontAwesome.
- `Pickleshare` – To show import completions on the Editor and Consoles.
- `PyZMQ` – To run introspection services on the Editor asynchronously.
- `QtPy` $\geq 1.1.0$ – To run Spyder with PyQt4 or PyQt5 seamlessly.
- `Chardet` $\geq 2.0.0$ – Character encoding auto-detection in Python.
- `Numpydoc` Used by Jedi to get return types for functions with Numpydoc docstrings.

Optional modules

- `Matplotlib` ≥ 1.0 – for 2D and 3D plotting in the consoles.
- `Pandas` $\geq 0.13.1$ – for view and editing DataFrames and Series in the Variable Explorer.
- `Numpy` – for view and editing two or three dimensional arrays in the Variable Explorer.
- `Sympy` $\geq 0.7.3$ – for working with symbolic mathematics in the IPython console.
- `Scipy` – for importing Matlab workspace files in the Variable Explorer.
- `Cython` ≥ 0.21 – Run Cython files or Python files that depend on Cython libraries in the IPython console.

Installation procedure

1. If you use Anaconda, you need to run this command to install Spyder:

```
conda install spyder
```

2. If you don't use Anaconda, you need to run:

```
pip install --upgrade spyder
```

Run without installing

You can execute Spyder without installing it first by following these steps:

1. Unzip the source package
2. Change current directory to the unzipped directory
3. Run Spyder with the command `python bootstrap.py`
4. (*Optional*) Build the documentation with `python setup.py build_doc`.

This is especially useful for beta-testing, troubleshooting and development of Spyder itself.

Installing the development version

If you want to try the next Spyder version, you have to:

1. Install Spyder *requirements*
2. Install [Git](#), a powerful source control management tool.
3. Clone the Spyder source code repository with the command:

```
git clone https://github.com/spyder-ide/spyder.git
```

4. To keep your repository up-to-date, run

```
git pull
```

inside the cloned directory.

5. (*Optional*) If you want to read the documentation, you must build it first with the command

```
python setup.py build_doc
```

Help and support

Spyder websites:

- For bug reports and feature requests you can go to our [website](#).
- For discussions and help requests, you can subscribe to our [Google Group](#).

Command line options

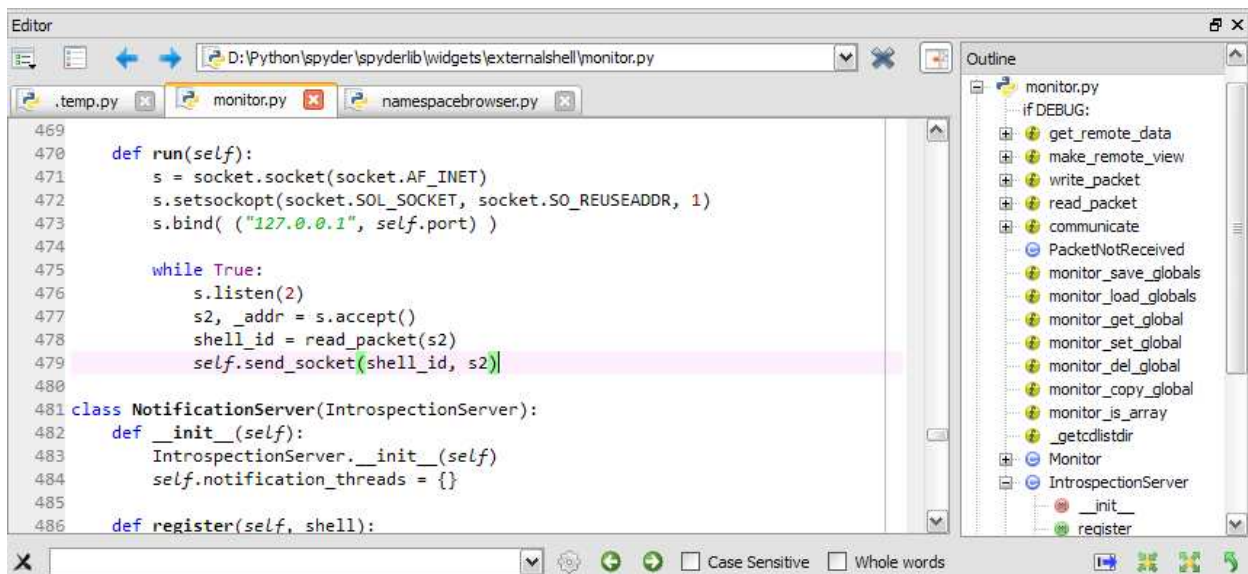
Spyder's command line options are the following:

Options:

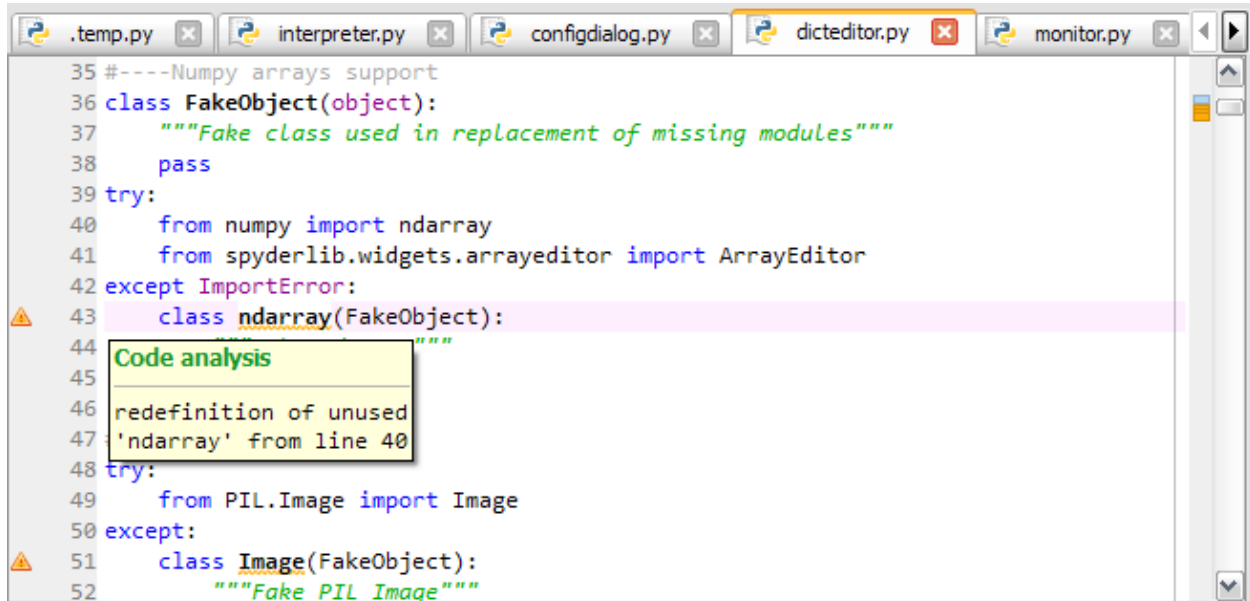
-h, --help	show this help message and exit
--new-instance	Run a new instance of Spyder, even if the single instance mode has been turned on (default)
--defaults	Reset configuration settings to defaults
--reset	Remove all configuration files!
--optimize	Optimize Spyder bytecode (this may require administrative privileges)
-w WORKING_DIRECTORY, --workdir=WORKING_DIRECTORY	Default working directory
--show-console	Do not hide parent console window (Windows)
--multithread	Internal console is executed in another thread (separate from main application thread)
--profile	Profile mode (internal test, not related with Python profiling)
--window-title=WINDOW_TITLE	String to show in the main window title

Spyder's text editor is a multi-language editor with features such as syntax coloring, code analysis (real-time code analysis powered by *pyflakes* and advanced code analysis using *pylint*), introspection capabilities such as code completion, calltips and go-to-definition features (powered by *rope*), function/class browser, horizontal/vertical splitting features, etc.

Function/class/method browser:

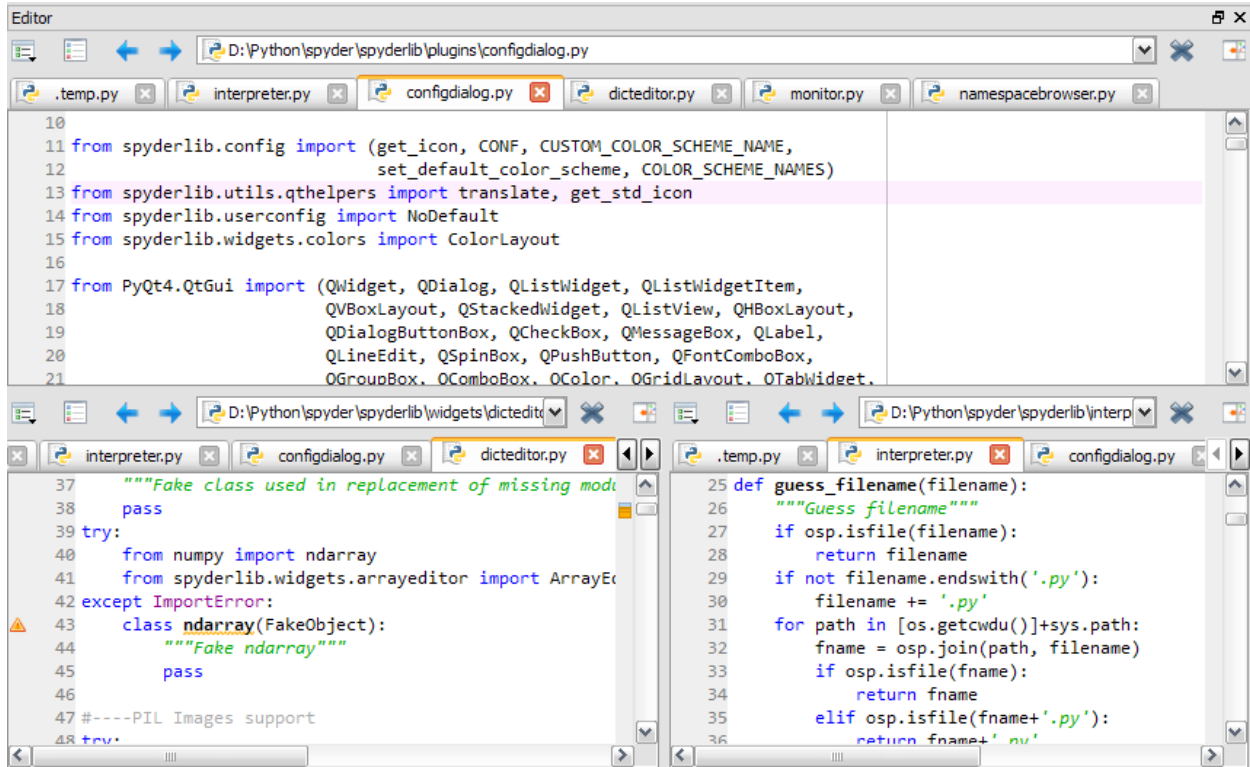


Code analysis with *pyflakes*:



```
.temp.py x interpreter.py x configdialog.py x dicteditor.py x monitor.py x
35 #---Numpy arrays support
36 class FakeObject(object):
37     """Fake class used in replacement of missing modules"""
38     pass
39 try:
40     from numpy import ndarray
41     from spyderlib.widgets.arrayeditor import ArrayEditor
42 except ImportError:
43     class ndarray(FakeObject):
44         """
45         Code analysis
46         redefinition of unused
47         'ndarray' from line 40
48     try:
49         from PIL.Image import Image
50 except:
51     class Image(FakeObject):
52         """Fake PIL Image"""
```

Horizontal/vertical splitting feature:



How to define a code cell

A “code cell” is a concept similar to MATLAB’s “cell” (except that there is no “cell mode” in Spyder), i.e. a block of lines to be executed at once in the current interpreter (Python or IPython). Every script may be divided in as many cells as needed.

Cells are separated by lines starting with:

- `###` (standard cell separator)
- `# %%` (standard cell separator, when file has been edited with Eclipse)
- `# <codecell>` (IPython notebook cell separator)

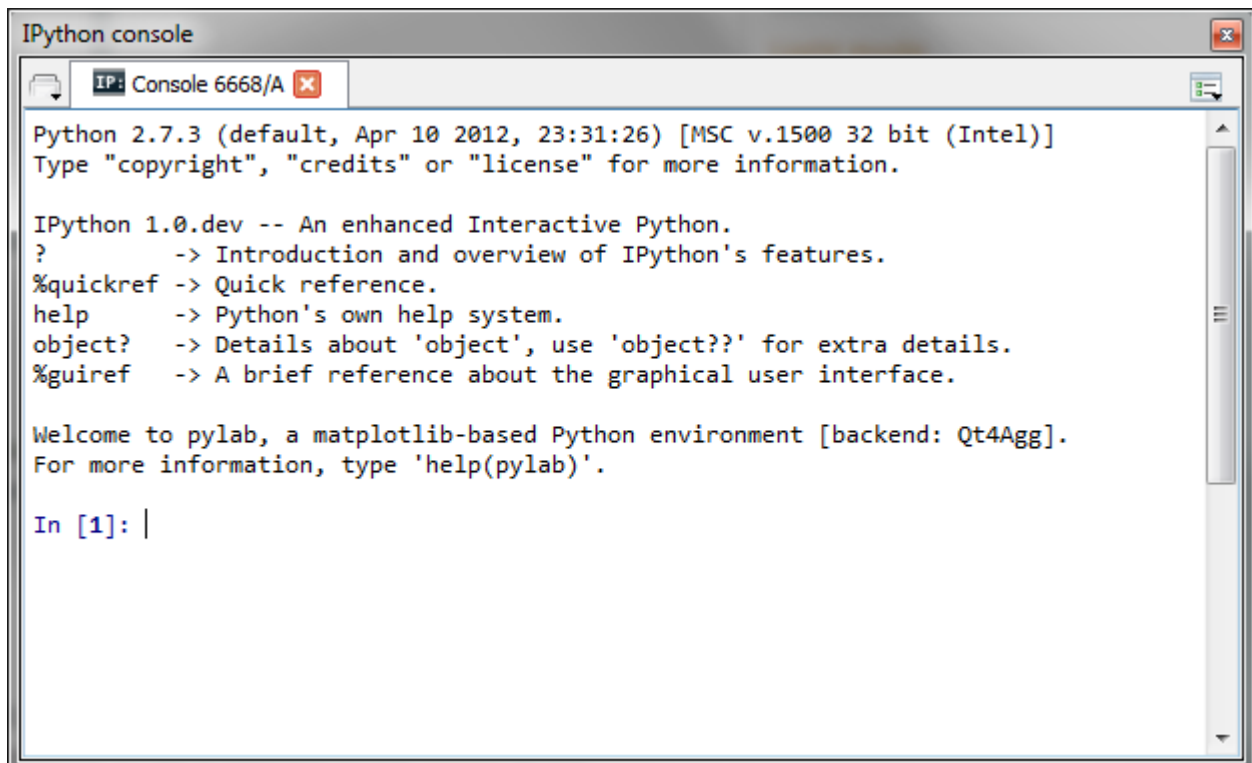
Related plugins:

- *Console*
- *File Explorer*
- *Find in files*

CHAPTER 5

IPython Console

Spyder's **IPython Console** implements a full two-process IPython session where a lightweight front-end interface connects to a full IPython kernel on the back end. Visit the IPython project website for full documentation of IPython's many features.



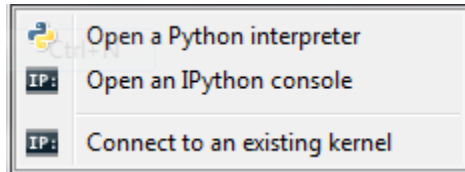
```
IPython console
IP: Console 6668/A x
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)]
Type "copyright", "credits" or "license" for more information.

IPython 1.0.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.
%gui?ref   -> A brief reference about the graphical user interface.

Welcome to pylab, a matplotlib-based Python environment [backend: Qt4Agg].
For more information, type 'help(pylab)'.

In [1]: |
```

From the Consoles menu, Spyder can launch **IPython Console** instances that attach to kernels that are managed by Spyder itself or it can connect to external kernels that are managed by IPython Qt Console sessions or the IPython Notebook.



When “Connect to an existing kernel” is selected, Spyder prompts for the kernel connection file details:



IPython Consoles that are attached to kernels that were created by Spyder support the following features:

- Code completion
- Variable explorer with GUI-based editors for arrays, lists, dictionaries, strings, etc.
- Debugging with standard Python debugger (*pdb*): at each breakpoint the corresponding script is opened in the *Editor* at the breakpoint line number
- User Module Deleter (see *Console* for more details)

IPython Consoles attached to external kernels support a smaller feature set:

- Code completion
- Debugging toolbar integration for launching the debugger and sending debugging step commands to the kernel. Breakpoints must be set manually from the console command line.

Reloading modules: the User Module Reloader (UMR)

When working with Python scripts interactively, one must keep in mind that Python import a module from its source code (on disk) only when parsing the first corresponding import statement. During this first import, the byte code is generated (.pyc file) if necessary and the imported module code object is cached in *sys.modules*. Then, when re-importing the same module, this cached code object will be directly used even if the source code file (.py[w] file) has changed meanwhile.

This behavior is sometimes unexpected when working with the Python interpreter in interactive mode, because one must either always restart the interpreter or remove manually the .pyc files to be sure that changes made in imported modules were taken into account.

The User Module Reloader (UMR) is a Spyder console's exclusive feature that forces the Python interpreter to reload modules completely when executing a Python script.

For example, when UMR is turned on, one may test complex applications within the same Python interpreter without having to restart it every time (restart time may be relatively long when testing GUI-based applications).

Related plugins:

- *Help*
- *Editor*
- *File Explorer*

Debugging in Spyder is supported thanks to the following Python modules:

- *pdb*: the Python debugger, which is included in Python standard library.
- *winpdb*: a graphical frontend to *pdb*, which is an external package (in the *Editor*, press F7 to run *winpdb* on the currently edited script).

Debugging with pdb

The Python debugger is partly integrated in Spyder:

- Breakpoints may be defined in the *Editor*.
 - Simple breakpoints can be set from the Run menu, by keyboard shortcut (F12 by default), or by double-click to the left of line numbers in the *Editor*.
 - Conditional breakpoints can also be set from the Run menu, by keyboard shortcut (Shift+F12 by default), or by Shift+double-click to the left of line numbers in the *Editor*.
- The current frame (debugging step) is highlighted in the *Editor*.
- At each breakpoint, globals may be accessed through the *Variable Explorer*.

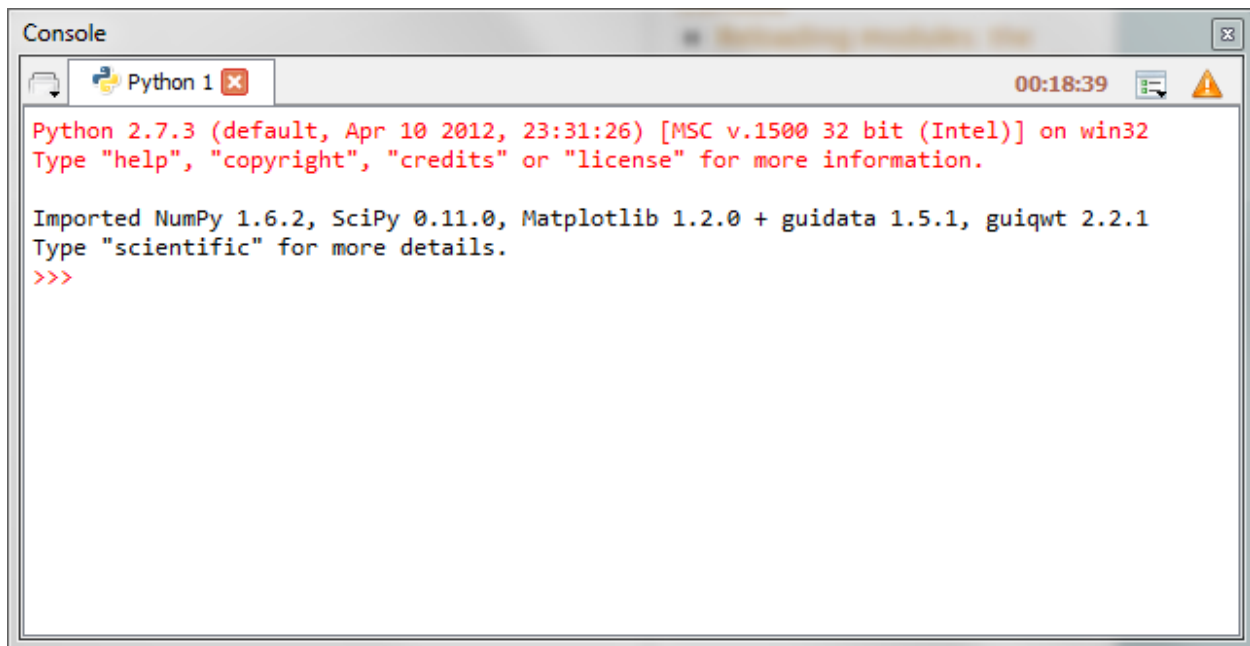
For a simple, yet quite complete introduction to *pdb*, you may read this: <http://pythonconquerstheuniverse.wordpress.com/category/python-debugger/>

Related plugins:

- *Editor*
- *Console*

Console

The **Console** is where you may enter, interact with and visualize data, inside a command interpreter. All the commands entered in the console are executed in a separate process, thus allowing the user to interrupt any process at any time.



```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.

Imported NumPy 1.6.2, SciPy 0.11.0, Matplotlib 1.2.0 + guidata 1.5.1, guiqwt 2.2.1
Type "scientific" for more details.
>>>
```

Many command windows may be created in the **Console**:

- Python interpreter
- Running Python script
- System command window (this terminal emulation window has quite limited features compared to a real terminal: it may be useful on Windows platforms where the system terminal is not much more powerful - on the contrary, on GNU/Linux, a real system terminal is opened, outside Spyder)

Python-based command windows support the following features:

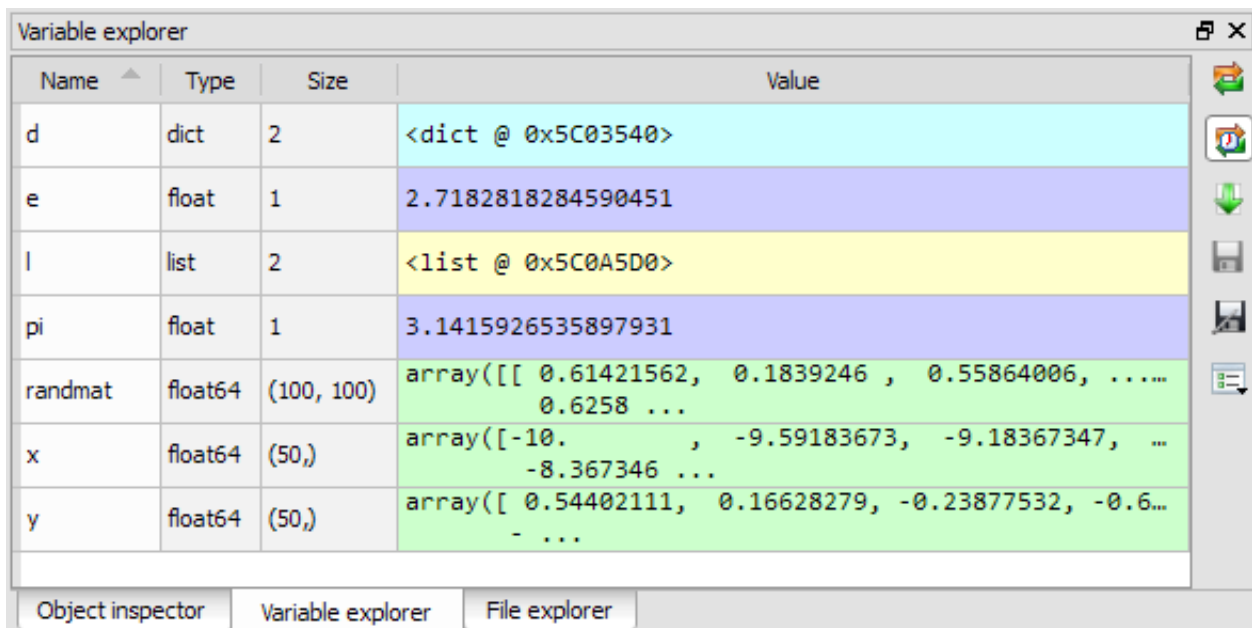
- Code completion and calltips
- Variable explorer with GUI-based editors for arrays, lists, dictionaries, strings, etc.
- Debugging with standard Python debugger (*pdb*): at each breakpoint the corresponding script is opened in the *Editor* at the breakpoint line number
- User Module Deleter (see below)

Related plugins:

- *Help*
- *History log*
- *Editor*
- *File Explorer*

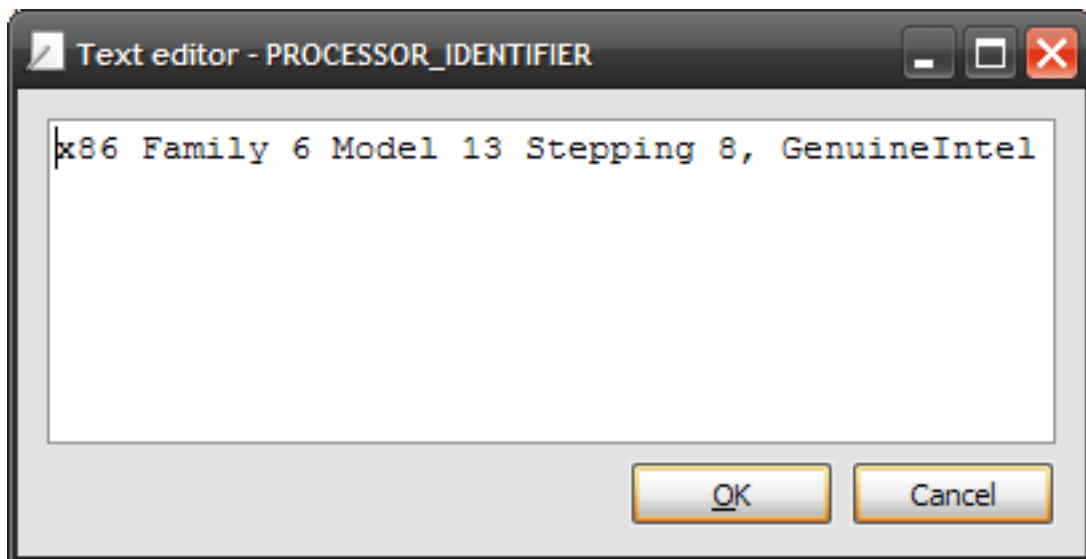
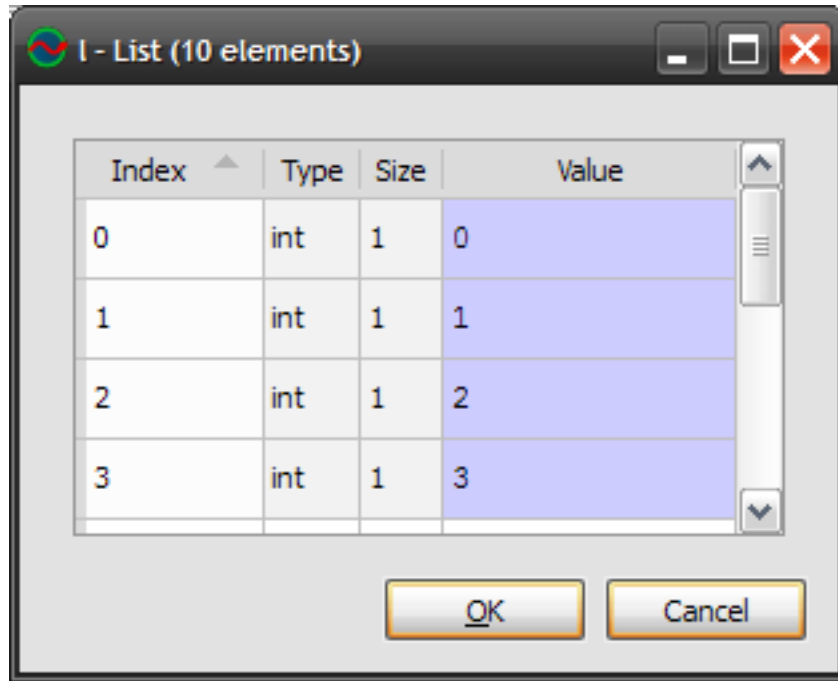
Variable Explorer

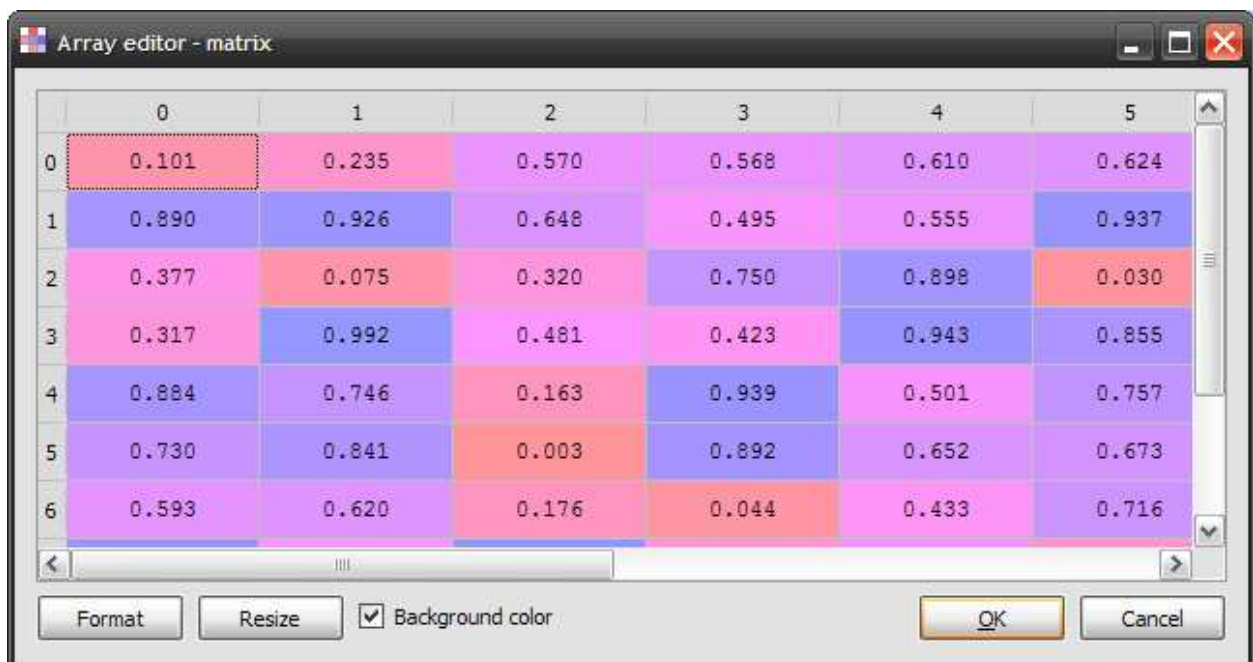
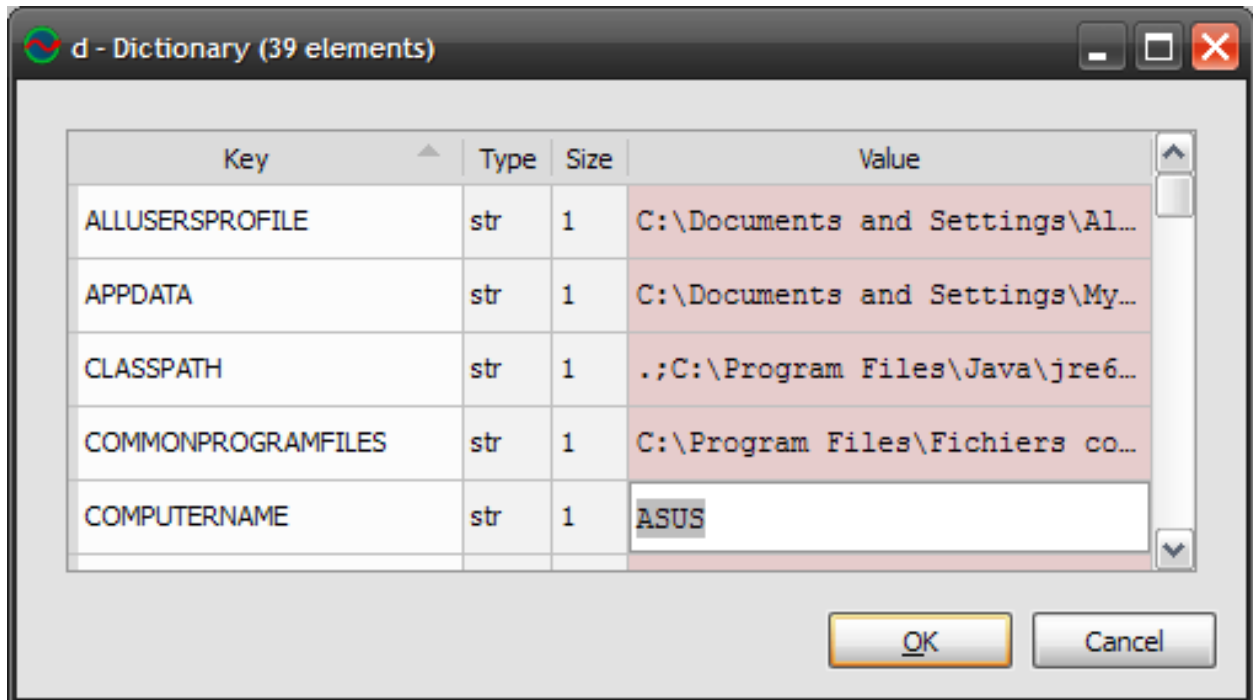
The variable explorer shows the namespace contents (i.e. all global object references) of the current console

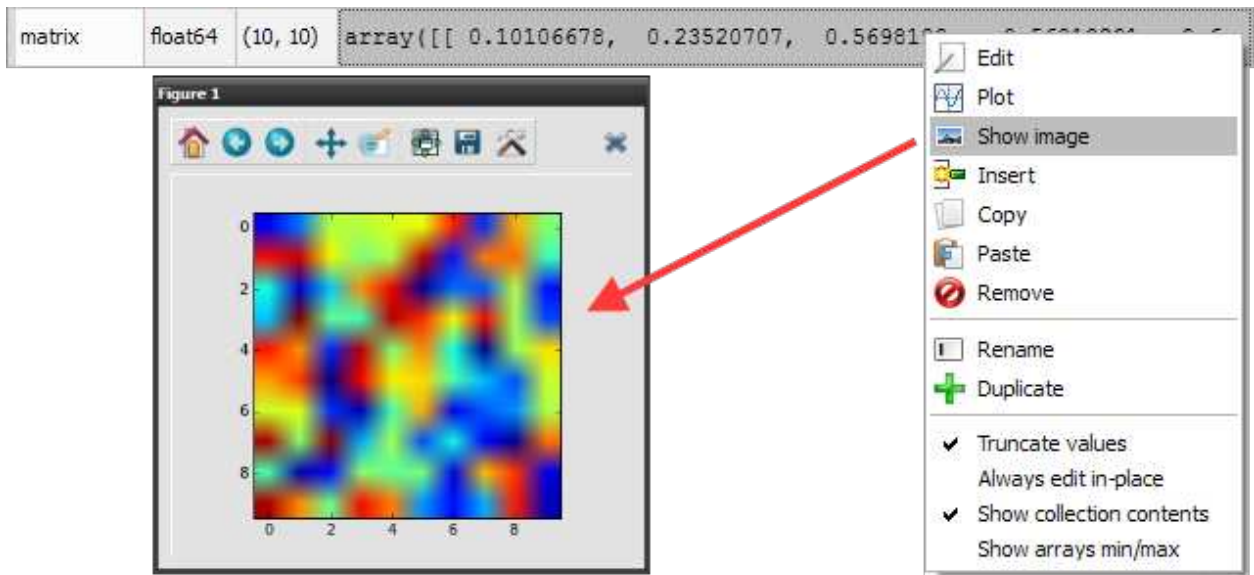
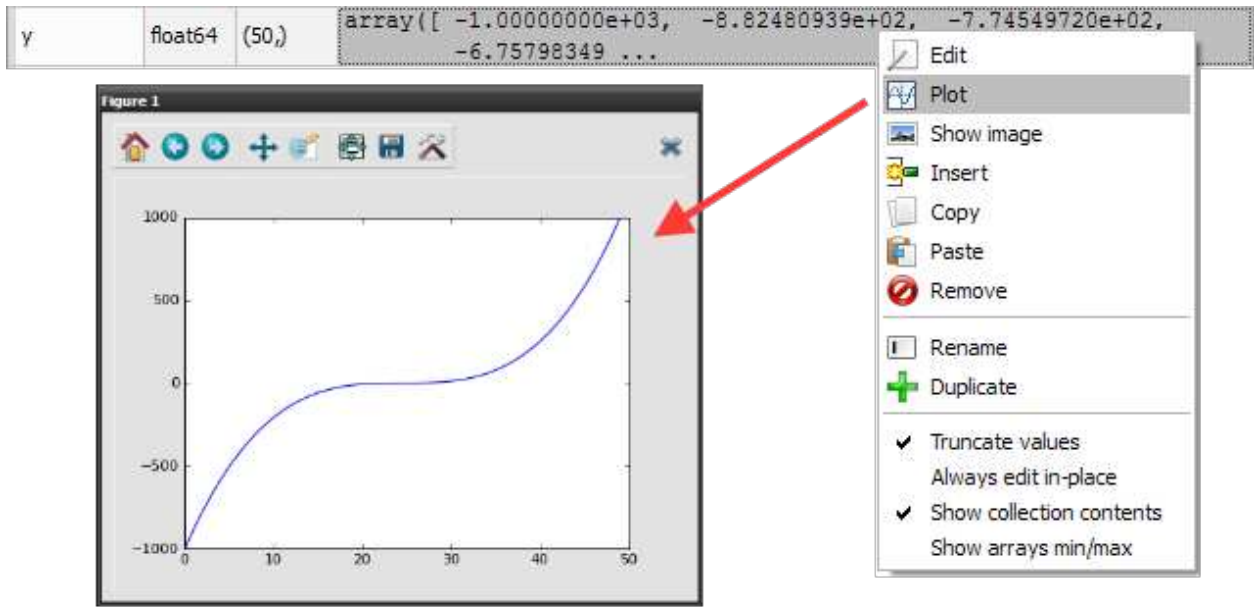


The following screenshots show some interesting features such as editing lists, strings, dictionaries, NumPy arrays, or

plotting/showing NumPy arrays data.







Supported types

The variable explorer can't show all types of objects. The ones currently supported are:

1. *Pandas* DataFrame, TimeSeries and DatetimeIndex objects

2. *NumPy* arrays and matrices
3. *PIL/Pillow* images
4. *datetime* dates
5. Integers
6. Floats
7. Complex numbers
8. Lists
9. Dictionaries
10. Tuples
11. Strings

Related plugins:

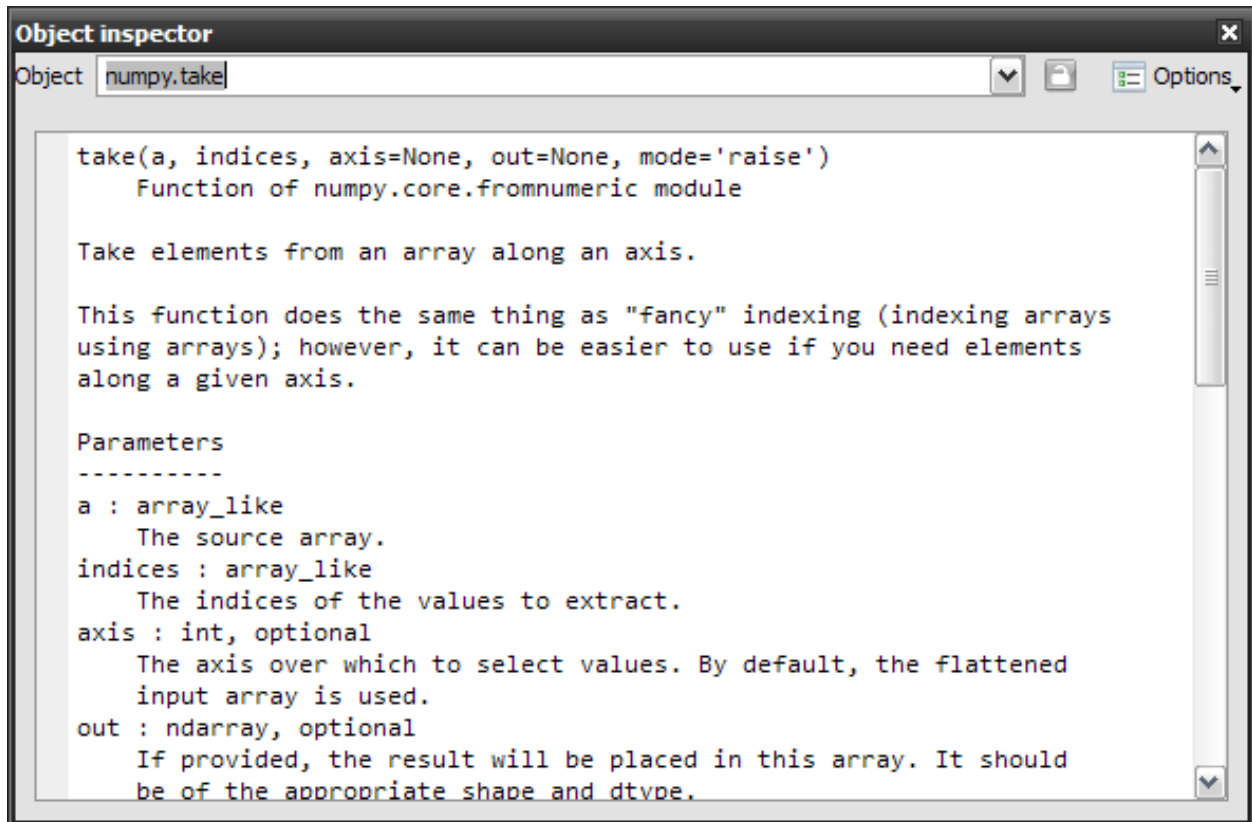
- *IPython Console*

The help plugin works together with the *Console* and the *Editor*: it shows automatically documentation available when the user is instantiating a class or calling a function (pressing the left parenthesis key after a valid function or class name triggers a call in the help pane).

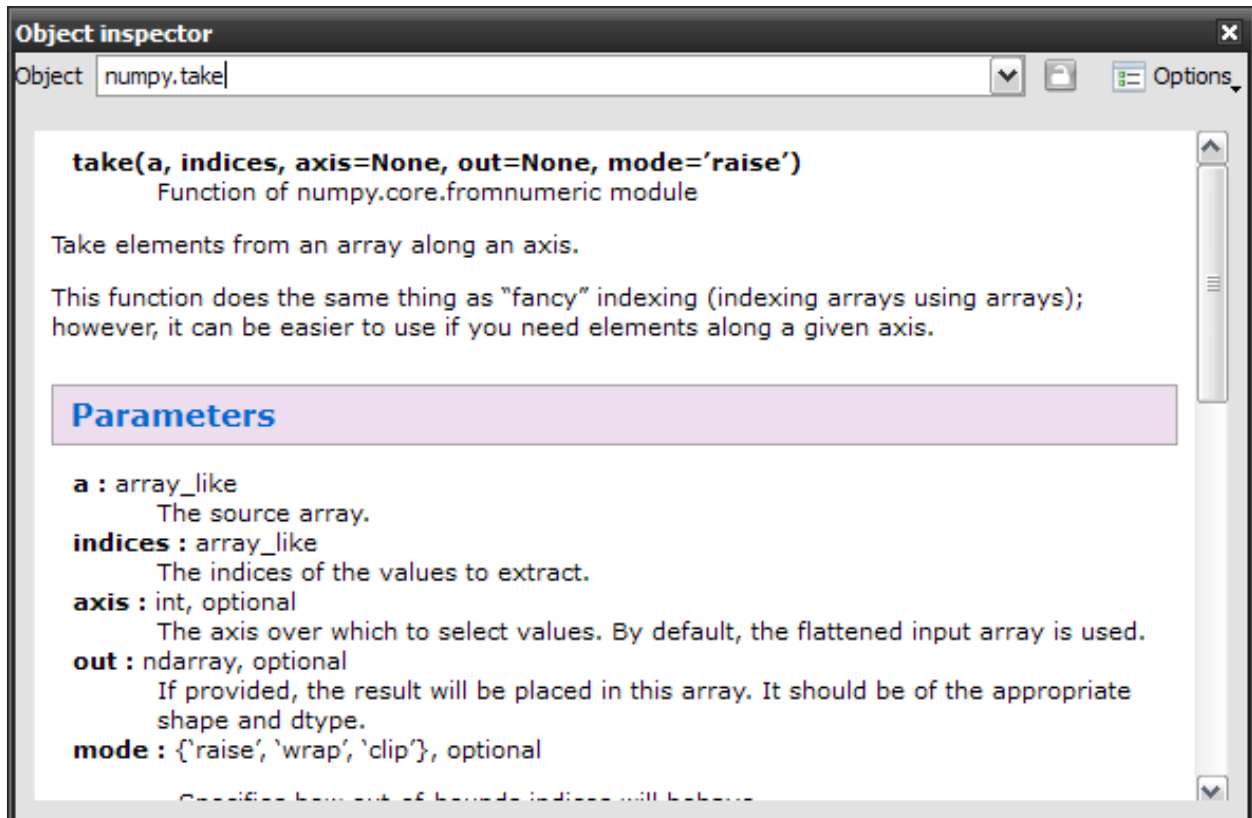
Note that this automatic link may be disabled by pressing the “Lock” button (at the top right corner of the window).

Of course, one can use the documentation viewer directly by entering an object name in the editable combo box field, or by selecting old documentation requests in the combo box.

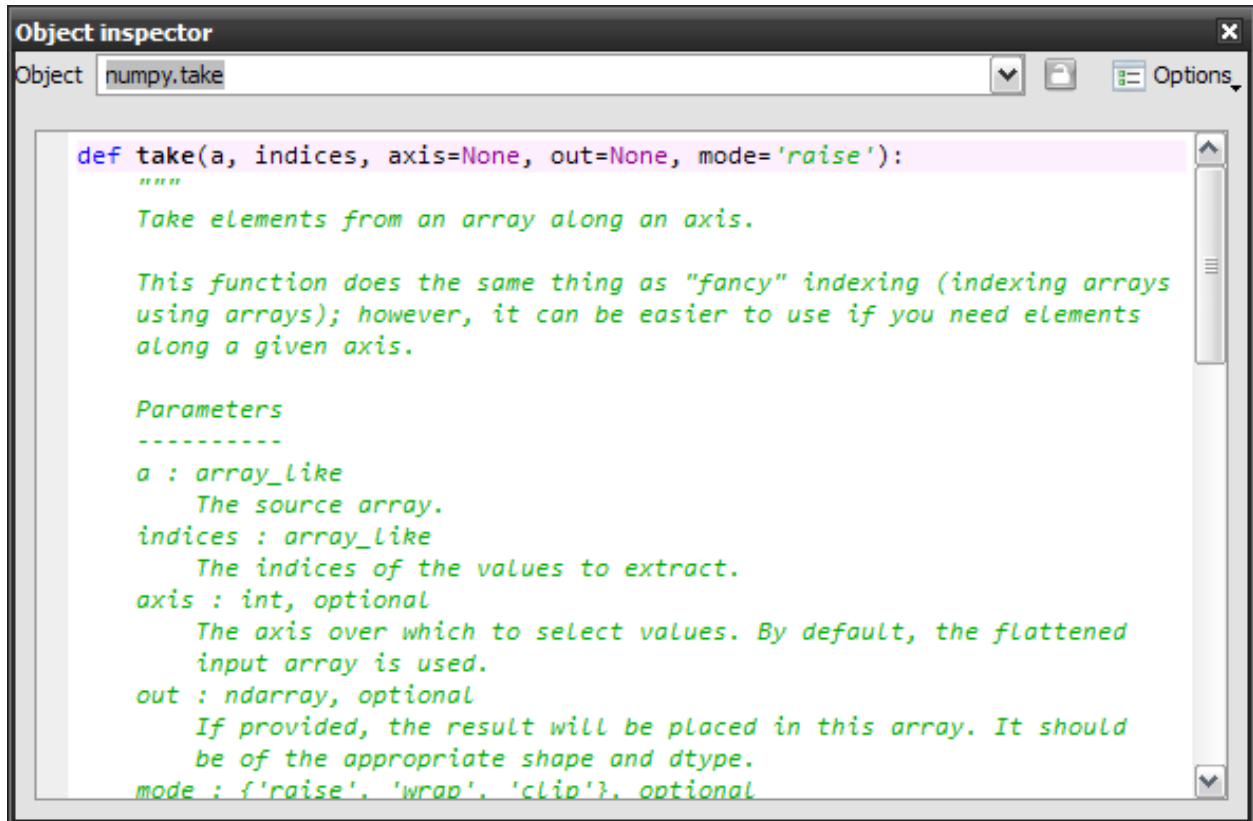
Plain text mode:



Rich text mode:



Sometimes, when docstrings are not available or not sufficient to document the object, the documentation viewer can show the source code (if available, i.e. if the object is pure Python):



The screenshot shows the 'Object inspector' window in Spyder. The 'Object' field contains 'numpy.take'. The main area displays the source code for the 'take' function, including its docstring and parameter descriptions.

```
def take(a, indices, axis=None, out=None, mode='raise'):
    """
    Take elements from an array along an axis.

    This function does the same thing as "fancy" indexing (indexing arrays
    using arrays); however, it can be easier to use if you need elements
    along a given axis.

    Parameters
    -----
    a : array_like
        The source array.
    indices : array_like
        The indices of the values to extract.
    axis : int, optional
        The axis over which to select values. By default, the flattened
        input array is used.
    out : ndarray, optional
        If provided, the result will be placed in this array. It should
        be of the appropriate shape and dtype.
    mode : {'raise', 'wrap', 'clip'}, optional
```

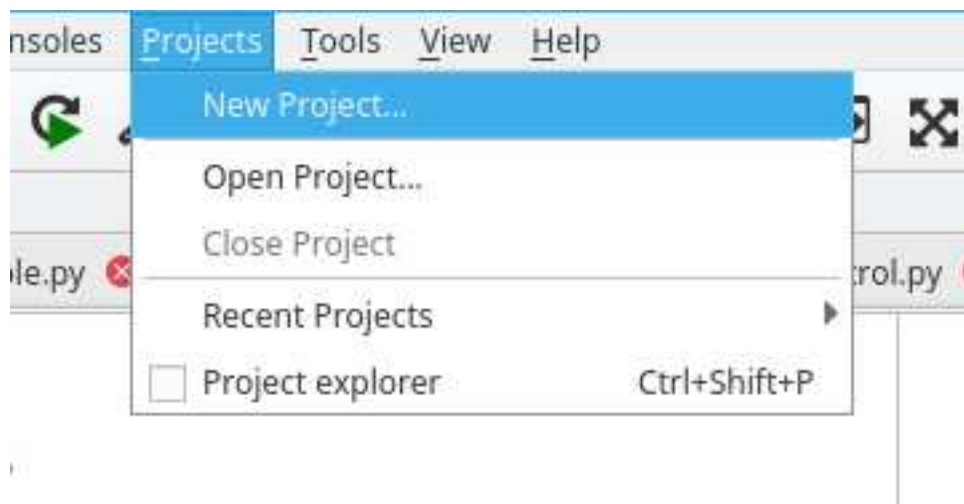
Related plugins:

- *Console*
- *Editor*

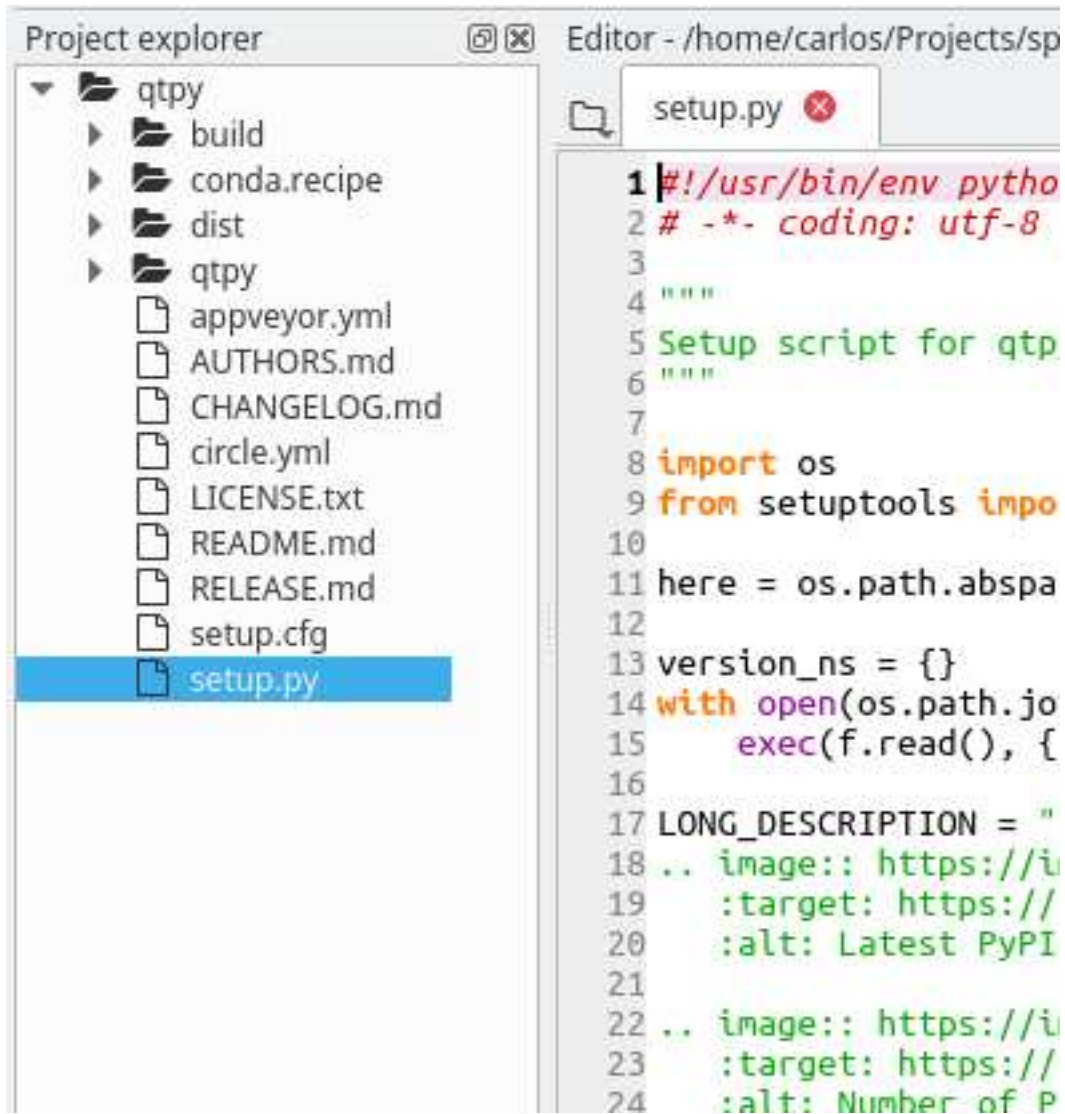
Spyder allows users to associate a given directory with a project. This has two main advantages:

1. Projects remember the list of open files in Editor. This permits to easily work on several coding efforts at the same time.
2. The project's path is added to the list of paths Python looks modules for, so that modules developed as part of a project can be easily imported in any console.

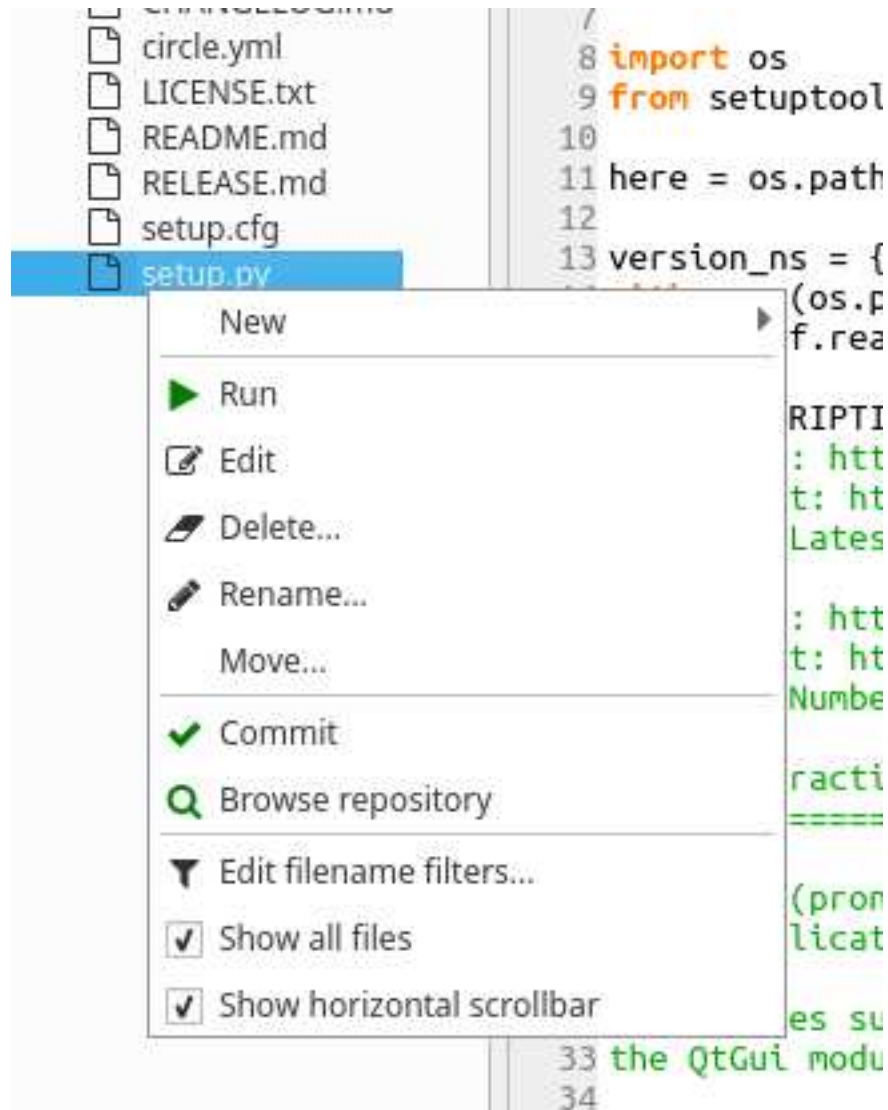
To create a project, it is necessary to select the *New Project* entry from the *Projects* menu:



When a project is activated, the *Project explorer* pane is shown, which presents a tree view structure of the current project



Through this pane it is possible to make several operations on the files that belong to project



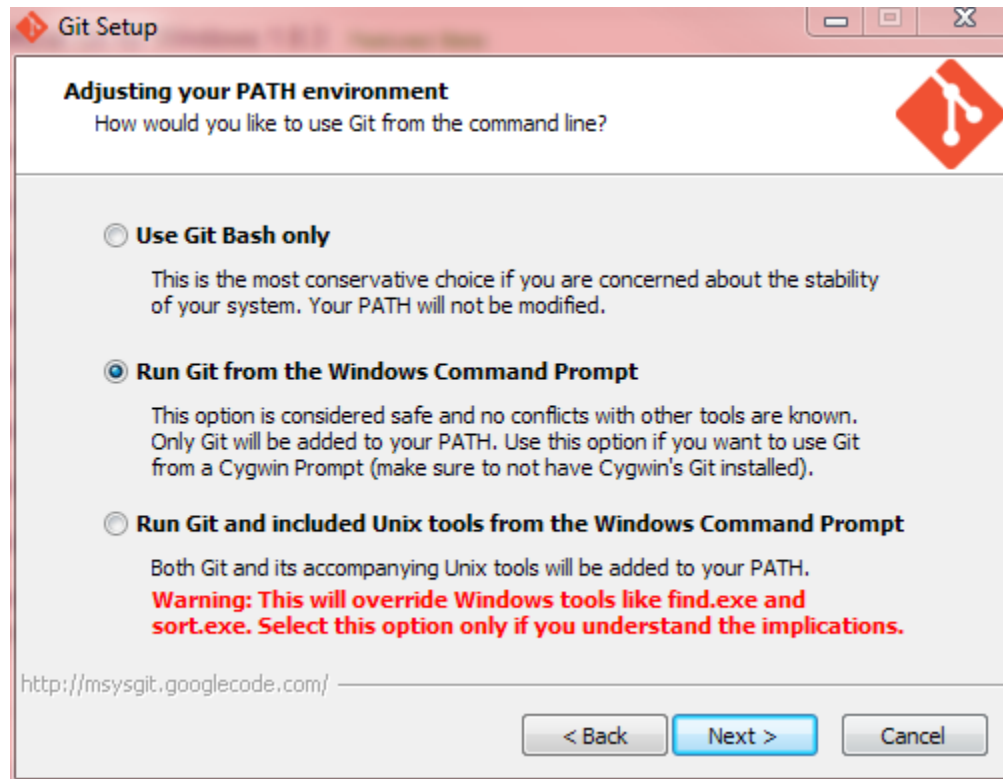
Note: Projects are completely optional and not imposed on users, i.e. users can work without creating any project.

Version Control Integration

Spyder has limited integration with [Git](#) and [Mercurial](#). Commit and browse commands are available by right-clicking on relevant files that reside within an already initialized repository. This menu assumes that certain commands are available on the system path.

- For Mercurial repositories, [TortoiseHG](#) must be installed, and either `thg` or `hgTk` must be on the system path.
- For git repositories, the commands `git` and `gitk` must be on the system path. For Windows systems, the

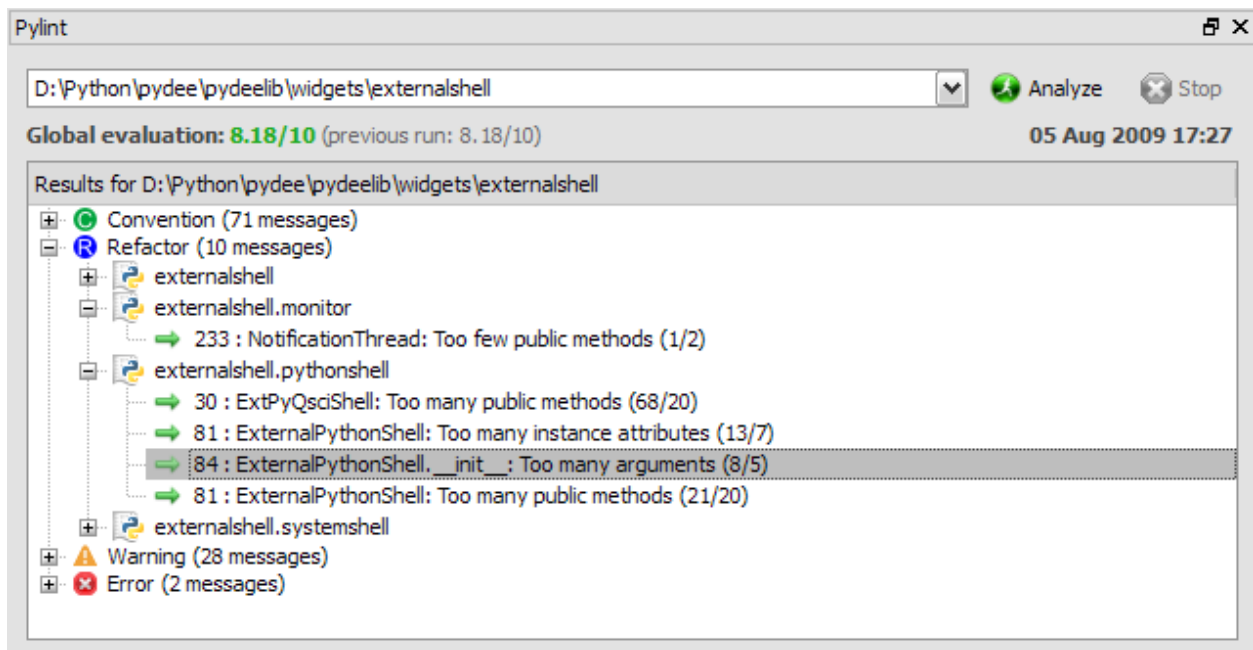
Git for Windows package provides a convenient installer and the option to place common git commands on the system path without creating conflicts with Windows system tools. The second option in the dialog below is generally a safe approach.



CHAPTER 11

Static code analysis

The static code analysis tool may be used directly from the *Editor*, or by entering manually the Python module or package path - i.e. it works either with *.py* (or *.pyw*) Python scripts or with whole Python packages (directories containing an *__init__.py* script).

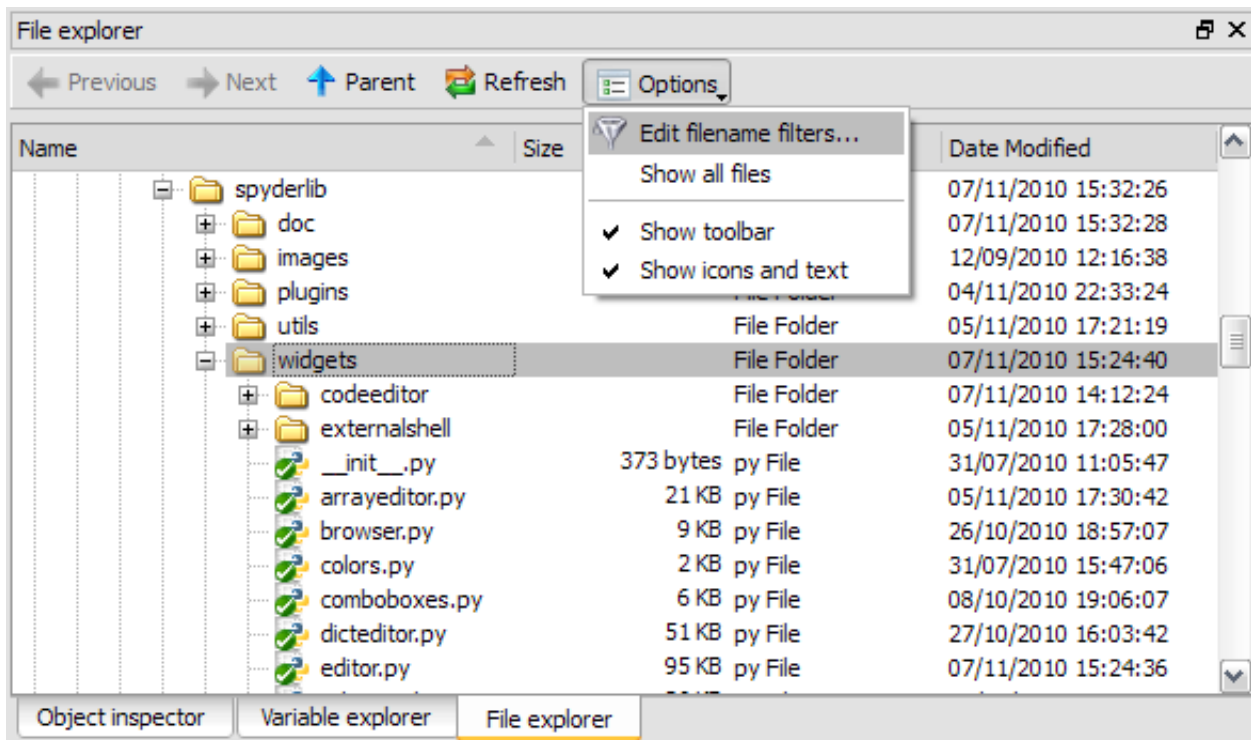


Related plugins:

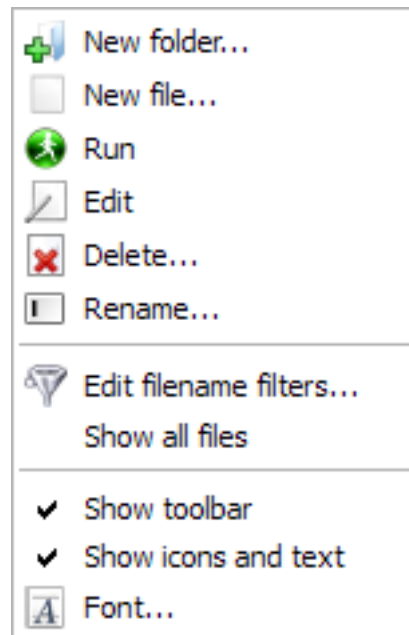
- *Editor*

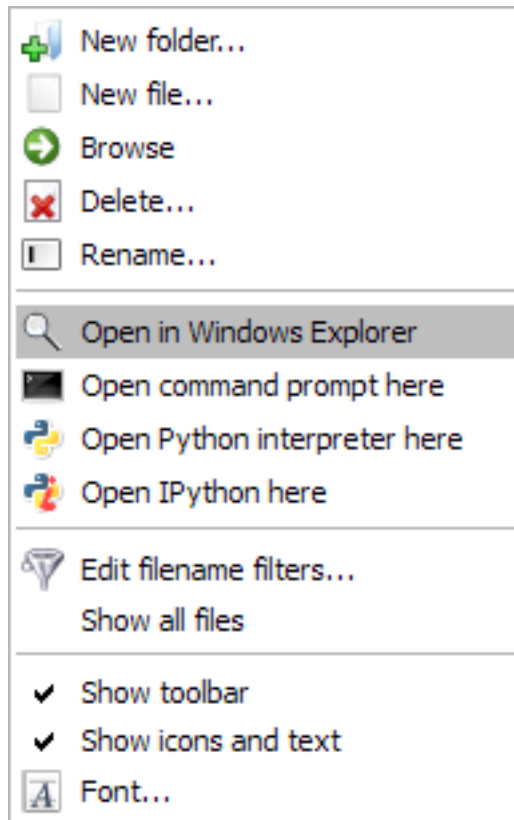
File Explorer

The file explorer pane is a file/directory browser allowing the user to open files with the internal editor or with the appropriate application (Windows only).



Context menus may be used to run a script, open a terminal window or run a Windows explorer window (Windows only):





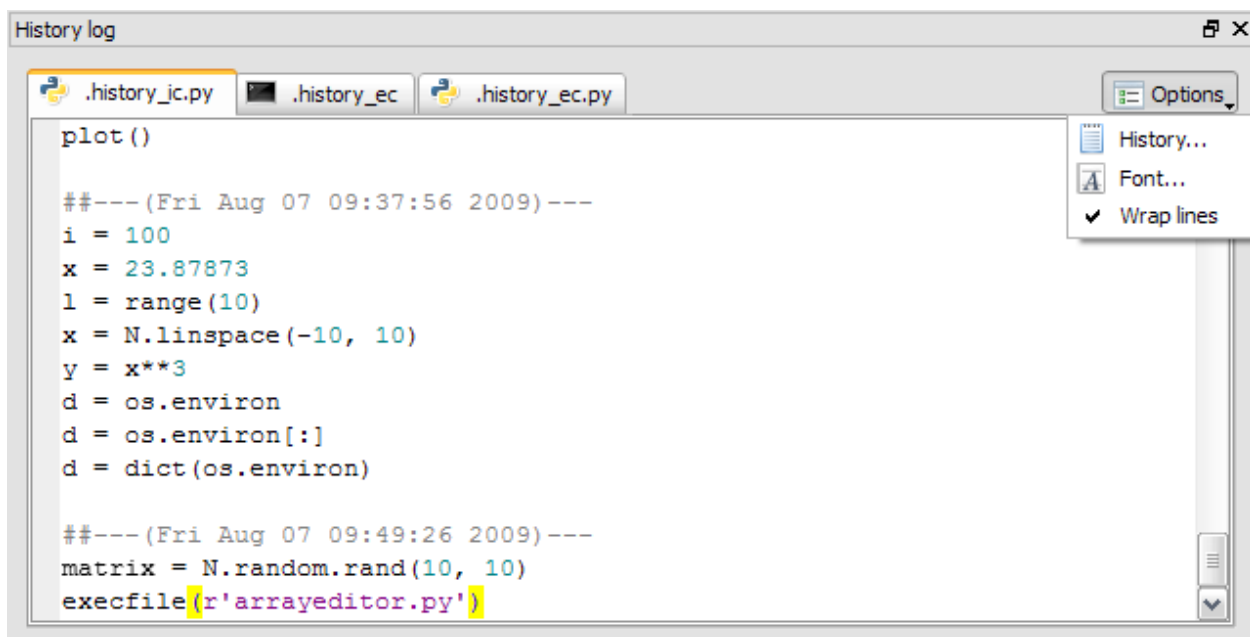
Related plugins:

- *IPython Console*
- *Editor*

CHAPTER 13

History log

The history log plugin collects command histories of Python/IPython interpreters or command windows.



The screenshot shows a window titled "History log" with three tabs: ".history_ic.py", ".history_ec", and ".history_ec.py". The main content area displays a log of Python commands and their execution times. The first log entry is for a session on Friday, August 07, 2009, at 09:37:56. The commands include `plot()`, `i = 100`, `x = 23.87873`, `l = range(10)`, `x = N.linspace(-10, 10)`, `y = x**3`, `d = os.environ`, `d = os.environ[:]`, and `d = dict(os.environ)`. The second log entry is for a session on Friday, August 07, 2009, at 09:49:26, showing `matrix = N.random.rand(10, 10)` and `execfile(r'arrayeditor.py')`. The `execfile` command is highlighted with a yellow background. An "Options" menu is open on the right, showing "History...", "Font...", and "Wrap lines" (checked).

```
plot ()

##--- (Fri Aug 07 09:37:56 2009) ---
i = 100
x = 23.87873
l = range(10)
x = N.linspace(-10, 10)
y = x**3
d = os.environ
d = os.environ[:]
d = dict(os.environ)

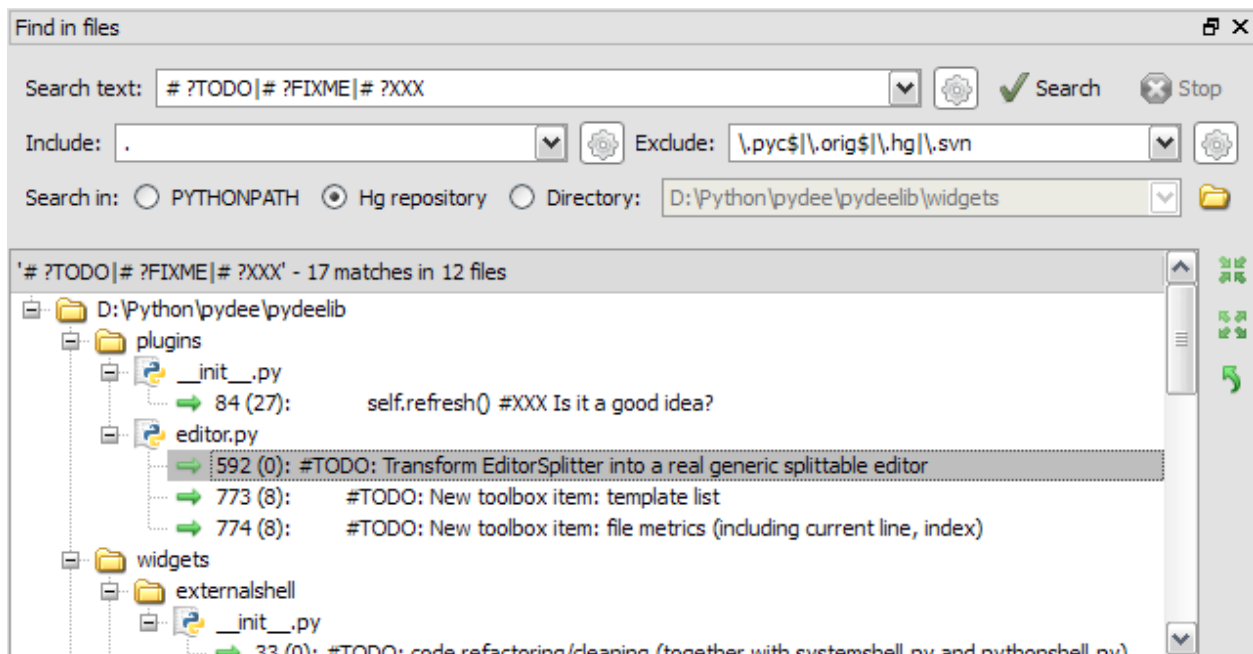
##--- (Fri Aug 07 09:49:26 2009) ---
matrix = N.random.rand(10, 10)
execfile(r'arrayeditor.py')
```

Related plugins:

- *Console*

Find in files

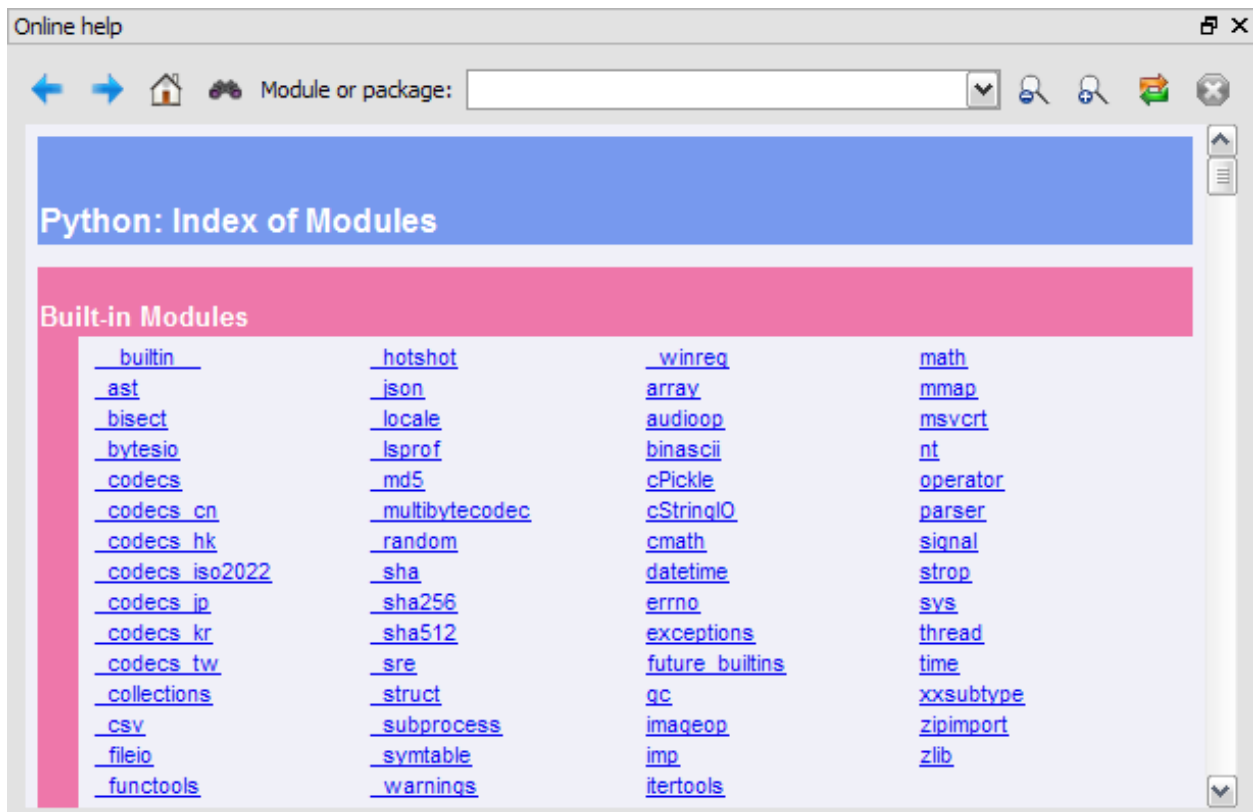
The *Find in Files* plugin provides text search in whole directories or *mercurial* repositories (or even in PYTHONPATH) with regular expression support for maximum search customization.



Related plugins:

- *Editor*

The online help plugin provides an internal web browser to explore dynamically generated Python documentation on installed module, including your own modules (this documentation is provided by a pydoc server running in background).



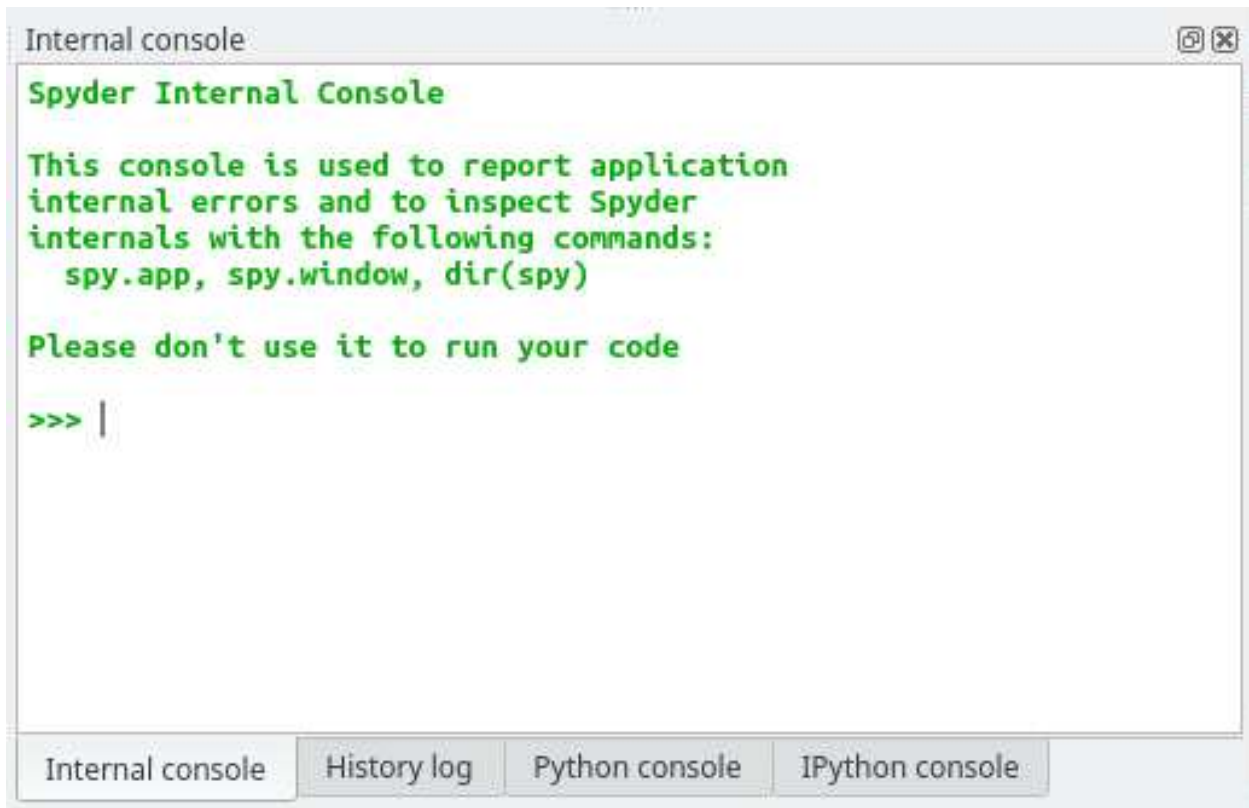
Related plugins:

- *Help*

CHAPTER 16

Internal Console

The **Internal Console** is dedicated to Spyder internal debugging or may be used as an embedded Python console in your own application. All the commands entered in the internal console are executed in the same process as Spyder's, but the Internal Console may be executed in a separate thread (this is optional and for example this is not the case in Spyder itself).



The internal console support the following features:

- Code completion and calltips

Indices and tables:

- genindex
- search