
Using Python

Release 2.6.3

Guido van Rossum
Fred L. Drake, Jr., editor

October 06, 2009

Python Software Foundation
Email: docs@python.org

CONTENTS

1	Command line and environment	3
1.1	Command line	3
1.2	Environment variables	7
2	Using Python on Unix platforms	9
2.1	Getting and installing the latest version of Python	9
2.2	Building Python	10
2.3	Python-related paths and files	10
2.4	Miscellaneous	10
2.5	Editors	10
3	Using Python on Windows	13
3.1	Installing Python	13
3.2	Alternative bundles	13
3.3	Configuring Python	14
3.4	Additional modules	15
3.5	Compiling Python on Windows	16
3.6	Other resources	16
4	Using Python on a Macintosh	17
4.1	Getting and Installing MacPython	17
4.2	The IDE	18
4.3	Installing Additional Python Packages	18
4.4	GUI Programming on the Mac	18
4.5	Distributing Python Applications on the Mac	19
4.6	Application Scripting	19
4.7	Other Resources	19
A	Glossary	21
B	About these documents	27
B.1	Contributors to the Python Documentation	27
C	History and License	29
C.1	History of the software	29
C.2	Terms and conditions for accessing or otherwise using Python	30
C.3	Licenses and Acknowledgements for Incorporated Software	32
D	Copyright	41
	Index	43

This part of the documentation is devoted to general information on the setup of the Python environment on different platform, the invocation of the interpreter and things that make working with Python easier.

COMMAND LINE AND ENVIRONMENT

The CPython interpreter scans the command line and the environment for various settings.

Note: Other implementations' command line schemes may differ. See *Alternate Implementations* (in *The Python Language Reference*) for further resources.

1.1 Command line

When invoking Python, you may specify any of these options:

```
python [-dEiOQsStuUvX3?] [-c command | -m module-name | script | - ] [args]
```

The most common use case is, of course, a simple invocation of a script:

```
python myscript.py
```

1.1.1 Interface options

The interpreter interface resembles that of the UNIX shell, but provides some additional methods of invocation:

- When called with standard input connected to a tty device, it prompts for commands and executes them until an EOF (an end-of-file character, you can produce that with *Ctrl-D* on UNIX or *Ctrl-Z, Enter* on Windows) is read.
- When called with a file name argument or with a file as standard input, it reads and executes a script from that file.
- When called with a directory name argument, it reads and executes an appropriately named script from that directory.
- When called with `-c command`, it executes the Python statement(s) given as *command*. Here *command* may contain multiple statements separated by newlines. Leading whitespace is significant in Python statements!
- When called with `-m module-name`, the given module is located on the Python module path and executed as a script.

In non-interactive mode, the entire input is parsed before it is executed.

An interface option terminates the list of options consumed by the interpreter, all consecutive arguments will end up in `sys.argv` – note that the first element, subscript zero (`sys.argv[0]`), is a string reflecting the program's source.

-c <command>

Execute the Python code in *command*. *command* can be one or more statements separated by newlines, with significant leading whitespace as in normal module code.

If this option is given, the first element of `sys.argv` will be `"-c"` and the current directory will be added to the start of `sys.path` (allowing modules in that directory to be imported as top level modules).

-m <module-name>

Search `sys.path` for the named module and execute its contents as the `__main__` module.

Since the argument is a *module* name, you must not give a file extension (`.py`). The `module-name` should be a valid Python module name, but the implementation may not always enforce this (e.g. it may allow you to use a name that includes a hyphen).

Note: This option cannot be used with builtin modules and extension modules written in C, since they do not have Python module files. However, it can still be used for precompiled modules, even if the original source file is not available.

If this option is given, the first element of `sys.argv` will be the full path to the module file. As with the `-c` option, the current directory will be added to the start of `sys.path`.

Many standard library modules contain code that is invoked on their execution as a script. An example is the `timeit` module:

```
python -mtimeit -s 'setup here' 'benchmarked code here'
python -mtimeit -h # for details
```

See Also:

`runpy.run_module()` The actual implementation of this feature.

PEP 338 – Executing modules as scripts

New in version 2.4.Changed in version 2.5: The named module can now be located inside a package.

-

Read commands from standard input (`sys.stdin`). If standard input is a terminal, `-i` is implied.

If this option is given, the first element of `sys.argv` will be `"-"` and the current directory will be added to the start of `sys.path`.

<script>

Execute the Python code contained in *script*, which must be a filesystem path (absolute or relative) referring to either a Python file, a directory containing a `__main__.py` file, or a zipfile containing a `__main__.py` file.

If this option is given, the first element of `sys.argv` will be the script name as given on the command line.

If the script name refers directly to a Python file, the directory containing that file is added to the start of `sys.path`, and the file is executed as the `__main__` module.

If the script name refers to a directory or zipfile, the script name is added to the start of `sys.path` and the `__main__.py` file in that location is executed as the `__main__` module. Changed in version 2.5: Directories and zipfiles containing a `__main__.py` file at the top level are now considered valid Python scripts.

If no interface option is given, `-i` is implied, `sys.argv[0]` is an empty string (`" "`) and the current directory will be added to the start of `sys.path`.

See Also:

Invoking the Interpreter (in *Python Tutorial*)

1.1.2 Generic options

-?

-h

-help

Print a short description of all command line options. Changed in version 2.5: The `--help` variant.

-v

-version

Print the Python version number and exit. Example output could be:

Python 2.5.1

Changed in version 2.5: The `--version` variant.

1.1.3 Miscellaneous options

-B

If given, Python won't try to write `.pyc` or `.pyo` files on the import of source modules. See also [PYTHONDONTWRITEBYTECODE](#). New in version 2.6.

-d

Turn on parser debugging output (for wizards only, depending on compilation options). See also [PYTHON-DEBUG](#).

-E

Ignore all **PYTHON*** environment variables, e.g. [PYTHONPATH](#) and [PYTHONHOME](#), that might be set. New in version 2.2.

-i

When a script is passed as first argument or the `-c` option is used, enter interactive mode after executing the script or the command, even when `sys.stdin` does not appear to be a terminal. The [PYTHONSTARTUP](#) file is not read.

This can be useful to inspect global variables or a stack trace when a script raises an exception. See also [PYTHONINSPECT](#).

-O

Turn on basic optimizations. This changes the filename extension for compiled (*bytecode*) files from `.pyc` to `.pyo`. See also [PYTHONOPTIMIZE](#).

-OO

Discard docstrings in addition to the `-O` optimizations.

-Q <arg>

Division control. The argument must be one of the following:

old division of `int/int` and `long/long` return an `int` or `long` (*default*)

new new division semantics, i.e. division of `int/int` and `long/long` returns a `float`

warn old division semantics with a warning for `int/int` and `long/long`

warnall old division semantics with a warning for all uses of the division operator

See Also:

`Tools/scripts/fixdiv.py` for a use of `warnall`

[PEP 238](#) – Changing the division operator

-s

Don't add user site directory to `sys.path` New in version 2.6.

See Also:

[PEP 370](#) – Per user site-packages directory

-S

Disable the import of the module `site` and the site-dependent manipulations of `sys.path` that it entails.

-t

Issue a warning when a source file mixes tabs and spaces for indentation in a way that makes it depend on the worth of a tab expressed in spaces. Issue an error when the option is given twice (`-tt`).

-u

Force `stdin`, `stdout` and `stderr` to be totally unbuffered. On systems where it matters, also put `stdin`, `stdout` and `stderr` in binary mode.

Note that there is internal buffering in `file.readlines()` and *File Objects* (in *The Python Library Reference*) (for `line in sys.stdin`) which is not influenced by this option. To work around this, you will want to use `file.readline()` inside a `while 1: loop`.

See also [PYTHONUNBUFFERED](#).

-v

Print a message each time a module is initialized, showing the place (filename or built-in module) from which it is loaded. When given twice (`-vv`), print a message for each file that is checked for when searching for a module. Also provides information on module cleanup at exit. See also [PYTHONVERBOSE](#).

-W arg

Warning control. Python's warning machinery by default prints warning messages to `sys.stderr`. A typical warning message has the following form:

```
file:line: category: message
```

By default, each warning is printed once for each source line where it occurs. This option controls how often warnings are printed.

Multiple `-W` options may be given; when a warning matches more than one option, the action for the last matching option is performed. Invalid `-W` options are ignored (though, a warning message is printed about invalid options when the first warning is issued).

Warnings can also be controlled from within a Python program using the `warnings` module.

The simplest form of argument is one of the following action strings (or a unique abbreviation):

ignore Ignore all warnings.

default Explicitly request the default behavior (printing each warning once per source line).

all Print a warning each time it occurs (this may generate many messages if a warning is triggered repeatedly for the same source line, such as inside a loop).

module Print each warning only the first time it occurs in each module.

once Print each warning only the first time it occurs in the program.

error Raise an exception instead of printing a warning message.

The full form of argument is:

```
action:message:category:module:line
```

Here, *action* is as explained above but only applies to messages that match the remaining fields. Empty fields match all values; trailing empty fields may be omitted. The *message* field matches the start of the warning message printed; this match is case-insensitive. The *category* field matches the warning category. This must be a class name; the match test whether the actual warning category of the message is a subclass of the specified warning category. The full class name must be given. The *module* field matches the (fully-qualified) module name; this match is case-sensitive. The *line* field matches the line number, where zero matches all line numbers and is thus equivalent to an omitted line number.

See Also:

`warnings` – the warnings module

[PEP 230](#) – Warning framework

-x

Skip the first line of the source, allowing use of non-Unix forms of `#!cmd`. This is intended for a DOS specific hack only.

Note: The line numbers in error messages will be off by one.

-3

Warn about Python 3.x incompatibilities which cannot be fixed trivially by *2to3* (in *The Python Library Reference*). Among these are:

- `dict.has_key()`
- `apply()`
- `callable()`
- `coerce()`
- `execfile()`
- `reduce()`
- `reload()`

Using these will emit a `DeprecationWarning`. New in version 2.6.

1.2 Environment variables

These environment variables influence Python's behavior.

PYTHONHOME

Change the location of the standard Python libraries. By default, the libraries are searched in `prefix/lib/pythonversion` and `exec_prefix/lib/pythonversion`, where `prefix` and `exec_prefix` are installation-dependent directories, both defaulting to `/usr/local`.

When **PYTHONHOME** is set to a single directory, its value replaces both `prefix` and `exec_prefix`. To specify different values for these, set **PYTHONHOME** to `prefix:exec_prefix`.

PYTHONPATH

Augment the default search path for module files. The format is the same as the shell's **PATH**: one or more directory pathnames separated by `os.pathsep` (e.g. colons on Unix or semicolons on Windows). Non-existent directories are silently ignored.

In addition to normal directories, individual **PYTHONPATH** entries may refer to zipfiles containing pure Python modules (in either source or compiled form). Extension modules cannot be imported from zipfiles.

The default search path is installation dependent, but generally begins with `prefix/lib/pythonversion` (see **PYTHONHOME** above). It is *always* appended to **PYTHONPATH**.

An additional directory will be inserted in the search path in front of **PYTHONPATH** as described above under *Interface options*. The search path can be manipulated from within a Python program as the variable `sys.path`.

PYTHONSTARTUP

If this is the name of a readable file, the Python commands in that file are executed before the first prompt is displayed in interactive mode. The file is executed in the same namespace where interactive commands are executed so that objects defined or imported in it can be used without qualification in the interactive session. You can also change the prompts `sys.ps1` and `sys.ps2` in this file.

PYTHONY2K

Set this to a non-empty string to cause the `time` module to require dates specified as strings to include 4-digit years, otherwise 2-digit years are converted based on rules described in the `time` module documentation.

PYTHONOPTIMIZE

If this is set to a non-empty string it is equivalent to specifying the `-O` option. If set to an integer, it is equivalent to specifying `-O` multiple times.

PYTHONDEBUG

If this is set to a non-empty string it is equivalent to specifying the `-d` option. If set to an integer, it is equivalent to specifying `-d` multiple times.

PYTHONINSPECT

If this is set to a non-empty string it is equivalent to specifying the `-i` option.

This variable can also be modified by Python code using `os.environ` to force inspect mode on program termination.

PYTHONUNBUFFERED

If this is set to a non-empty string it is equivalent to specifying the `-u` option.

PYTHONVERBOSE

If this is set to a non-empty string it is equivalent to specifying the `-v` option. If set to an integer, it is equivalent to specifying `-v` multiple times.

PYTHONCASEOK

If this is set, Python ignores case in `import` statements. This only works on Windows.

PYTHONDONTWRITEBYTECODE

If this is set, Python won't try to write `.pyc` or `.pyo` files on the import of source modules. New in version 2.6.

PYTHONIOENCODING

Overrides the encoding used for stdin/stdout/stderr, in the syntax `encodingname:errorhandler`. The `:errorhandler` part is optional and has the same meaning as in `str.encode()`. New in version 2.6.

PYTHONNOUSERSITE

If this is set, Python won't add the user site directory to `sys.path`. New in version 2.6.

See Also:

[PEP 370](#) – Per user site-packages directory

PYTHONUSERBASE

Sets the base directory for the user site directory. New in version 2.6.

See Also:

[PEP 370](#) – Per user site-packages directory

PYTHONEXECUTABLE

If this environment variable is set, `sys.argv[0]` will be set to its value instead of the value got through the C runtime. Only works on Mac OS X.

1.2.1 Debug-mode variables

Setting these variables only has an effect in a debug build of Python, that is, if Python was configured with the `--with-pydebug` build option.

PYTHONTHREADDEBUG

If set, Python will print threading debug info. Changed in version 2.6: Previously, this variable was called `THREADDEBUG`.

PYTHONDUMPPREFS

If set, Python will dump objects and reference counts still alive after shutting down the interpreter.

PYTHONMALLOCSSTATS

If set, Python will print memory allocation statistics every time a new object arena is created, and on shutdown.

USING PYTHON ON UNIX PLATFORMS

2.1 Getting and installing the latest version of Python

2.1.1 On Linux

Python comes preinstalled on most Linux distributions, and is available as a package on all others. However there are certain features you might want to use that are not available on your distro's package. You can easily compile the latest version of Python from source.

In the event that Python doesn't come preinstalled and isn't in the repositories as well, you can easily make packages for your own distro. Have a look at the following links:

See Also:

<http://www.linux.com/articles/60383> for Debian users

<http://linuxmafia.com/pub/linux/suse-linux-internals/chapter35.html> for OpenSuse users

<http://docs.fedoraproject.org/drafts/rpm-guide-en/ch-creating-rpms.html> for Fedora users

<http://www.slackbook.org/html/package-management-making-packages.html> for Slackware users

2.1.2 On FreeBSD and OpenBSD

- FreeBSD users, to add the package use:

```
pkg_add -r python
```

- OpenBSD users use:

```
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/<insert your architecture here>
```

For example i386 users get the 2.5.1 version of Python using:

```
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/i386/python-2.5.1p2.tgz
```

2.1.3 On OpenSolaris

To install the newest Python versions on OpenSolaris, install blastwave (<http://www.blastwave.org/howto.html>) and type “`pkg_get -i python`” at the prompt.

2.2 Building Python

If you want to compile CPython yourself, first thing you should do is get the [source](#). You can download either the latest release's source or just grab a fresh [checkout](#).

The build process consists the usual

```
./configure
make
make install
```

invocations. Configuration options and caveats for specific Unix platforms are extensively documented in the README file in the root of the Python source tree.

Warning: `make install` can overwrite or masquerade the `python` binary. `make altinstall` is therefore recommended instead of `make install` since it only installs `exec_prefix/bin/pythonversion`.

2.3 Python-related paths and files

These are subject to difference depending on local installation conventions; **prefix** (`${prefix}`) and **exec_prefix** (`${exec_prefix}`) are installation-dependent and should be interpreted as for GNU software; they may be the same.

For example, on most Linux systems, the default for both is `/usr`.

File/directory	Meaning
<code>exec_prefix/bin/python</code>	Recommended location of the interpreter.
<code>prefix/lib/pythonversion</code> , <code>exec_prefix/lib/pythonversion</code>	Recommended locations of the directories containing the standard modules.
<code>prefix/include/pythonversion</code> , <code>exec_prefix/include/pythonversion</code>	Recommended locations of the directories containing the include files needed for developing Python extensions and embedding the interpreter.
<code>~/.pythonrc.py</code>	User-specific initialization file loaded by the user module; not used by default or by most applications.

2.4 Miscellaneous

To easily use Python scripts on Unix, you need to make them executable, e.g. with

```
$ chmod +x script
```

and put an appropriate Shebang line at the top of the script. A good choice is usually

```
#!/usr/bin/env python
```

which searches for the Python interpreter in the whole **PATH**. However, some Unices may not have the **env** command, so you may need to hardcode `/usr/bin/python` as the interpreter path.

To use shell commands in your python scripts, look at the `subprocess` module.

2.5 Editors

Vim and Emacs are excellent editors which support Python very well. For more information on how to code in python in these editors, look at:

- http://www.vim.org/scripts/script.php?script_id=790

- <http://sourceforge.net/projects/python-mode>

Geany is an excellent IDE with support for a lot of languages. For more information, read: <http://geany.uvena.de/>

Komodo edit is another extremely good IDE. It also has support for a lot of languages. For more information, read: <http://www.activestate.com/store/productdetail.aspx?prdGuid=20f4ed15-6684-4118-a78b-d37ff4058c5f>

USING PYTHON ON WINDOWS

This document aims to give an overview of Windows-specific behaviour you should know about when using Python on Microsoft Windows.

3.1 Installing Python

Unlike most Unix systems and services, Windows does not require Python natively and thus does not pre-install a version of Python. However, the CPython team has compiled Windows installers (MSI packages) with every [release](#) for many years.

With ongoing development of Python, some platforms that used to be supported earlier are no longer supported (due to the lack of users or developers). Check [PEP 11](#) for details on all unsupported platforms.

- DOS and Windows 3.x are deprecated since Python 2.0 and code specific to these systems was removed in Python 2.1.
- Up to 2.5, Python was still compatible with Windows 95, 98 and ME (but already raised a deprecation warning on installation). For Python 2.6 (and all following releases), this support was dropped and new releases are just expected to work on the Windows NT family.
- [Windows CE](#) is still supported.
- The [Cygwin](#) installer offers to install the [Python interpreter](#) as well; it is located under “Interpreters.” (cf. [Cygwin package source](#), [Maintainer releases](#))

See [Python for Windows \(and DOS\)](#) for detailed information about platforms with precompiled installers.

See Also:

[Python on XP](#) “7 Minutes to “Hello World!”” by Richard Dooling, 2006

[Installing on Windows](#) in “[Dive into Python: Python from novice to pro](#)” by Mark Pilgrim, 2004, ISBN 1-59059-356-1

[For Windows users](#) in “Installing Python” in “[A Byte of Python](#)” by Swaroop C H, 2003

3.2 Alternative bundles

Besides the standard CPython distribution, there are modified packages including additional functionality. The following is a list of popular versions and their key features:

[ActivePython](#) Installer with multi-platform compatibility, documentation, PyWin32

[Python Enthought Edition](#) Popular modules (such as PyWin32) with their respective documentation, tool suite for building extensible python applications

Notice that these packages are likely to install *older* versions of Python.

3.3 Configuring Python

In order to run Python flawlessly, you might have to change certain environment settings in Windows.

3.3.1 Excursus: Setting environment variables

Windows has a built-in dialog for changing environment variables (following guide applies to XP classical view): Right-click the icon for your machine (usually located on your Desktop and called “My Computer”) and choose *Properties* there. Then, open the **Advanced** tab and click the **Environment Variables** button.

In short, your path is:

My Computer → *Properties* → *Advanced* → *Environment Variables*

In this dialog, you can add or modify User and System variables. To change System variables, you need non-restricted access to your machine (i.e. Administrator rights).

Another way of adding variables to your environment is using the **set** command:

```
set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
```

To make this setting permanent, you could add the corresponding command line to your `autoexec.bat`. **msconfig** is a graphical interface to this file.

Viewing environment variables can also be done more straight-forward: The command prompt will expand strings wrapped into percent signs automatically:

```
echo %PATH%
```

Consult **set /?** for details on this behaviour.

See Also:

<http://support.microsoft.com/kb/100843> Environment variables in Windows NT

<http://support.microsoft.com/kb/310519> How To Manage Environment Variables in Windows XP

<http://www.chem.gla.ac.uk/~louis/software/faq/q1.html> Setting Environment variables, Louis J. Farrugia

3.3.2 Finding the Python executable

Besides using the automatically created start menu entry for the Python interpreter, you might want to start Python in the DOS prompt. To make this work, you need to set your **%PATH%** environment variable to include the directory of your Python distribution, delimited by a semicolon from other entries. An example variable could look like this (assuming the first two entries are Windows’ default):

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Python25
```

Typing **python** on your command prompt will now fire up the Python interpreter. Thus, you can also execute your scripts with command line options, see *Command line* documentation.

3.3.3 Finding modules

Python usually stores its library (and thereby your site-packages folder) in the installation directory. So, if you had installed Python to `C:\Python\`, the default library would reside in `C:\Python\Lib\` and third-party modules should be stored in `C:\Python\Lib\site-packages\`.

You can add folders to your search path to make Python’s import mechanism search in these directories as well. Use **PYTHONPATH**, as described in *Environment variables*, to modify `sys.path`. On Windows, paths are separated by semicolons, though, to distinguish them from drive identifiers (`C:\` etc.).

Modifying the module search path can also be done through the Windows registry: Edit `HKEY_LOCAL_MACHINE\SOFTWARE\Python\PythonCore\version\PythonPath\`, as described

above for the environment variable `%PYTHONPATH%`. A convenient registry editor is **regedit** (start it by typing “regedit” into *Start* → *Run*).

3.3.4 Executing scripts

Python scripts (files with the extension `.py`) will be executed by **python.exe** by default. This executable opens a terminal, which stays open even if the program uses a GUI. If you do not want this to happen, use the extension `.pyw` which will cause the script to be executed by **pythonw.exe** by default (both executables are located in the top-level of your Python installation directory). This suppresses the terminal window on startup.

You can also make all `.py` scripts execute with **pythonw.exe**, setting this through the usual facilities, for example (might require administrative rights):

1. Launch a command prompt.
2. Associate the correct file group with `.py` scripts:


```
assoc .py=Python.File
```
3. Redirect all Python files to the new executable:


```
ftype Python.File=C:\Path\to\pythonw.exe "%1" %*
```

3.4 Additional modules

Even though Python aims to be portable among all platforms, there are features that are unique to Windows. A couple of modules, both in the standard library and external, and snippets exist to use these features.

The Windows-specific standard modules are documented in *MS Windows Specific Services* (in *The Python Library Reference*).

3.4.1 PyWin32

The **PyWin32** module by Mark Hammond is a collection of modules for advanced Windows-specific support. This includes utilities for:

- [Component Object Model](#) (COM)
- Win32 API calls
- Registry
- Event log
- [Microsoft Foundation Classes](#) (MFC) user interfaces

PythonWin is a sample MFC application shipped with PyWin32. It is an embeddable IDE with a built-in debugger.

See Also:

Win32 How Do I...? by Tim Golden

Python and COM by David and Paul Boddie

3.4.2 Py2exe

Py2exe is a `distutils` extension (see *Extending Distutils* (in *Distributing Python Modules*)) which wraps Python scripts into executable Windows programs (`*.exe` files). When you have done this, you can distribute your application without requiring your users to install Python.

3.4.3 WConio

Since Python’s advanced terminal handling layer, `curses`, is restricted to Unix-like systems, there is a library exclusive to Windows as well: Windows Console I/O for Python.

`WConio` is a wrapper for Turbo-C’s `CONIO.H`, used to create text user interfaces.

3.5 Compiling Python on Windows

If you want to compile CPython yourself, first thing you should do is get the [source](#). You can download either the latest release’s source or just grab a fresh [checkout](#).

For Microsoft Visual C++, which is the compiler with which official Python releases are built, the source tree contains solutions/project files. View the `readme.txt` in their respective directories:

Directory	MSVC version	Visual Studio version
PC/VC6/	6.0	97
PC/VS7.1/	7.1	2003
PC/VS8.0/	8.0	2005
PCbuild/	9.0	2008

Note that not all of these build directories are fully supported. Read the release notes to see which compiler version the official releases for your version are built with.

Check `PC/readme.txt` for general information on the build process.

For extension modules, consult *Building C and C++ Extensions on Windows* (in *Extending and Embedding Python*).

See Also:

Python + Windows + distutils + SWIG + gcc MinGW or “Creating Python extensions in C/C++ with SWIG and compiling them with MinGW gcc under Windows” or “Installing Python extension with distutils and without Microsoft Visual C++” by Sébastien Sauvage, 2003

MingW – Python extensions by Trent Apted et al, 2007

3.6 Other resources

See Also:

Python Programming On Win32 “Help for Windows Programmers” by Mark Hammond and Andy Robinson, O’Reilly Media, 2000, ISBN 1-56592-621-8

A Python for Windows Tutorial by Amanda Birmingham, 2004

USING PYTHON ON A MACINTOSH

Author Bob Savage <bobsavage@mac.com>

Python on a Macintosh running Mac OS X is in principle very similar to Python on any other Unix platform, but there are a number of additional features such as the IDE and the Package Manager that are worth pointing out.

The Mac-specific modules are documented in *Mac OS X specific services* (in *The Python Library Reference*).

Python on Mac OS 9 or earlier can be quite different from Python on Unix or Windows, but is beyond the scope of this manual, as that platform is no longer supported, starting with Python 2.4. See <http://www.cwi.nl/~jack/macpython> for installers for the latest 2.3 release for Mac OS 9 and related documentation.

4.1 Getting and Installing MacPython

Mac OS X 10.5 comes with Python 2.5.1 pre-installed by Apple. If you wish, you are invited to install the most recent version of Python from the Python website (<http://www.python.org>). A current “universal binary” build of Python, which runs natively on the Mac’s new Intel and legacy PPC CPU’s, is available there.

What you get after installing is a number of things:

- A MacPython 2.5 folder in your Applications folder. In here you find IDLE, the development environment that is a standard part of official Python distributions; PythonLauncher, which handles double-clicking Python scripts from the Finder; and the “Build Applet” tool, which allows you to package Python scripts as standalone applications on your system.
- A framework `/Library/Frameworks/Python.framework`, which includes the Python executable and libraries. The installer adds this location to your shell path. To uninstall MacPython, you can simply remove these three things. A symlink to the Python executable is placed in `/usr/local/bin/`.

The Apple-provided build of Python is installed in `/System/Library/Frameworks/Python.framework` and `/usr/bin/python`, respectively. You should never modify or delete these, as they are Apple-controlled and are used by Apple- or third-party software. Remember that if you choose to install a newer Python version from `python.org`, you will have two different but functional Python installations on your computer, so it will be important that your paths and usages are consistent with what you want to do.

IDLE includes a help menu that allows you to access Python documentation. If you are completely new to Python you should start reading the tutorial introduction in that document.

If you are familiar with Python on other Unix platforms you should read the section on running Python scripts from the Unix shell.

4.1.1 How to run a Python script

Your best way to get started with Python on Mac OS X is through the IDLE integrated development environment, see section *The IDE* and use the Help menu when the IDE is running.

If you want to run Python scripts from the Terminal window command line or from the Finder you first need an editor to create your script. Mac OS X comes with a number of standard Unix command line editors, **vim** and **emacs** among them. If you want a more Mac-like editor, **BBEdit** or **TextWrangler** from Bare Bones Software (see <http://www.barebones.com/products/bbedit/index.shtml>) are good choices, as is **TextMate** (see <http://macromates.com/>). Other editors include **Gvim** (<http://macvim.org>) and **Aquamacs** (<http://aquamacs.org>).

To run your script from the Terminal window you must make sure that `/usr/local/bin` is in your shell search path.

To run your script from the Finder you have two options:

- Drag it to **PythonLauncher**
- Select **PythonLauncher** as the default application to open your script (or any `.py` script) through the finder Info window and double-click it. **PythonLauncher** has various preferences to control how your script is launched. Option-dragging allows you to change these for one invocation, or use its Preferences menu to change things globally.

4.1.2 Running scripts with a GUI

With older versions of Python, there is one Mac OS X quirk that you need to be aware of: programs that talk to the Aqua window manager (in other words, anything that has a GUI) need to be run in a special way. Use **pythonw** instead of **python** to start such scripts.

With Python 2.5, you can use either **python** or **pythonw**.

4.1.3 Configuration

Python on OS X honors all standard Unix environment variables such as **PYTHONPATH**, but setting these variables for programs started from the Finder is non-standard as the Finder does not read your `.profile` or `.cshrc` at startup. You need to create a file `~/.MacOSX/environment.plist`. See Apple's Technical Document QA1067 for details.

For more information on installation Python packages in MacPython, see section *Installing Additional Python Packages*.

4.2 The IDE

MacPython ships with the standard IDLE development environment. A good introduction to using IDLE can be found at http://hkn.eecs.berkeley.edu/dyoo/python/idle_intro/index.html.

4.3 Installing Additional Python Packages

There are several methods to install additional Python packages:

- <http://pythonmac.org/packages/> contains selected compiled packages for Python 2.5, 2.4, and 2.3.
- Packages can be installed via the standard Python distutils mode (`python setup.py install`).
- Many packages can also be installed via the **setuptools** extension.

4.4 GUI Programming on the Mac

There are several options for building GUI applications on the Mac with Python.

PyObjC is a Python binding to Apple's Objective-C/Cocoa framework, which is the foundation of most modern Mac development. Information on PyObjC is available from <http://pyobjc.sourceforge.net>.

The standard Python GUI toolkit is *Tkinter*, based on the cross-platform Tk toolkit (<http://www.tcl.tk>). An Aqua-native version of Tk is bundled with OS X by Apple, and the latest version can be downloaded and installed from <http://www.activestate.com>; it can also be built from source.

wxPython is another popular cross-platform GUI toolkit that runs natively on Mac OS X. Packages and documentation are available from <http://www.wxpython.org>.

PyQt is another popular cross-platform GUI toolkit that runs natively on Mac OS X. More information can be found at <http://www.riverbankcomputing.co.uk/software/pyqt/intro>.

4.5 Distributing Python Applications on the Mac

The “Build Applet” tool that is placed in the MacPython 2.5 folder is fine for packaging small Python scripts on your own machine to run as a standard Mac application. This tool, however, is not robust enough to distribute Python applications to other users.

The standard tool for deploying standalone Python applications on the Mac is **py2app**. More information on installing and using py2app can be found at <http://undefined.org/python/#py2app>.

4.6 Application Scripting

Python can also be used to script other Mac applications via Apple's Open Scripting Architecture (OSA); see <http://appscript.sourceforge.net>. Appscript is a high-level, user-friendly Apple event bridge that allows you to control scriptable Mac OS X applications using ordinary Python scripts. Appscript makes Python a serious alternative to Apple's own *AppleScript* language for automating your Mac. A related package, *PyOSA*, is an OSA language component for the Python scripting language, allowing Python code to be executed by any OSA-enabled application (Script Editor, Mail, iTunes, etc.). PyOSA makes Python a full peer to AppleScript.

4.7 Other Resources

The MacPython mailing list is an excellent support resource for Python users and developers on the Mac:

<http://www.python.org/community/sigs/current/pythonmac-sig/>

Another useful resource is the MacPython wiki:

<http://wiki.python.org/moin/MacPython>

GLOSSARY

>>> The default Python prompt of the interactive shell. Often seen for code examples which can be executed interactively in the interpreter.

... The default Python prompt of the interactive shell when entering code for an indented code block or within a pair of matching left and right delimiters (parentheses, square brackets or curly braces).

2to3 A tool that tries to convert Python 2.x code to Python 3.x code by handling most of the incompatibilities which can be detected by parsing the source and traversing the parse tree.

2to3 is available in the standard library as `lib2to3`; a standalone entry point is provided as `Tools/scripts/2to3`. See *2to3 - Automated Python 2 to 3 code translation* (in *The Python Library Reference*).

abstract base class Abstract Base Classes (abbreviated ABCs) complement *duck-typing* by providing a way to define interfaces when other techniques like `hasattr()` would be clumsy. Python comes with many builtin ABCs for data structures (in the `collections` module), numbers (in the `numbers` module), and streams (in the `io` module). You can create your own ABC with the `abc` module.

argument A value passed to a function or method, assigned to a named local variable in the function body. A function or method may have both positional arguments and keyword arguments in its definition. Positional and keyword arguments may be variable-length: `*` accepts or passes (if in the function definition or call) several positional arguments in a list, while `**` does the same for keyword arguments in a dictionary.

Any expression may be used within the argument list, and the evaluated value is passed to the local variable.

attribute A value associated with an object which is referenced by name using dotted expressions. For example, if an object *o* has an attribute *a* it would be referenced as *o.a*.

BDFL Benevolent Dictator For Life, a.k.a. [Guido van Rossum](#), Python's creator.

bytecode Python source code is compiled into bytecode, the internal representation of a Python program in the interpreter. The bytecode is also cached in `.pyc` and `.pyo` files so that executing the same file is faster the second time (recompilation from source to bytecode can be avoided). This “intermediate language” is said to run on a *virtual machine* that executes the machine code corresponding to each bytecode.

class A template for creating user-defined objects. Class definitions normally contain method definitions which operate on instances of the class.

classic class Any class which does not inherit from `object`. See *new-style class*. Classic classes will be removed in Python 3.0.

coercion The implicit conversion of an instance of one type to another during an operation which involves two arguments of the same type. For example, `int(3.15)` converts the floating point number to the integer 3, but in `3+4.5`, each argument is of a different type (one int, one float), and both must be converted to the same type before they can be added or it will raise a `TypeError`. Coercion between two operands can be performed with the `coerce` builtin function; thus, `3+4.5` is equivalent to calling `operator.add(*coerce(3, 4.5))` and results in `operator.add(3.0, 4.5)`. Without coercion, all arguments of even compatible types would have to be normalized to the same value by the programmer, e.g., `float(3)+4.5` rather than just `3+4.5`.

complex number An extension of the familiar real number system in which all numbers are expressed as a sum of a real part and an imaginary part. Imaginary numbers are real multiples of the imaginary unit (the square root of -1), often written i in mathematics or j in engineering. Python has builtin support for complex numbers, which are written with this latter notation; the imaginary part is written with a j suffix, e.g., $3+1j$. To get access to complex equivalents of the `math` module, use `cmath`. Use of complex numbers is a fairly advanced mathematical feature. If you're not aware of a need for them, it's almost certain you can safely ignore them.

context manager An object which controls the environment seen in a `with` statement by defining `__enter__()` and `__exit__()` methods. See [PEP 343](#).

CPython The canonical implementation of the Python programming language. The term “CPython” is used in contexts when necessary to distinguish this implementation from others such as Jython or IronPython.

decorator A function returning another function, usually applied as a function transformation using the `@wrapper` syntax. Common examples for decorators are `classmethod()` and `staticmethod()`.

The decorator syntax is merely syntactic sugar, the following two function definitions are semantically equivalent:

```
def f(...):
    ...
f = staticmethod(f)

@staticmethod
def f(...):
    ...
```

See *the documentation for function definition* (in *The Python Language Reference*) for more about decorators.

descriptor Any *new-style* object which defines the methods `__get__()`, `__set__()`, or `__delete__()`. When a class attribute is a descriptor, its special binding behavior is triggered upon attribute lookup. Normally, using `a.b` to get, set or delete an attribute looks up the object named `b` in the class dictionary for `a`, but if `b` is a descriptor, the respective descriptor method gets called. Understanding descriptors is a key to a deep understanding of Python because they are the basis for many features including functions, methods, properties, class methods, static methods, and reference to super classes.

For more information about descriptors' methods, see *Implementing Descriptors* (in *The Python Language Reference*).

dictionary An associative array, where arbitrary keys are mapped to values. The use of `dict` closely resembles that for `list`, but the keys can be any object with a `__hash__()` function, not just integers. Called a hash in Perl.

docstring A string literal which appears as the first expression in a class, function or module. While ignored when the suite is executed, it is recognized by the compiler and put into the `__doc__` attribute of the enclosing class, function or module. Since it is available via introspection, it is the canonical place for documentation of the object.

duck-typing A pythonic programming style which determines an object's type by inspection of its method or attribute signature rather than by explicit relationship to some type object (“If it looks like a duck and quacks like a duck, it must be a duck.”) By emphasizing interfaces rather than specific types, well-designed code improves its flexibility by allowing polymorphic substitution. Duck-typing avoids tests using `type()` or `isinstance()`. (Note, however, that duck-typing can be complemented with abstract base classes.) Instead, it typically employs `hasattr()` tests or [EAFP](#) programming.

EAFP Easier to ask for forgiveness than permission. This common Python coding style assumes the existence of valid keys or attributes and catches exceptions if the assumption proves false. This clean and fast style is characterized by the presence of many `try` and `except` statements. The technique contrasts with the [LBYL](#) style common to many other languages such as C.

expression A piece of syntax which can be evaluated to some value. In other words, an expression is an accumulation of expression elements like literals, names, attribute access, operators or function calls which all return a value. In contrast to many other languages, not all language constructs are expressions. There

are also *statements* which cannot be used as expressions, such as `print` or `if`. Assignments are also statements, not expressions.

extension module A module written in C or C++, using Python's C API to interact with the core and with user code.

finder An object that tries to find the *loader* for a module. It must implement a method named `find_module()`. See [PEP 302](#) for details.

function A series of statements which returns some value to a caller. It can also be passed zero or more arguments which may be used in the execution of the body. See also *argument* and *method*.

__future__ A pseudo module which programmers can use to enable new language features which are not compatible with the current interpreter. For example, the expression `11/4` currently evaluates to 2. If the module in which it is executed had enabled *true division* by executing:

```
from __future__ import division
```

the expression `11/4` would evaluate to 2.75. By importing the `__future__` module and evaluating its variables, you can see when a new feature was first added to the language and when it will become the default:

```
>>> import __future__
>>> __future__.division
_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192)
```

garbage collection The process of freeing memory when it is not used anymore. Python performs garbage collection via reference counting and a cyclic garbage collector that is able to detect and break reference cycles.

generator A function which returns an iterator. It looks like a normal function except that values are returned to the caller using a `yield` statement instead of a `return` statement. Generator functions often contain one or more `for` or `while` loops which `yield` elements back to the caller. The function execution is stopped at the `yield` keyword (returning the result) and is resumed there when the next element is requested by calling the `next()` method of the returned iterator.

generator expression An expression that returns a generator. It looks like a normal expression followed by a `for` expression defining a loop variable, range, and an optional `if` expression. The combined expression generates values for an enclosing function:

```
>>> sum(i*i for i in range(10))           # sum of squares 0, 1, 4, ... 81
285
```

GIL See *global interpreter lock*.

global interpreter lock The lock used by Python threads to assure that only one thread executes in the *CPython virtual machine* at a time. This simplifies the CPython implementation by assuring that no two processes can access the same memory at the same time. Locking the entire interpreter makes it easier for the interpreter to be multi-threaded, at the expense of much of the parallelism afforded by multi-processor machines. Efforts have been made in the past to create a “free-threaded” interpreter (one which locks shared data at a much finer granularity), but so far none have been successful because performance suffered in the common single-processor case.

hashable An object is *hashable* if it has a hash value which never changes during its lifetime (it needs a `__hash__()` method), and can be compared to other objects (it needs an `__eq__()` or `__cmp__()` method). Hashable objects which compare equal must have the same hash value.

Hashability makes an object usable as a dictionary key and a set member, because these data structures use the hash value internally.

All of Python's immutable built-in objects are hashable, while no mutable containers (such as lists or dictionaries) are. Objects which are instances of user-defined classes are hashable by default; they all compare unequal, and their hash value is their `id()`.

IDLE An Integrated Development Environment for Python. IDLE is a basic editor and interpreter environment which ships with the standard distribution of Python. Good for beginners, it also serves as clear example code for those wanting to implement a moderately sophisticated, multi-platform GUI application.

immutable An object with a fixed value. Immutable objects include numbers, strings and tuples. Such an object cannot be altered. A new object has to be created if a different value has to be stored. They play an important role in places where a constant hash value is needed, for example as a key in a dictionary.

integer division Mathematical division discarding any remainder. For example, the expression `11 / 4` currently evaluates to 2 in contrast to the `2.75` returned by float division. Also called *floor division*. When dividing two integers the outcome will always be another integer (having the floor function applied to it). However, if one of the operands is another numeric type (such as a `float`), the result will be coerced (see *coercion*) to a common type. For example, an integer divided by a float will result in a float value, possibly with a decimal fraction. Integer division can be forced by using the `//` operator instead of the `/` operator. See also *__future__*.

importer An object that both finds and loads a module; both a *finder* and *loader* object.

interactive Python has an interactive interpreter which means you can enter statements and expressions at the interpreter prompt, immediately execute them and see their results. Just launch `python` with no arguments (possibly by selecting it from your computer's main menu). It is a very powerful way to test out new ideas or inspect modules and packages (remember `help(x)`).

interpreted Python is an interpreted language, as opposed to a compiled one, though the distinction can be blurry because of the presence of the bytecode compiler. This means that source files can be run directly without explicitly creating an executable which is then run. Interpreted languages typically have a shorter development/debug cycle than compiled ones, though their programs generally also run more slowly. See also *interactive*.

iterable A container object capable of returning its members one at a time. Examples of iterables include all sequence types (such as `list`, `str`, and `tuple`) and some non-sequence types like `dict` and `file` and objects of any classes you define with an `__iter__()` or `__getitem__()` method. Iterables can be used in a `for` loop and in many other places where a sequence is needed (`zip()`, `map()`, ...). When an iterable object is passed as an argument to the builtin function `iter()`, it returns an iterator for the object. This iterator is good for one pass over the set of values. When using iterables, it is usually not necessary to call `iter()` or deal with iterator objects yourself. The `for` statement does that automatically for you, creating a temporary unnamed variable to hold the iterator for the duration of the loop. See also *iterator*, *sequence*, and *generator*.

iterator An object representing a stream of data. Repeated calls to the iterator's `next()` method return successive items in the stream. When no more data are available a `StopIteration` exception is raised instead. At this point, the iterator object is exhausted and any further calls to its `next()` method just raise `StopIteration` again. Iterators are required to have an `__iter__()` method that returns the iterator object itself so every iterator is also iterable and may be used in most places where other iterables are accepted. One notable exception is code which attempts multiple iteration passes. A container object (such as a `list`) produces a fresh new iterator each time you pass it to the `iter()` function or use it in a `for` loop. Attempting this with an iterator will just return the same exhausted iterator object used in the previous iteration pass, making it appear like an empty container.

More information can be found in *Iterator Types* (in *The Python Library Reference*).

keyword argument Arguments which are preceded with a `variable_name=` in the call. The variable name designates the local name in the function to which the value is assigned. `**` is used to accept or pass a dictionary of keyword arguments. See *argument*.

lambda An anonymous inline function consisting of a single *expression* which is evaluated when the function is called. The syntax to create a lambda function is `lambda [arguments]: expression`

LBYL Look before you leap. This coding style explicitly tests for pre-conditions before making calls or lookups. This style contrasts with the *EAFP* approach and is characterized by the presence of many `if` statements.

list A built-in Python *sequence*. Despite its name it is more akin to an array in other languages than to a linked list since access to elements are $O(1)$.

list comprehension A compact way to process all or part of the elements in a sequence and return a list with the results. `result = ["0x%02x" % x for x in range(256) if x % 2 == 0]` generates a list of strings containing even hex numbers (0x..) in the range from 0 to 255. The `if` clause is optional. If omitted, all elements in `range(256)` are processed.

loader An object that loads a module. It must define a method named `load_module()`. A loader is typically returned by a *finder*. See [PEP 302](#) for details.

mapping A container object (such as `dict`) which supports arbitrary key lookups using the special method `__getitem__()`.

metaclass The class of a class. Class definitions create a class name, a class dictionary, and a list of base classes. The metaclass is responsible for taking those three arguments and creating the class. Most object oriented programming languages provide a default implementation. What makes Python special is that it is possible to create custom metaclasses. Most users never need this tool, but when the need arises, metaclasses can provide powerful, elegant solutions. They have been used for logging attribute access, adding thread-safety, tracking object creation, implementing singletons, and many other tasks.

More information can be found in *Customizing class creation* (in *The Python Language Reference*).

method A function which is defined inside a class body. If called as an attribute of an instance of that class, the method will get the instance object as its first *argument* (which is usually called `self`). See *function* and *nested scope*.

mutable Mutable objects can change their value but keep their `id()`. See also *immutable*.

named tuple Any tuple-like class whose indexable elements are also accessible using named attributes (for example, `time.localtime()` returns a tuple-like object where the *year* is accessible either with an index such as `t[0]` or with a named attribute like `t.tm_year`).

A named tuple can be a built-in type such as `time.struct_time`, or it can be created with a regular class definition. A full featured named tuple can also be created with the factory function `collections.namedtuple()`. The latter approach automatically provides extra features such as a self-documenting representation like `Employee(name='jones', title='programmer')`.

namespace The place where a variable is stored. Namespaces are implemented as dictionaries. There are the local, global and builtin namespaces as well as nested namespaces in objects (in methods). Namespaces support modularity by preventing naming conflicts. For instance, the functions `__builtin__.open()` and `os.open()` are distinguished by their namespaces. Namespaces also aid readability and maintainability by making it clear which module implements a function. For instance, writing `random.seed()` or `itertools.izip()` makes it clear that those functions are implemented by the `random` and `itertools` modules, respectively.

nested scope The ability to refer to a variable in an enclosing definition. For instance, a function defined inside another function can refer to variables in the outer function. Note that nested scopes work only for reference and not for assignment which will always write to the innermost scope. In contrast, local variables both read and write in the innermost scope. Likewise, global variables read and write to the global namespace.

new-style class Any class which inherits from `object`. This includes all built-in types like `list` and `dict`. Only new-style classes can use Python's newer, versatile features like `__slots__`, descriptors, properties, and `__getattr__()`.

More information can be found in *New-style and classic classes* (in *The Python Language Reference*).

object Any data with state (attributes or value) and defined behavior (methods). Also the ultimate base class of any *new-style class*.

positional argument The arguments assigned to local names inside a function or method, determined by the order in which they were given in the call. `*` is used to either accept multiple positional arguments (when in the definition), or pass several arguments as a list to a function. See *argument*.

Python 3000 Nickname for the next major Python version, 3.0 (coined long ago when the release of version 3 was something in the distant future.) This is also abbreviated “Py3k”.

Pythonic An idea or piece of code which closely follows the most common idioms of the Python language, rather than implementing code using concepts common to other languages. For example, a common idiom in Python is to loop over all elements of an iterable using a `for` statement. Many other languages don't have this type of construct, so people unfamiliar with Python sometimes use a numerical counter instead:

```
for i in range(len(food)):
    print food[i]
```

As opposed to the cleaner, Pythonic method:

```
for piece in food:
    print piece
```

reference count The number of references to an object. When the reference count of an object drops to zero, it is deallocated. Reference counting is generally not visible to Python code, but it is a key element of the *CPython* implementation. The `sys` module defines a `getrefcount()` function that programmers can call to return the reference count for a particular object.

__slots__ A declaration inside a *new-style class* that saves memory by pre-declaring space for instance attributes and eliminating instance dictionaries. Though popular, the technique is somewhat tricky to get right and is best reserved for rare cases where there are large numbers of instances in a memory-critical application.

sequence An *iterable* which supports efficient element access using integer indices via the `__getitem__()` special method and defines a `len()` method that returns the length of the sequence. Some built-in sequence types are `list`, `str`, `tuple`, and `unicode`. Note that `dict` also supports `__getitem__()` and `__len__()`, but is considered a mapping rather than a sequence because the lookups use arbitrary *immutable* keys rather than integers.

slice An object usually containing a portion of a *sequence*. A slice is created using the subscript notation, `[]` with colons between numbers when several are given, such as in `variable_name[1:3:5]`. The bracket (subscript) notation uses `slice` objects internally (or in older versions, `__getslice__()` and `__setslice__()`).

special method A method that is called implicitly by Python to execute a certain operation on a type, such as addition. Such methods have names starting and ending with double underscores. Special methods are documented in *Special method names* (in *The Python Language Reference*).

statement A statement is part of a suite (a “block” of code). A statement is either an *expression* or a one of several constructs with a keyword, such as `if`, `while` or `print`.

triple-quoted string A string which is bound by three instances of either a quotation mark (") or an apostrophe ('). While they don't provide any functionality not available with single-quoted strings, they are useful for a number of reasons. They allow you to include unescaped single and double quotes within a string and they can span multiple lines without the use of the continuation character, making them especially useful when writing docstrings.

type The type of a Python object determines what kind of object it is; every object has a type. An object's type is accessible as its `__class__` attribute or can be retrieved with `type(obj)`.

virtual machine A computer defined entirely in software. Python's virtual machine executes the *bytecode* emitted by the bytecode compiler.

Zen of Python Listing of Python design principles and philosophies that are helpful in understanding and using the language. The listing can be found by typing `import this` at the interactive prompt.

ABOUT THESE DOCUMENTS

These documents are generated from [reStructuredText](#) sources by *Sphinx*, a document processor specifically written for the Python documentation.

Development of the documentation and its toolchain takes place on the docs@python.org mailing list. We're always looking for volunteers wanting to help with the docs, so feel free to send a mail there!

Many thanks go to:

- Fred L. Drake, Jr., the creator of the original Python documentation toolset and writer of much of the content;
- the [Docutils](#) project for creating reStructuredText and the Docutils suite;
- Fredrik Lundh for his [Alternative Python Reference](#) project from which Sphinx got many good ideas.

See *Reporting Bugs in Python* for information how to report bugs in this documentation, or Python itself.

B.1 Contributors to the Python Documentation

This section lists people who have contributed in some way to the Python documentation. It is probably not complete – if you feel that you or anyone else should be on this list, please let us know (send email to docs@python.org), and we'll be glad to correct the problem.

Aahz, Michael Abbott, Steve Alexander, Jim Ahlstrom, Fred Allen, A. Amoroso, Pehr Anderson, Oliver Andrich, Heidi Annexstad, Jesús Cea Avi3n, Daniel Barclay, Chris Barker, Don Bashford, Anthony Baxter, Alexander Belopolsky, Bennett Benson, Jonathan Black, Robin Boerdijk, Michal Bozon, Aaron Brancotti, Georg Brandl, Keith Briggs, Ian Bruntlett, Lee Busby, Lorenzo M. Catucci, Carl Cerecke, Mauro Cicognini, Gilles Civario, Mike Clarkson, Steve Clift, Dave Cole, Matthew Cowles, Jeremy Craven, Andrew Dalke, Ben Darnell, L. Peter Deutsch, Robert Donohue, Fred L. Drake, Jr., Josip Dzolong, Jeff Epler, Michael Ernst, Blame Andy Eskilsson, Carey Evans, Martijn Faassen, Carl Feynman, Dan Finnie, Hern3n Mart3nez Foffani, Stefan Franke, Jim Fulton, Peter Funk, Lele Gaifax, Matthew Gallagher, Gabriel Genellina, Ben Gertzfield, Nadim Ghaznavi, Jonathan Giddy, Shelley Gooch, Nathaniel Gray, Grant Griffin, Thomas Guettler, Anders Hammarquist, Mark Hammond, Harald Hanche-Olsen, Manus Hand, Gerhard H3ring, Travis B. Hartwell, Tim Hatch, Janko Hauser, Thomas Heller, Bernhard Herzog, Magnus L. Hetland, Konrad Hinsén, Stefan Hoffmeister, Albert Hofkamp, Gregor Hofleit, Steve Holden, Thomas Holenstein, Gerrit Holl, Rob Hooft, Brian Hooper, Randall Hopper, Michael Hudson, Eric Huss, Jeremy Hylton, Roger Irwin, Jack Jansen, Philip H. Jensen, Pedro Diaz Jimenez, Kent Johnson, Lucas de Jonge, Andreas Jung, Robert Kern, Jim Kerr, Jan Kim, Greg Kochanski, Guido Kollerie, Peter A. Koren, Daniel Kozan, Andrew M. Kuchling, Dave Kuhlman, Erno Kuusela, Thomas Lamb, Detlef Lannert, Piers Lauder, Glyph Lefkowitz, Robert Lehmann, Marc-André Lemburg, Ross Light, Ulf A. Lindgren, Everett Lipman, Mirko Liss, Martin von L3wis, Fredrik Lundh, Jeff MacDonald, John Machin, Andrew MacIntyre, Vladimir Marangozov, Vincent Marchetti, Laura Matson, Daniel May, Rebecca McCreary, Doug Mennella, Paolo Milani, Skip Montanaro, Paul Moore, Ross Moore, Sjoerd Mullender, Dale Nagata, Ng Pheng Siong, Koray Oner, Tomas Oppelstrup, Denis S. Otkidach, Zooko O'Whielacronx, Shriphani Palakodety, William Park, Joonas Paalasmaa, Harri Pasanen, Bo Peng, Tim Peters, Benjamin Peterson, Christopher Petrilli, Justin D. Pettit, Chris Phoenix, Fran3ois Pinard, Paul Prescod, Eric S. Raymond, Edward K. Ream, Sean Reifschneider, Bernhard Reiter, Armin Rigo, Wes Rishel, Armin Ronacher, Jim Roskind, Guido van Rossum, Donald Wallace Rouse II, Mark Russell, Nick Russo, Chris

Ryland, Constantina S., Hugh Sasse, Bob Savage, Scott Schram, Neil Schemenauer, Barry Scott, Joakim Sernbrant, Justin Sheehy, Charlie Shepherd, Michael Simcich, Ionel Simionescu, Michael Sloan, Gregory P. Smith, Roy Smith, Clay Spence, Nicholas Spies, Tage Stabell-Kulo, Frank Stajano, Anthony Starks, Greg Stein, Peter Stoehr, Mark Summerfield, Reuben Sumner, Kalle Svensson, Jim Tittsler, David Turner, Ville Vainio, Martijn Vries, Charles G. Waldman, Greg Ward, Barry Warsaw, Corran Webster, Glyn Webster, Bob Weiner, Eddy Welbourne, Jeff Wheeler, Mats Wichmann, Gerry Wiener, Timothy Wild, Collin Winter, Blake Winton, Dan Wolfe, Steven Work, Thomas Wouters, Ka-Ping Yee, Rory Yorke, Moshe Zadka, Milan Zamazal, Cheng Zhang.

It is only with the input and contributions of the Python community that Python has such wonderful documentation – Thank You!

HISTORY AND LICENSE

C.1 History of the software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see <http://www.cwi.nl/>) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see <http://www.cnri.reston.va.us/>) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen Python-Labs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see <http://www.zope.com/>). In 2001, the Python Software Foundation (PSF, see <http://www.python.org/psf/>) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see <http://www.opensource.org/> for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Release	Derived from	Year	Owner	GPL compatible?
0.9.0 thru 1.2	n/a	1991-1995	CWI	yes
1.3 thru 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	no
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.2	2.1.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2.1	2.2	2002	PSF	yes
2.2.2	2.2.1	2002	PSF	yes
2.2.3	2.2.2	2002-2003	PSF	yes
2.3	2.2.2	2002-2003	PSF	yes
2.3.1	2.3	2002-2003	PSF	yes
2.3.2	2.3.1	2003	PSF	yes
2.3.3	2.3.2	2003	PSF	yes
2.3.4	2.3.3	2004	PSF	yes
2.3.5	2.3.4	2005	PSF	yes
2.4	2.3	2004	PSF	yes
2.4.1	2.4	2005	PSF	yes
2.4.2	2.4.1	2005	PSF	yes
2.4.3	2.4.2	2006	PSF	yes

Continued on next page

Table C.1 – continued from previous page

2.4.4	2.4.3	2006	PSF	yes
2.5	2.4	2006	PSF	yes
2.5.1	2.5	2007	PSF	yes
2.5.2	2.5.1	2008	PSF	yes
2.5.3	2.5.2	2008	PSF	yes
2.6	2.5	2008	PSF	yes
2.6.1	2.6	2008	PSF	yes

Note: GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

C.2 Terms and conditions for accessing or otherwise using Python

PSF LICENSE AGREEMENT FOR PYTHON 2.6.3

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.6.3 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.6.3 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2009 Python Software Foundation; All Rights Reserved" are retained in Python 2.6.3 alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 2.6.3 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.6.3.
4. PSF is making Python 2.6.3 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 2.6.3 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.6.3 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.6.3, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python 2.6.3, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0 BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.
3. BeOpen is making the Software available to Licensee on an “AS IS” basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the “BeOpen Python” logos available at <http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.
7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 (“CNRI”), and the Individual or Organization (“Licensee”) accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI’s License Agreement and CNRI’s notice of copyright, i.e., “Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved” are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI’s License Agreement, Licensee may substitute the following text (omitting the quotes): “Python 1.6.1 is made available subject to the terms and conditions in CNRI’s License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: <http://hdl.handle.net/1895.22/1013>.”
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.
4. CNRI is making Python 1.6.1 available to Licensee on an “AS IS” basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2 Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

C.3 Licenses and Acknowledgements for Incorporated Software

This section is an incomplete, but growing list of licenses and acknowledgements for third-party software incorporated in the Python distribution.

C.3.1 Mersenne Twister

The `_random` module includes code based on a download from <http://www.math.keio.ac.jp/matumoto/MT2002/emt19937ar.html>. The following are the verbatim comments from the original code:

```
A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.
```

```
Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).
```

```
Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Any feedback is very welcome.

<http://www.math.keio.ac.jp/matumoto/emt.html>

email: matumoto@math.keio.ac.jp

C.3.2 Sockets

The socket module uses the functions, `getaddrinfo()`, and `getnameinfo()`, which are coded in separate source files from the WIDE Project, <http://www.wide.ad.jp/>.

Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS'' AND GAI_ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE FOR GAI_ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON GAI_ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN GAI_ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

L. Peter Deutsch
ghost@aladdin.com

Independent implementation of MD5 (RFC 1321).

This code implements the MD5 Algorithm defined in RFC 1321, whose text is available at

<http://www.ietf.org/rfc/rfc1321.txt>

The code is derived from the text of the RFC, including the test suite (section A.5) but excluding the rest of Appendix A. It does not include any code or documentation that is identified in the RFC as being copyrighted.

The original and principal author of md5.h is L. Peter Deutsch <ghost@aladdin.com>. Other authors are noted in the change history that follows (in reverse chronological order):

2002-04-13 lpd Removed support for non-ANSI compilers; removed references to Ghostscript; clarified derivation from RFC 1321; now handles byte order either statically or dynamically.
1999-11-04 lpd Edited comments slightly for automatic TOC extraction.
1999-10-18 lpd Fixed typo in header comment (ansi2knr rather than md5); added conditionalization for C++ compilation from Martin Purschke <purschke@bnl.gov>.
1999-05-03 lpd Original version.

C.3.5 Asynchronous socket services

The `asynchat` and `asyncore` modules contain the following notice:

Copyright 1996 by Sam Rushing

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Sam Rushing not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

C.3.6 Cookie management

The Cookie module contains the following notice:

Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Timothy O'Malley not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL Timothy O'Malley BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

C.3.7 Profiling

The profile and pstats modules contain the following notice:

Copyright 1994, by InfoSeek Corporation, all rights reserved.

Written by James Roskind

Permission to use, copy, modify, and distribute this Python software and its associated documentation for any purpose (subject to the restriction in the following sentence) without fee is hereby granted, provided that the above copyright notice appears in all copies, and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of InfoSeek not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. This permission is explicitly restricted to the copying and modification of the software to remain in Python, compiled Python, or other languages (such as C) wherein the modified or derived code is exclusively imported into a Python module.

INFOSEEK CORPORATION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL INFOSEEK CORPORATION BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

C.3.8 Execution tracing

The trace module contains the following notice:

portions copyright 2001, Autonomous Zones Industries, Inc., all rights...
err... reserved and offered to the public under the terms of the
Python 2.2 license.

Author: Zooko O'Whielacronx

<http://zooko.com/>

<mailto:zooko@zooko.com>

Copyright 2000, Mojam Media, Inc., all rights reserved.

Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.

Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserved.

Author: Skip Montanaro

Copyright 1991-1995, Stichting Mathematisch Centrum, all rights reserved.

Permission to use, copy, modify, and distribute this Python software and
its associated documentation for any purpose without fee is hereby
granted, provided that the above copyright notice appears in all copies,
and that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of neither Automatrix,
Bioreason or Mojam Media be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission.

C.3.9 UUencode and UUdecode functions

The uu module contains the following notice:

Copyright 1994 by Lance Ellinghouse

Cathedral City, California Republic, United States of America.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Lance Ellinghouse
not be used in advertising or publicity pertaining to distribution
of the software without specific, written prior permission.

LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Modified by Jack Jansen, CWI, July 1995:

- Use binascii module to do the actual line-by-line conversion
between ascii and binary. This results in a 1000-fold speedup. The C
version is still 5 times faster, though.
- Arguments more compliant with python standard

C.3.10 XML Remote Procedure Calls

The `xmlrpclib` module contains the following notice:

The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its associated documentation, you agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its associated documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies, and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Secret Labs AB or the author not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

C.3.11 `test_epoll`

The `test_epoll` contains the following notice:

Copyright (c) 2001-2006 Twisted Matrix Laboratories.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

C.3.12 Select kqueue

The select and contains the following notice for the kqueue interface:

Copyright (c) 2000 Doug White, 2006 James Knight, 2007 Christian Heimes
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

COPYRIGHT

Python and this documentation is:

Copyright © 2001-2008 Python Software Foundation. All rights reserved.

Copyright © 2000 BeOpen.com. All rights reserved.

Copyright © 1995-2000 Corporation for National Research Initiatives. All rights reserved.

Copyright © 1991-1995 Stichting Mathematisch Centrum. All rights reserved.

See *[History and License](#)* for complete license and permissions information.

INDEX

Symbols

-help
 command line option, 4
-version
 command line option, 4
-3
 command line option, 6
-B
 command line option, 5
-E
 command line option, 5
-O
 command line option, 5
-OO
 command line option, 5
-Q <arg>
 command line option, 5
-S
 command line option, 5
-V
 command line option, 4
-W arg
 command line option, 6
-c <command>
 command line option, 3
-d
 command line option, 5
-h
 command line option, 4
-i
 command line option, 5
-m <module-name>
 command line option, 3
-s
 command line option, 5
-t
 command line option, 5
-u
 command line option, 5
-v
 command line option, 6
-x
 command line option, 6
..., 21
%PATH%, 14

%PYTHONPATH%, 15
__future__, 23
__slots__, 26
>>>, 21
2to3, 21

A

abstract base class, 21
argument, 21
attribute, 21

B

BDFL, 21
bytecode, 21

C

class, 21
classic class, 21
coercion, 21
command line option
 -help, 4
 -version, 4
 -3, 6
 -B, 5
 -E, 5
 -O, 5
 -OO, 5
 -Q <arg>, 5
 -S, 5
 -V, 4
 -W arg, 6
 -c <command>, 3
 -d, 5
 -h, 4
 -i, 5
 -m <module-name>, 3
 -s, 5
 -t, 5
 -u, 5
 -v, 6
 -x, 6
complex number, 21
context manager, 22
CPython, 22

D

decorator, 22
descriptor, 22
dictionary, 22
docstring, 22
duck-typing, 22

E

EAFP, 22
environment variable
 %PATH%, 14
 %PYTHONPATH%, 15
exec_prefix, 10
PATH, 7, 10
prefix, 10
PYTHON*, 5
PYTHONCASEOK, 8
PYTHONDEBUG, 5, 7
PYTHONDONTWRITEBYTECODE, 5, 8
PYTHONDUMPREFS, 8
PYTHONEXECUTABLE, 8
PYTHONHOME, 5, 7
PYTHONINSPECT, 5, 7
PYTHONIOENCODING, 8
PYTHONMALLOCSTATS, 8
PYTHONNOUSERSITE, 8
PYTHONOPTIMIZE, 5, 7
PYTHONPATH, 5, 7, 14, 18
PYTHONSTARTUP, 5, 7
PYTHONTHREADDEBUG, 8
PYTHONUNBUFFERED, 6, 8
PYTHONUSERBASE, 8
PYTHONVERBOSE, 6, 8
PYTHONY2K, 7
exec_prefix, 10
expression, 22
extension module, 23

F

finder, 23
function, 23

G

garbage collection, 23
generator, 23
generator expression, 23
GIL, 23
global interpreter lock, 23

H

hashable, 23

I

IDLE, 23
immutable, 23
importer, 24
integer division, 24

interactive, 24
interpreted, 24
iterable, 24
iterator, 24

K

keyword argument, 24

L

lambda, 24
LBYL, 24
list, 24
list comprehension, 24
loader, 24

M

mapping, 25
metaclass, 25
method, 25
mutable, 25

N

named tuple, 25
namespace, 25
nested scope, 25
new-style class, 25

O

object, 25

P

PATH, 7, 10
positional argument, 25
prefix, 10
Python 3000, 25
Python Enhancement Proposals
 PEP 11, 13
 PEP 230, 6
 PEP 238, 5
 PEP 302, 23, 25
 PEP 338, 4
 PEP 343, 22
 PEP 370, 5, 8
PYTHON*, 5
PYTHONDEBUG, 5
PYTHONDONTWRITEBYTECODE, 5
PYTHONHOME, 5, 7
Pythonic, 25
PYTHONINSPECT, 5
PYTHONOPTIMIZE, 5
PYTHONPATH, 5, 7, 14, 18
PYTHONSTARTUP, 5
PYTHONUNBUFFERED, 6
PYTHONVERBOSE, 6

R

reference count, 26

S

sequence, [26](#)
slice, [26](#)
special method, [26](#)
statement, [26](#)

T

triple-quoted string, [26](#)
type, [26](#)

V

virtual machine, [26](#)

Z

Zen of Python, [26](#)