
Python Setup and Usage

Δημοσίευση 3.14.0rc2

Guido van Rossum and the Python development team

Σεπτεμβρίου 01, 2025

**Python Software Foundation
Email: docs@python.org**

1	Command line and environment	3
1.1	Command line	3
1.1.1	Interface options	3
1.1.2	Generic options	5
1.1.3	Miscellaneous options	6
1.1.4	Controlling color	11
1.2	Environment variables	11
1.2.1	Debug-mode variables	18
2	Χρήση της Python σε πλατφόρμες Unix	19
2.1	Λήψη και εγκατάσταση της πιο πρόσφατης έκδοσης Python	19
2.1.1	Σε Linux	19
2.1.2	Σε FreeBSD και OpenBSD	20
2.2	Μεταγλώττιση της Python	20
2.3	Διαδρομές και αρχεία που σχετίζονται με την Python	20
2.4	Διάφορα	21
2.5	Custom OpenSSL	21
3	Configure Python	23
3.1	Build Requirements	23
3.2	Generated files	24
3.2.1	configure script	24
3.3	Configure Options	24
3.3.1	General Options	24
3.3.2	C compiler options	27
3.3.3	Linker options	28
3.3.4	Options for third-party dependencies	28
3.3.5	WebAssembly Options	29
3.3.6	Install Options	29
3.3.7	Performance options	30
3.3.8	Python Debug Build	32
3.3.9	Debug options	32
3.3.10	Linker options	33
3.3.11	Libraries options	34
3.3.12	Security Options	35
3.3.13	macOS Options	36
3.3.14	iOS Options	36
3.3.15	Cross Compiling Options	37
3.4	Python Build System	37
3.4.1	Main files of the build system	37
3.4.2	Main build steps	37

3.4.3	Main Makefile targets	38
3.4.4	C extensions	39
3.5	Compiler and linker flags	39
3.5.1	Preprocessor flags	40
3.5.2	Compiler flags	40
3.5.3	Linker flags	42
4	Using Python on Windows	43
4.1	Python Install Manager	43
4.1.1	Installation	43
4.1.2	Basic Use	44
4.1.3	Command Help	45
4.1.4	Listing Runtimes	45
4.1.5	Installing Runtimes	46
4.1.6	Offline Installs	46
4.1.7	Uninstalling Runtimes	47
4.1.8	Configuration	47
4.1.9	Shebang lines	48
4.1.10	Advanced Installation	49
4.1.11	Administrative Configuration	50
4.1.12	Installing Free-threaded Binaries	51
4.1.13	Troubleshooting	51
4.2	The embeddable package	53
4.2.1	Python Application	53
4.2.2	Embedding Python	53
4.3	The nuget.org packages	54
4.3.1	Free-threaded packages	54
4.4	Alternative bundles	55
4.5	Supported Windows versions	55
4.6	Removing the MAX_PATH Limitation	55
4.7	UTF-8 mode	55
4.8	Finding modules	56
4.9	Additional modules	57
4.9.1	PyWin32	57
4.9.2	cx_Freeze	57
4.10	Compiling Python on Windows	57
4.11	The full installer (deprecated)	58
4.11.1	Installation steps	58
4.11.2	Removing the MAX_PATH Limitation	59
4.11.3	Installing Without UI	59
4.11.4	Installing Without Downloading	61
4.11.5	Modifying an install	61
4.11.6	Installing Free-threaded Binaries	62
4.12	Python Launcher for Windows (Deprecated)	62
4.12.1	Getting started	63
4.12.2	Shebang Lines	64
4.12.3	Arguments in shebang lines	65
4.12.4	Customization	65
4.12.5	Diagnostics	67
4.12.6	Dry Run	67
4.12.7	Install on demand	67
4.12.8	Return codes	67
5	Χρησιμοποιώντας Python σε macOS	69
5.1	Χρησιμοποιώντας Python για macOS από το <code>python.org</code>	69
5.1.1	Βήματα εγκατάστασης	69
5.1.2	Πώς να εκτελέσετε ένα σενάριο Python	77
5.2	Εναλλακτικές Διανομές	77

5.3	Εγκατάσταση πρόσθετων πακέτων Python	78
5.4	Προγραμματισμός GUI	78
5.5	Προχωρημένα θέματα	78
5.5.1	Εγκατάσταση εκδόσεων Free-threaded	78
5.5.2	Εγκατάσταση χρησιμοποιώντας τη γραμμή εντολών	80
5.5.3	Διανομή εφαρμογών Python	81
5.5.4	Συμμόρφωση App Store	81
5.6	Άλλοι πόροι	82
6	Using Python on Android	83
6.1	Adding Python to an Android app	83
6.2	Building a Python package for Android	84
7	Using Python on iOS	85
7.1	Python at runtime on iOS	85
7.1.1	iOS version compatibility	85
7.1.2	Platform identification	85
7.1.3	Standard library availability	86
7.1.4	Binary extension modules	86
7.1.5	Compiler stub binaries	86
7.2	Installing Python on iOS	87
7.2.1	Tools for building iOS apps	87
7.2.2	Adding Python to an iOS project	87
7.2.3	Testing a Python package	90
7.3	App Store Compliance	90
8	Επεξεργαστές Κειμένου και IDEs	91
8.1	IDLE — Επεξεργαστής και shell της Python	91
8.2	Άλλοι Επεξεργαστές Κειμένου και IDEs	91
A'	Γλωσσάρι	93
B'	Σχετικά με την τεκμηρίωση	115
B'.1	Συντελεστές στη τεκμηρίωση της Python	115
Γ'	Ιστορία και Άδεια	117
Γ'.1	Η ιστορία του λογισμικού	117
Γ'.2	Όροι και προϋποθέσεις για την πρόσβαση ή την χρήση της Python με άλλους τρόπους	118
Γ'.2.1	PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2	118
Γ'.2.2	ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ BEOPEN.COM ΓΙΑ PYTHON 2.0	119
Γ'.2.3	ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ CNRI ΓΙΑ PYTHON 1.6.1	120
Γ'.2.4	ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ CWI ΓΙΑ PYTHON 0.9.0 ΕΩΣ 1.2	122
Γ'.2.5	ZERO-CLAUSE BSD ΑΔΕΙΑ ΓΙΑ ΤΟΝ ΚΩΔΙΚΑ ΣΤΗΝ ΤΕΚΜΗΡΙΩΣΗ ΤΗΣ PYTHON	123
Γ'.3	Άδειες και Ευχαριστίες για Ενσωματωμένο Λογισμικό	123
Γ'.3.1	Mersenne Twister	123
Γ'.3.2	Sockets	124
Γ'.3.3	Ασύγχρονες socket υπηρεσίες	125
Γ'.3.4	Διαχείριση Cookie	125
Γ'.3.5	Ανίχνευση εκτέλεσης	125
Γ'.3.6	Συναρτήσεις UUencode και UUdecode	126
Γ'.3.7	Κλήσεις Απομακρυσμένης Διαδικασίας XML	127
Γ'.3.8	test_epoll	127
Γ'.3.9	Επιλογή kqueue	128
Γ'.3.10	SipHash24	128
Γ'.3.11	strtod και dtoa	129
Γ'.3.12	OpenSSL	129
Γ'.3.13	expat	132
Γ'.3.14	libffi	133
Γ'.3.15	zlib	133

Γ'.3.16	cfuhash	134
Γ'.3.17	libmpdec	135
Γ'.3.18	W3C C14N σουίτα δοκιμής	135
Γ'.3.19	mimalloc	136
Γ'.3.20	asyncio	136
Γ'.3.21	Καθολικές Απεριόριστες Ακολουθίες (ΚΑΑ)	137
Γ'.3.22	Δεσμεύσεις Zstandard	137
Δ' Copyright		139
Ευρετήριο		141

Αυτό το μέρος της τεκμηρίωσης είναι αφιερωμένο σε γενικές πληροφορίες σχετικά με τη ρύθμιση του περιβάλλοντος της Python σε διαφορετικές πλατφόρμες, την εκκίνηση του διεργαστή και πράγματα που κάνουν το να δουλεύεις με την Python ευκολότερα.

Command line and environment

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

Λεπτομέρεια υλοποίησης CPython: Other implementations’ command line schemes may differ. See implementations for further resources.

1.1 Command line

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

```
python [-bBdEhiIOPqRsSuvVWx?] [-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. For more on interactive mode, see `tut-interac`.
- 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).

Raises an auditing event `cpython.run_command` with argument `command`.

Άλλαξε στην έκδοση 3.14: *command* is automatically dedented before execution.

-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 absolute 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).

Package names (including namespace packages) are also permitted. When a package name is supplied instead of a normal module, the interpreter will execute `<pkg>.__main__` as the main module. This behaviour is deliberately similar to the handling of directories and zipfiles that are passed to the interpreter as the script argument.

Σημείωση

This option cannot be used with built-in 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 (while the module file is being located, the first element will be set to `"-m"`). As with the `-c` option, the current directory will be added to the start of `sys.path`.

`-I` option can be used to run the script in isolated mode where `sys.path` contains neither the current directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too.

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

```
python -m timeit -s "setup here" "benchmarked code here"
python -m timeit -h # for details
```

Raises an auditing event `cpython.run_module` with argument `module-name`.

Δείτε επίσης

`runpy.run_module()`

Equivalent functionality directly available to Python code

PEP 338 – Executing modules as scripts

Άλλαξε στην έκδοση 3.1: Supply the package name to run a `__main__` submodule.

Άλλαξε στην έκδοση 3.4: namespace packages are also supported

-

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`.

Raises an auditing event `cpython.run_stdin` with no arguments.

<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.

`-I` option can be used to run the script in isolated mode where `sys.path` contains neither the script's directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too.

Raises an auditing event `cpython.run_file` with argument `filename`.

➡ Δείτε επίσης

`runpy.run_path()`

Equivalent functionality directly available to Python code

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`. Also, tab-completion and history editing is automatically enabled, if available on your platform (see `rlcompleter-config`).

➡ Δείτε επίσης

tut-invoking

Άλλαξε στην έκδοση 3.4: Automatic enabling of tab-completion and history editing.

1.1.2 Generic options

`-?`

`-h`

`--help`

Print a short description of all command line options and corresponding environment variables and exit.

`--help-env`

Print a short description of Python-specific environment variables and exit.

Added in version 3.11.

`--help-xoptions`

Print a description of implementation-specific `-X` options and exit.

Added in version 3.11.

--help-all

Print complete usage information and exit.

Added in version 3.11.

-V

--version

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

```
Python 3.8.0b2+
```

When given twice, print more information about the build, like:

```
Python 3.8.0b2+ (3.8:0c076caaa8, Apr 20 2019, 21:55:00)
[GCC 6.2.0 20161005]
```

Added in version 3.6: The `-VV` option.

1.1.3 Miscellaneous options

-b

Issue a warning when converting `bytes` or `bytearray` to `str` without specifying encoding or comparing `bytes` or `bytearray` with `str` or `bytes` with `int`. Issue an error when the option is given twice (`-bb`).

Αλλάξε στην έκδοση 3.5: Affects also comparisons of `bytes` with `int`.

-B

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

--check-hash-based-pycs `default|always|never`

Control the validation behavior of hash-based `.pyc` files. See [pyc-invalidation](#). When set to `default`, checked and unchecked hash-based bytecode cache files are validated according to their default semantics. When set to `always`, all hash-based `.pyc` files, whether checked or unchecked, are validated against their corresponding source file. When set to `never`, hash-based `.pyc` files are not validated against their corresponding source files.

The semantics of timestamp-based `.pyc` files are unaffected by this option.

-d

Turn on parser debugging output (for expert only). See also the [PYTHONDEBUG](#) environment variable.

This option requires a [debug build of Python](#), otherwise it's ignored.

-E

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

See also the `-P` and `-I` (isolated) options.

-i

Enter interactive mode after execution.

Using the `-i` option will enter interactive mode in any of the following circumstances:

- When a script is passed as first argument
- When the `-c` option is used
- When the `-m` option is used

Interactive mode will start 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](#).

-I

Run Python in isolated mode. This also implies `-E`, `-P` and `-s` options.

In isolated mode `sys.path` contains neither the script's directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too. Further restrictions may be imposed to prevent the user from injecting malicious code.

Added in version 3.4.

-O

Remove assert statements and any code conditional on the value of `__debug__`. Augment the filename for compiled (*bytecode*) files by adding `.opt-1` before the `.pyc` extension (see [PEP 488](#)). See also [PYTHONOPTIMIZE](#).

Άλλαξε στην έκδοση 3.5: Modify `.pyc` filenames according to [PEP 488](#).

-OO

Do `-O` and also discard docstrings. Augment the filename for compiled (*bytecode*) files by adding `.opt-2` before the `.pyc` extension (see [PEP 488](#)).

Άλλαξε στην έκδοση 3.5: Modify `.pyc` filenames according to [PEP 488](#).

-P

Don't prepend a potentially unsafe path to `sys.path`:

- `python -m module` command line: Don't prepend the current working directory.
- `python script.py` command line: Don't prepend the script's directory. If it's a symbolic link, resolve symbolic links.
- `python -c code` and `python` (REPL) command lines: Don't prepend an empty string, which means the current working directory.

See also the [PYTHONSAFEPATH](#) environment variable, and `-E` and `-I` (isolated) options.

Added in version 3.11.

-q

Don't display the copyright and version messages even in interactive mode.

Added in version 3.2.

-R

Turn on hash randomization. This option only has an effect if the [PYTHONHASHSEED](#) environment variable is set to 0, since hash randomization is enabled by default.

On previous versions of Python, this option turns on hash randomization, so that the `__hash__()` values of `str` and `bytes` objects are «salted» with an unpredictable random value. Although they remain constant within an individual Python process, they are not predictable between repeated invocations of Python.

Hash randomization is intended to provide protection against a denial-of-service caused by carefully chosen inputs that exploit the worst case performance of a dict construction, $O(n^2)$ complexity. See <http://ocert.org/advisories/ocert-2011-003.html> for details.

[PYTHONHASHSEED](#) allows you to set a fixed value for the hash seed secret.

Added in version 3.2.3.

Άλλαξε στην έκδοση 3.7: The option is no longer ignored.

-s

Don't add the user `site-packages` directory to `sys.path`.

See also [PYTHONNOUSERSITE](#).

 Δείτε επίσης

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. Also disable these manipulations if `site` is explicitly imported later (call `site.main()` if you want them to be triggered).

-u

Force the stdout and stderr streams to be unbuffered. This option has no effect on the stdin stream.

See also [PYTHONUNBUFFERED](#).

Άλλαξε στην έκδοση 3.7: The text layer of the stdout and stderr streams now is unbuffered.

-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.

Άλλαξε στην έκδοση 3.10: The `site` module reports the site-specific paths and `.pth` files being processed.

See also [PYTHONVERBOSE](#).

-W *arg*

Warning control. Python's warning machinery by default prints warning messages to `sys.stderr`.

The simplest settings apply a particular action unconditionally to all warnings emitted by a process (even those that are otherwise ignored by default):

```
-Wdefault    # Warn once per call location
-Werror      # Convert to exceptions
-Walways     # Warn every time
-Wall        # Same as -Walways
-Wmodule     # Warn once per calling module
-Wonce       # Warn once per Python process
-Wignore     # Never warn
```

The action names can be abbreviated as desired and the interpreter will resolve them to the appropriate action name. For example, `-Wi` is the same as `-Wignore`.

The full form of argument is:

```
action:message:category:module:lineno
```

Empty fields match all values; trailing empty fields may be omitted. For example `-W ignore::DeprecationWarning` ignores all `DeprecationWarning` warnings.

The *action* field is as explained above but only applies to warnings that match the remaining fields.

The *message* field must match the whole warning message; this match is case-insensitive.

The *category* field matches the warning category (ex: `DeprecationWarning`). 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 *module* field matches the (fully qualified) module name; this match is case-sensitive.

The *lineno* field matches the line number, where zero matches all line numbers and is thus equivalent to an omitted line number.

Multiple `-W` options can 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 using the `PYTHONWARNINGS` environment variable and from within a Python program using the `warnings` module. For example, the `warnings.filterwarnings()` function can be used to use a regular expression on the warning message.

See `warning-filter` and `describing-warning-filters` for more details.

-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.

-X

Reserved for various implementation-specific options. CPython currently defines the following possible values:

- `-X faulthandler` to enable `faulthandler`. See also `PYTHONFAULTHANDLER`.
Added in version 3.3.
- `-X showrefcount` to output the total reference count and number of used memory blocks when the program finishes or after each statement in the interactive interpreter. This only works on *debug builds*.
Added in version 3.4.
- `-X tracemalloc` to start tracing Python memory allocations using the `tracemalloc` module. By default, only the most recent frame is stored in a traceback of a trace. Use `-X tracemalloc=NFRAME` to start tracing with a traceback limit of `NFRAME` frames. See `tracemalloc.start()` and `PYTHONTRACEMALLOC` for more information.
Added in version 3.4.
- `-X int_max_str_digits` configures the integer string conversion length limitation. See also `PYTHONINTMAXSTRDIGITS`.
Added in version 3.11.
- `-X importtime` to show how long each import takes. It shows module name, cumulative time (including nested imports) and self time (excluding nested imports). Note that its output may be broken in multi-threaded application. Typical usage is `python -X importtime -c 'import asyncio'`.
`-X importtime=2` enables additional output that indicates when an imported module has already been loaded. In such cases, the string cached will be printed in both time columns.
See also `PYTHONPROFILEIMPORTTIME`.
Added in version 3.7.
Αλλάξε στην έκδοση 3.14: Added `-X importtime=2` to also trace imports of loaded modules, and reserved values other than 1 and 2 for future use.
- `-X dev`: enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default. See also `PYTHONDEVMODE`.
Added in version 3.7.
- `-X utf8` enables the Python UTF-8 Mode. `-X utf8=0` explicitly disables Python UTF-8 Mode (even when it would otherwise activate automatically). See also `PYTHONUTF8`.
Added in version 3.7.
- `-X pycache_prefix=PATH` enables writing `.pyc` files to a parallel tree rooted at the given directory instead of to the code tree. See also `PYTHONPYCACHEPREFIX`.
Added in version 3.8.
- `-X warn_default_encoding` issues a `EncodingWarning` when the locale-specific default encoding is used for opening files. See also `PYTHONWARNDEFAULTENCODING`.

Added in version 3.10.

- `-X no_debug_ranges` disables the inclusion of the tables mapping extra location information (end line, start column offset and end column offset) to every instruction in code objects. This is useful when smaller code objects and pyc files are desired as well as suppressing the extra visual location indicators when the interpreter displays tracebacks. See also [PYTHONNODEBUGRANGES](#).

Added in version 3.11.

- `-X frozen_modules` determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` if this is an installed Python (the normal case). If it's under development (running from the source tree) then the default is `off`. Note that the `importlib_bootstrap` and `importlib_bootstrap_external` frozen modules are always used, even if this flag is set to `off`. See also [PYTHON_FROZEN_MODULES](#).

Added in version 3.11.

- `-X perf` enables support for the Linux `perf` profiler. When this option is provided, the `perf` profiler will be able to report Python calls. This option is only available on some platforms and will do nothing if is not supported on the current system. The default value is «off». See also [PYTHONPERFSUPPORT](#) and `perf_profiling`.

Added in version 3.12.

- `-X perf_jit` enables support for the Linux `perf` profiler with DWARF support. When this option is provided, the `perf` profiler will be able to report Python calls using DWARF information. This option is only available on some platforms and will do nothing if is not supported on the current system. The default value is «off». See also [PYTHON_PERF_JIT_SUPPORT](#) and `perf_profiling`.

Added in version 3.13.

- `-X disable_remote_debug` disables the remote debugging support as described in [PEP 768](#). This includes both the functionality to schedule code for execution in another process and the functionality to receive code for execution in the current process.

This option is only available on some platforms and will do nothing if is not supported on the current system. See also [PYTHON_DISABLE_REMOTE_DEBUG](#) and [PEP 768](#).

Added in version 3.14.

- `-X cpu_count=n` overrides `os.cpu_count()`, `os.process_cpu_count()`, and `multiprocessing.cpu_count()`. `n` must be greater than or equal to 1. This option may be useful for users who need to limit CPU resources of a container system. See also [PYTHON_CPU_COUNT](#). If `n` is default, nothing is overridden.

Added in version 3.13.

- `-X presite=package.module` specifies a module that should be imported before the `site` module is executed and before the `__main__` module exists. Therefore, the imported module isn't `__main__`. This can be used to execute code early during Python initialization. Python needs to be *built in debug mode* for this option to exist. See also [PYTHON_PRESITE](#).

Added in version 3.13.

- `-X gil=0,1` forces the GIL to be disabled or enabled, respectively. Setting to 0 is only available in builds configured with `--disable-gil`. See also [PYTHON_GIL](#) and `whatsnew313-free-threaded-cpython`.

Added in version 3.13.

- `-X thread_inherit_context=0,1` causes `Thread` to, by default, use a copy of context of the caller of `Thread.start()` when starting. Otherwise, threads will start with an empty context. If unset, the value of this option defaults to 1 on free-threaded builds and to 0 otherwise. See also [PYTHON_THREAD_INHERIT_CONTEXT](#).

Added in version 3.14.

- `-X context_aware_warnings=0,1` causes the `warnings.catch_warnings` context manager to use a `ContextVar` to store warnings filter state. If unset, the value of this option defaults to 1 on free-threaded builds and to 0 otherwise. See also [PYTHON_CONTEXT_AWARE_WARNINGS](#).

Added in version 3.14.

- `-X tlbc=0,1` enables (1, the default) or disables (0) thread-local bytecode in builds configured with `--disable-gil`. When disabled, this also disables the specializing interpreter. See also [PYTHON_TLBC](#).

Added in version 3.14.

It also allows passing arbitrary values and retrieving them through the `sys._xoptions` dictionary.

Added in version 3.2.

Άλλαξε στην έκδοση 3.9: Removed the `-X showalloccount` option.

Άλλαξε στην έκδοση 3.10: Removed the `-X oldparser` option.

Removed in version 3.14: `-J` is no longer reserved for use by [Jython](#), and now has no special meaning.

1.1.4 Controlling color

The Python interpreter is configured by default to use colors to highlight output in certain situations such as when displaying tracebacks. This behavior can be controlled by setting different environment variables.

Setting the environment variable `TERM` to `dumb` will disable color.

If the [FORCE_COLOR](#) environment variable is set, then color will be enabled regardless of the value of `TERM`. This is useful on CI systems which aren't terminals but can still display ANSI escape sequences.

If the [NO_COLOR](#) environment variable is set, Python will disable all color in the output. This takes precedence over [FORCE_COLOR](#).

All these environment variables are used also by other tools to control color output. To control the color output only in the Python interpreter, the [PYTHON_COLORS](#) environment variable can be used. This variable takes precedence over `NO_COLOR`, which in turn takes precedence over [FORCE_COLOR](#).

1.2 Environment variables

These environment variables influence Python's behavior, they are processed before the command-line switches other than `-E` or `-I`. It is customary that command-line switches override environmental variables where there is a conflict.

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`.

PYTHONSAFEPATH

If this is set to a non-empty string, don't prepend a potentially unsafe path to `sys.path`: see the `-P` option for details.

Added in version 3.11.

PYTHONPLATLIBDIR

If this is set to a non-empty string, it overrides the `sys.platlibdir` value.

Added in version 3.9.

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` and the hook `sys.__interactivehook__` in this file.

Raises an auditing event `cpython.run_startup` with the filename as the argument when called on startup.

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.

PYTHONBREAKPOINT

If this is set, it names a callable using dotted-path notation. The module containing the callable will be imported and then the callable will be run by the default implementation of `sys.breakpointhook()` which itself is called by built-in `breakpoint()`. If not set, or set to the empty string, it is equivalent to the value `«pdb.set_trace»`. Setting this to the string `«0»` causes the default implementation of `sys.breakpointhook()` to do nothing but return immediately.

Added in version 3.7.

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.

This environment variable requires a *debug build of Python*, otherwise it's ignored.

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.

Raises an auditing event `cpython.run_stdin` with no arguments.

Αλλάξε στην έκδοση 3.12.5: (also 3.11.10, 3.10.15, 3.9.20, and 3.8.20) Emits audit events.

Αλλάξε στην έκδοση 3.13: Uses PyREPL if possible, in which case `PYTHONSTARTUP` is also executed. Emits audit events.

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 and macOS.

PYTHONDONTWRITEBYTECODE

If this is set to a non-empty string, Python won't try to write `.pyc` files on the import of source modules. This is equivalent to specifying the `-B` option.

PYTHONPYCACHEPREFIX

If this is set, Python will write `.pyc` files in a mirror directory tree at this path, instead of in `__pycache__` directories within the source tree. This is equivalent to specifying the `-X pycache_prefix=PATH` option.

Added in version 3.8.

PYTHONHASHSEED

If this variable is not set or set to `random`, a random value is used to seed the hashes of `str` and `bytes` objects.

If `PYTHONHASHSEED` is set to an integer value, it is used as a fixed seed for generating the `hash()` of the types covered by the hash randomization.

Its purpose is to allow repeatable hashing, such as for selftests for the interpreter itself, or to allow a cluster of python processes to share hash values.

The integer must be a decimal number in the range `[0,4294967295]`. Specifying the value 0 will disable hash randomization.

Added in version 3.2.3.

PYTHONINTMAXSTRDIGITS

If this variable is set to an integer, it is used to configure the interpreter's global integer string conversion length limitation.

Added in version 3.11.

PYTHONIOENCODING

If this is set before running the interpreter, it overrides the encoding used for `stdin/stdout/stderr`, in the syntax `encodingname:errorhandler`. Both the `encodingname` and the `:errorhandler` parts are optional and have the same meaning as in `str.encode()`.

For `stderr`, the `:errorhandler` part is ignored; the handler will always be `'backslashreplace'`.

Αλλαξε στην έκδοση 3.4: The `encodingname` part is now optional.

Αλλαξε στην έκδοση 3.6: On Windows, the encoding specified by this variable is ignored for interactive console buffers unless `PYTHONLEGACYWINDOWSSTDIO` is also specified. Files and pipes redirected through the standard streams are not affected.

PYTHONNOUSERSITE

If this is set, Python won't add the user `site-packages` directory to `sys.path`.

➡ Δείτε επίσης

PEP 370 – Per user site-packages directory

PYTHONUSERBASE

Defines the user base directory, which is used to compute the path of the user `site-packages` directory and installation paths for `python -m pip install --user`.

➡ Δείτε επίσης

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 macOS.

PYTHONWARNINGS

This is equivalent to the `-W` option. If set to a comma separated string, it is equivalent to specifying `-W` multiple times, with filters later in the list taking precedence over those earlier in the list.

The simplest settings apply a particular action unconditionally to all warnings emitted by a process (even those that are otherwise ignored by default):

```
PYTHONWARNINGS=default # Warn once per call location
PYTHONWARNINGS=error   # Convert to exceptions
PYTHONWARNINGS=always  # Warn every time
PYTHONWARNINGS=all     # Same as PYTHONWARNINGS=always
PYTHONWARNINGS=module  # Warn once per calling module
PYTHONWARNINGS=once    # Warn once per Python process
PYTHONWARNINGS=ignore  # Never warn
```

See `warning-filter` and `describing-warning-filters` for more details.

PYTHONFAULTHANDLER

If this environment variable is set to a non-empty string, `faulthandler.enable()` is called at startup: install a handler for `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` and `SIGILL` signals to dump the Python traceback. This is equivalent to `-X faulthandler` option.

Added in version 3.3.

PYTHONTRACEMALLOC

If this environment variable is set to a non-empty string, start tracing Python memory allocations using the `tracemalloc` module. The value of the variable is the maximum number of frames stored in a traceback of a trace. For example, `PYTHONTRACEMALLOC=1` stores only the most recent frame. See the `tracemalloc.start()` function for more information. This is equivalent to setting the `-X tracemalloc` option.

Added in version 3.4.

PYTHONPROFILEIMPORTTIME

If this environment variable is set to 1, Python will show how long each import takes. If set to 2, Python will include output for imported modules that have already been loaded. This is equivalent to setting the `-X importtime` option.

Added in version 3.7.

Άλλαξε στην έκδοση 3.14: Added `PYTHONPROFILEIMPORTTIME=2` to also trace imports of loaded modules.

PYTHONASYNCIODEBUG

If this environment variable is set to a non-empty string, enable the debug mode of the `asyncio` module.

Added in version 3.4.

PYTHONMALLOC

Set the Python memory allocators and/or install debug hooks.

Set the family of memory allocators used by Python:

- `default`: use the default memory allocators.
- `malloc`: use the `malloc()` function of the C library for all domains (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).
- `pymalloc`: use the `pymalloc` allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.
- `mimalloc`: use the `mimalloc` allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.

Install debug hooks:

- `debug`: install debug hooks on top of the default memory allocators.

- `malloc_debug`: same as `malloc` but also install debug hooks.
- `pymalloc_debug`: same as `pymalloc` but also install debug hooks.
- `mimalloc_debug`: same as `mimalloc` but also install debug hooks.

Added in version 3.6.

Άλλαξε στην έκδοση 3.7: Added the "default" allocator.

PYTHONMALLOCSSTATS

If set to a non-empty string, Python will print statistics of the pymalloc memory allocator every time a new pymalloc object arena is created, and on shutdown.

This variable is ignored if the `PYTHONMALLOC` environment variable is used to force the `malloc()` allocator of the C library, or if Python is configured without pymalloc support.

Άλλαξε στην έκδοση 3.6: This variable can now also be used on Python compiled in release mode. It now has no effect if set to an empty string.

PYTHONLEGACYWINDOWSFSENCODING

If set to a non-empty string, the default *filesystem encoding and error handler* mode will revert to their pre-3.6 values of "mbcs" and "replace", respectively. Otherwise, the new defaults "utf-8" and "surrogatepass" are used.

This may also be enabled at runtime with `sys._enablelegacywindowsfsencoding()`.

Διαθεσιμότητα: Windows.

Added in version 3.6: See [PEP 529](#) for more details.

PYTHONLEGACYWINDOWSSTDIO

If set to a non-empty string, does not use the new console reader and writer. This means that Unicode characters will be encoded according to the active console code page, rather than using utf-8.

This variable is ignored if the standard streams are redirected (to files or pipes) rather than referring to console buffers.

Διαθεσιμότητα: Windows.

Added in version 3.6.

PYTHONCOERCECLOCALE

If set to the value 0, causes the main Python command line application to skip coercing the legacy ASCII-based C and POSIX locales to a more capable UTF-8 based alternative.

If this variable is *not* set (or is set to a value other than 0), the `LC_ALL` locale override environment variable is also not set, and the current locale reported for the `LC_CTYPE` category is either the default C locale, or else the explicitly ASCII-based `POSIX` locale, then the Python CLI will attempt to configure the following locales for the `LC_CTYPE` category in the order listed before loading the interpreter runtime:

- `C.UTF-8`
- `C.utf8`
- `UTF-8`

If setting one of these locale categories succeeds, then the `LC_CTYPE` environment variable will also be set accordingly in the current process environment before the Python runtime is initialized. This ensures that in addition to being seen by both the interpreter itself and other locale-aware components running in the same process (such as the GNU readline library), the updated setting is also seen in subprocesses (regardless of whether or not those processes are running a Python interpreter), as well as in operations that query the environment rather than the current C locale (such as Python's own `locale.getdefaultlocale()`).

Configuring one of these locales (either explicitly or via the above implicit locale coercion) automatically enables the `surrogateescape` error handler for `sys.stdin` and `sys.stdout` (`sys.stderr` continues to use `backslashreplace` as it does in any other locale). This stream handling behavior can be overridden using `PYTHONIOENCODING` as usual.

For debugging purposes, setting `PYTHONCOERCECLOCALE=warn` will cause Python to emit warning messages on `stderr` if either the locale coercion activates, or else if a locale that *would* have triggered coercion is still active when the Python runtime is initialized.

Also note that even when locale coercion is disabled, or when it fails to find a suitable target locale, `PYTHONUTF8` will still activate by default in legacy ASCII-based locales. Both features must be disabled in order to force the interpreter to use ASCII instead of UTF-8 for system interfaces.

Διαθεσιμότητα: Unix.

Added in version 3.7: See [PEP 538](#) for more details.

PYTHONDEVMODE

If this environment variable is set to a non-empty string, enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default. This is equivalent to setting the `-X dev` option.

Added in version 3.7.

PYTHONUTF8

If set to 1, enable the Python UTF-8 Mode.

If set to 0, disable the Python UTF-8 Mode.

Setting any other non-empty string causes an error during interpreter initialisation.

Added in version 3.7.

PYTHONWARNDEFAULTENCODING

If this environment variable is set to a non-empty string, issue a `EncodingWarning` when the locale-specific default encoding is used.

See `io-encoding-warning` for details.

Added in version 3.10.

PYTHONNODEBUGRANGES

If this variable is set, it disables the inclusion of the tables mapping extra location information (end line, start column offset and end column offset) to every instruction in code objects. This is useful when smaller code objects and pyc files are desired as well as suppressing the extra visual location indicators when the interpreter displays tracebacks.

Added in version 3.11.

PYTHONPERFSUPPORT

If this variable is set to a nonzero value, it enables support for the Linux `perf` profiler so Python calls can be detected by it.

If set to 0, disable Linux `perf` profiler support.

See also the `-X perf` command-line option and `perf_profiling`.

Added in version 3.12.

PYTHON_PERF_JIT_SUPPORT

If this variable is set to a nonzero value, it enables support for the Linux `perf` profiler so Python calls can be detected by it using DWARF information.

If set to 0, disable Linux `perf` profiler support.

See also the `-X perf_jit` command-line option and `perf_profiling`.

Added in version 3.13.

PYTHON_DISABLE_REMOTE_DEBUG

If this variable is set to a non-empty string, it disables the remote debugging feature described in [PEP 768](#). This includes both the functionality to schedule code for execution in another process and the functionality to receive code for execution in the current process.

See also the `-X disable_remote_debug` command-line option.

Added in version 3.14.

PYTHON_CPU_COUNT

If this variable is set to a positive integer, it overrides the return values of `os.cpu_count()` and `os.process_cpu_count()`.

See also the `-X cpu_count` command-line option.

Added in version 3.13.

PYTHON_FROZEN_MODULES

If this variable is set to `on` or `off`, it determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` for non-debug builds (the normal case) and `off` for debug builds. Note that the `importlib_bootstrap` and `importlib_bootstrap_external` frozen modules are always used, even if this flag is set to `off`.

See also the `-X frozen_modules` command-line option.

Added in version 3.13.

PYTHON_COLORS

If this variable is set to 1, the interpreter will colorize various kinds of output. Setting it to 0 deactivates this behavior. See also [Controlling color](#).

Added in version 3.13.

PYTHON_BASIC_REPL

If this variable is set to any value, the interpreter will not attempt to load the Python-based [REPL](#) that requires `curses` and `readline`, and will instead use the traditional parser-based [REPL](#).

Added in version 3.13.

PYTHON_HISTORY

This environment variable can be used to set the location of a `.python_history` file (by default, it is `.python_history` in the user's home directory).

Added in version 3.13.

PYTHON_GIL

If this variable is set to 1, the global interpreter lock (GIL) will be forced on. Setting it to 0 forces the GIL off (needs Python configured with the `--disable-gil` build option).

See also the `-X gil` command-line option, which takes precedence over this variable, and [whatsnew313-free-threaded-cpython](#).

Added in version 3.13.

PYTHON_THREAD_INHERIT_CONTEXT

If this variable is set to 1 then `Thread` will, by default, use a copy of context of the caller of `Thread.start()` when starting. Otherwise, new threads will start with an empty context. If unset, this variable defaults to 1 on free-threaded builds and to 0 otherwise. See also `-X thread_inherit_context`.

Added in version 3.14.

PYTHON_CONTEXT_AWARE_WARNINGS

If set to 1 then the `warnings.catch_warnings` context manager will use a `ContextVar` to store warnings filter state. If unset, this variable defaults to 1 on free-threaded builds and to 0 otherwise. See `-X context_aware_warnings`.

Added in version 3.14.

PYTHON_JIT

On builds where experimental just-in-time compilation is available, this variable can force the JIT to be disabled (0) or enabled (1) at interpreter startup.

Added in version 3.13.

PYTHON_TLBC

If set to 1 enables thread-local bytecode. If set to 0 thread-local bytecode and the specializing interpreter are disabled. Only applies to builds configured with `--disable-gil`.

See also the `-X tlb` command-line option.

Added in version 3.14.

1.2.1 Debug-mode variables

PYTHONDUMPREFS

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

Needs Python configured with the `--with-trace-refs` build option.

PYTHONDUMPREFSFILE

If set, Python will dump objects and reference counts still alive after shutting down the interpreter into a file under the path given as the value to this environment variable.

Needs Python configured with the `--with-trace-refs` build option.

Added in version 3.11.

PYTHON_PRESITE

If this variable is set to a module, that module will be imported early in the interpreter lifecycle, before the `site` module is executed, and before the `__main__` module is created. Therefore, the imported module is not treated as `__main__`.

This can be used to execute code early during Python initialization.

To import a submodule, use `package.module` as the value, like in an import statement.

See also the `-X presite` command-line option, which takes precedence over this variable.

Needs Python configured with the `--with-pydebug` build option.

Added in version 3.13.

Χρήση της Python σε πλατφόρμες Unix

2.1 Λήψη και εγκατάσταση της πιο πρόσφατης έκδοσης Python

2.1.1 Σε Linux

Η Python είναι προεγκατεστημένη στις περισσότερες διανομές Linux, ενώ διατίθεται ως πακέτο σε όλες τις υπόλοιπες. Ωστόσο, υπάρχουν ορισμένα χαρακτηριστικά που μπορεί να θέλετε να χρησιμοποιήσετε, τα οποία δεν είναι διαθέσιμα στο πακέτο της διανομής σας. Μπορείτε να κάνετε `compile` την πιο πρόσφατη έκδοση της Python από τον πηγαίο κώδικα.

Στην περίπτωση που η πιο πρόσφατη έκδοση Python δεν είναι προεγκατεστημένη και επίσης δεν ανήκει στα αποθετήρια, μπορείτε να δημιουργήσετε πακέτα για τη δική σας διανομή. Δείτε τα παρακάτω links:

Δείτε επίσης

<https://www.debian.org/doc/manuals/maint-guide/first.en.html>

για τους χρήστες του Debian

<https://en.opensuse.org/Portal:Packaging>

για τους χρήστες του OpenSuse

https://docs.fedoraproject.org/en-US/package-maintainers/Packaging_Tutorial_GNU_Hello/

για τους χρήστες του Fedora

<https://slackbook.org/html/package-management-making-packages.html>

για τους χρήστες του Slackware

Εγκατάσταση του IDLE

Σε μερικές περιπτώσεις, το IDLE μπορεί να μην περιλαμβάνεται στην εγκατάσταση της Python.

- Για τους χρήστες του Debian και του Ubuntu:

```
sudo apt update
sudo apt install idle
```

- Για τους χρήστες των Fedora, RHEL, και CentOS:

```
sudo dnf install python3-idle
```

- Για τους χρήστες των SUSE και OpenSUSE:

```
sudo zypper install python3-idle
```

- Για τους χρήστες του Alpine Linux:

```
sudo apk add python3-idle
```

2.1.2 Σε FreeBSD και OpenBSD

- Χρήστες του FreeBSD, για την προσθήκη του πακέτου χρησιμοποιήστε:

```
pkg install python3
```

- Χρήστες του OpenBSD, για την προσθήκη του πακέτου χρησιμοποιήστε:

```
pkg_add -r python  
  
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/<insert your_  
↪architecture here>/python-<version>.tgz
```

Για παράδειγμα, χρήστες του i386 κάνουν λήψη της έκδοσης 2.5.1 χρησιμοποιώντας:

```
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/i386/python-2.5.  
↪1p2.tgz
```

2.2 Μεταγλώττιση της Python

Αν θέλετε να κάνετε compile το CPython μόνοι σας, το πρώτο πράγμα που πρέπει να κάνετε είναι να πάρετε τον πηγαίο κώδικα <<https://www.python.org/downloads/source/>>_. Μπορείτε να κατεβάσετε είτε τον πηγαίο κώδικα της τελευταίας έκδοσης είτε απλά να πάρετε έναν καινούργιο **clone**. (Αν θέλετε να συνεισφέρετε διορθώσεις, θα χρειαστείτε έναν κλώνο).

Η διαδικασία της μεταγλώττισης απαρτίζεται από τις συνήθεις εντολές:

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

Το *Configuration options* και οι όροι για συγκεκριμένες πλατφόρμες Unix τεκμηριώνονται εκτενώς στο αρχείο **README.rst** στην βάση του πηγαίου δέντρου της Python.

Προειδοποίηση

Το `make install` μπορεί να αντικαταστήσει ή να μεταμφιέσει το `python3` σε δυαδικό. Επομένως προτείνεται το `make altinstall` σε σχέση με το `make install` που μπορεί να εγκαταστήσει μόνο το `:file{exec_prefix}/bin/python{version}`.

2.3 Διαδρομές και αρχεία που σχετίζονται με την Python

Αυτά ενδέχεται να διαφέρουν ανάλογα με τις τοπικές συμβάσεις εγκατάστασης• τα *prefix* και *exec_prefix* εξαρτώνται από την εγκατάσταση και θα πρέπει να ερμηνεύονται όπως για το λογισμικό GNU• μπορεί να είναι τα ίδια.

Για παράδειγμα, στα περισσότερα Linux συστήματα, η προεπιλογή είναι και για τα δύο `/usr`.

File/directory	Που σημαίνει
<code>exec_prefix/bin/python3</code>	Προτεινόμενη θέση του διερμηνέα.
<code>prefix/lib/</code> <code>pythonversion,</code> <code>exec_prefix/lib/</code> <code>pythonversion</code>	Προτεινόμενες θέσεις για τους καταλόγους που περιέχουν τα βασικά modules.
<code>prefix/include/</code> <code>pythonversion,</code> <code>exec_prefix/include/</code> <code>pythonversion</code>	Προτεινόμενες θέσεις των καταλόγων που περιέχουν τα αρχεία κεφαλίδων (include files) που απαιτούνται για την ανάπτυξη επεκτάσεων της Python και την ενσωμάτωση του διερμηνέα.

2.4 Διάφορα

Για να εκτελείτε εύκολα τα Python scripts σε συστήματα Unix, πρέπει να τα κάνετε εκτελέσιμα, για παράδειγμα με

```
$ chmod +x script
```

και να προσθέσετε μια κατάλληλη γραμμή Shebang στην αρχή του script. Μια καλή επιλογή είναι συνήθως

```
#!/usr/bin/env python3
```

που κάνει αναζήτηση για τον διερμηνέα της Python σε ολόκληρο το PATH. Ωστόσο, ορισμένα Unixes μπορεί να μην έχουν την εντολή `env`, οπότε μπορεί να χρειαστεί να κωδικοποιήσετε το `/usr/bin/python3` ως διαδρομή του διερμηνέα.

Για να χρησιμοποιήσετε εντολές shell στα Python script σας, δείτε την ενότητα `subprocess`.

2.5 Custom OpenSSL

1. Για να χρησιμοποιήσετε τις ρυθμίσεις του OpenSSL και το αποθετήριο εμπιστοσύνης συστήματος, εντοπίστε τον κατάλογο με το αρχείο `openssl.cnf` ή τον συμβολικό σύνδεσμο στο `/etc`. Στις περισσότερες διανομές το αρχείο βρίσκεται είτε στο `/etc/ssl` είτε στο `/etc/pki/tls`. Ο κατάλογος θα πρέπει επίσης να περιέχει ένα αρχείο `cert.pem` και/ή έναν κατάλογο `certs`.

```
$ find /etc/ -name openssl.cnf -printf "%h\n"
/etc/ssl
```

2. Λήψη, δημιουργία και εγκατάσταση του OpenSSL. Βεβαιωθείτε ότι χρησιμοποιείτε το `install_sw` και όχι το `install`. Ο στόχος `install_sw` δεν παρακάμπτει το αρχείο `openssl.cnf`.

```
$ curl -O https://www.openssl.org/source/openssl-VERSION.tar.gz
$ tar xzf openssl-VERSION
$ pushd openssl-VERSION
$ ./config \
  --prefix=/usr/local/custom-openssl \
  --libdir=lib \
  --openssldir=/etc/ssl
$ make -j1 depend
$ make -j8
$ make install_sw
$ popd
```

3. Μεταγλώττιση της Python με προσαρμοσμένο OpenSSL (δείτε τις επιλογές `configure` `---with-openssl` και `---with-openssl-rpath`)

```
$ pushd python-3.x.x
$ ./configure -C \
    --with-openssl=/usr/local/custom-openssl \
    --with-openssl-rpath=auto \
    --prefix=/usr/local/python-3.x.x
$ make -j8
$ make altinstall
```

Σημείωση

Οι διορθωτικές εκδόσεις του OpenSSL έχουν μια δυναδική διεπαφή εφαρμογής (ABI) συμβατή προς τα πίσω. Δεν χρειάζεται να κάνετε compile εκ νέου την Python για να ενημερώσετε το OpenSSL. Αρκεί να αντικαταστήσετε την προσαρμοσμένη εγκατάσταση του OpenSSL με μια νεότερη έκδοση.

3.1 Build Requirements

Features and minimum versions required to build CPython:

- A C11 compiler. `Optional C11 features` are not required.
- On Windows, Microsoft Visual Studio 2017 or later is required.
- Support for `IEEE 754` floating-point numbers and `floating-point Not-a-Number (NaN)`.
- Support for threads.
- OpenSSL 1.1.1 is the minimum version and OpenSSL 3.0.16 is the recommended minimum version for the `ssl` and `hashlib` extension modules.
- SQLite 3.15.2 for the `sqlite3` extension module.
- Tcl/Tk 8.5.12 for the `tkinter` module.
- `libmpdec` 2.5.0 for the `decimal` module.
- Autoconf 2.72 and `aclocal` 1.16.5 are required to regenerate the `configure` script.

Άλλαξε στην έκδοση 3.1: Tcl/Tk version 8.3.1 is now required.

Άλλαξε στην έκδοση 3.5: On Windows, Visual Studio 2015 or later is now required. Tcl/Tk version 8.4 is now required.

Άλλαξε στην έκδοση 3.6: Selected C99 features are now required, like `<stdint.h>` and `static inline` functions.

Άλλαξε στην έκδοση 3.7: Thread support and OpenSSL 1.0.2 are now required.

Άλλαξε στην έκδοση 3.10: OpenSSL 1.1.1 is now required. Require SQLite 3.7.15.

Άλλαξε στην έκδοση 3.11: C11 compiler, IEEE 754 and NaN support are now required. On Windows, Visual Studio 2017 or later is required. Tcl/Tk version 8.5.12 is now required for the `tkinter` module.

Άλλαξε στην έκδοση 3.13: Autoconf 2.71, `aclocal` 1.16.5 and SQLite 3.15.2 are now required.

Άλλαξε στην έκδοση 3.14: Autoconf 2.72 is now required.

See also **PEP 7** «Style Guide for C Code» and **PEP 11** «CPython platform support».

3.2 Generated files

To reduce build dependencies, Python source code contains multiple generated files. Commands to regenerate all generated files:

```
make regen-all
make regen-stdlib-module-names
make regen-limited-abi
make regen-configure
```

The `Makefile.pre.in` file documents generated files, their inputs, and tools used to regenerate them. Search for `regen-*` make targets.

3.2.1 configure script

The `make regen-configure` command regenerates the `aclocal.m4` file and the configure script using the `Tools/build/regen-configure.sh` shell script which uses an Ubuntu container to get the same tools versions and have a reproducible output.

The container is optional, the following command can be run locally:

```
autoreconf -ivf -Werror
```

The generated files can change depending on the exact `autoconf-archive`, `aclocal` and `pkg-config` versions.

3.3 Configure Options

List all configure script options using:

```
./configure --help
```

See also the `Misc/SpecialBuilds.txt` in the Python source distribution.

3.3.1 General Options

--enable-loadable-sqlite-extensions

Support loadable extensions in the `_sqlite` extension module (default is no) of the `sqlite3` module.

See the `sqlite3.Connection.enable_load_extension()` method of the `sqlite3` module.

Added in version 3.6.

--disable-ipv6

Disable IPv6 support (enabled by default if supported), see the `socket` module.

--enable-big-digits=[15|30]

Define the size in bits of Python `int` digits: 15 or 30 bits.

By default, the digit size is 30.

Define the `PYLONG_BITS_IN_DIGIT` to 15 or 30.

See `sys.int_info.bits_per_digit`.

--with-suffix=SUFFIX

Set the Python executable suffix to *SUFFIX*.

The default suffix is `.exe` on Windows and macOS (`python.exe` executable), `.js` on Emscripten node, `.html` on Emscripten browser, `.wasm` on WASI, and an empty string on other platforms (`python` executable).

Αλλάξε στην έκδοση 3.11: The default suffix on WASM platform is one of `.js`, `.html` or `.wasm`.

--with-tzpath=<list of absolute paths separated by pathsep>

Select the default time zone search path for `zoneinfo.TZPATH`. See the Compile-time configuration of the `zoneinfo` module.

Default: `/usr/share/zoneinfo:/usr/lib/zoneinfo:/usr/share/lib/zoneinfo:/etc/zoneinfo`.

See `os.pathsep` path separator.

Added in version 3.9.

--without-decimal-contextvar

Build the `_decimal` extension module using a thread-local context rather than a coroutine-local context (default), see the `decimal` module.

See `decimal.HAVE_CONTEXTVAR` and the `contextvars` module.

Added in version 3.9.

--with-dbmliborder=<list of backend names>

Override order to check db backends for the `dbm` module

A valid value is a colon (:) separated string with the backend names:

- `ndbm`;
- `gdbm`;
- `bdb`.

--without-c-locale-coercion

Disable C locale coercion to a UTF-8 based locale (enabled by default).

Don't define the `PY_COERCE_C_LOCALE` macro.

See [PYTHONCOERCECLOCALE](#) and the [PEP 538](#).

--with-platlibdir=DIRNAME

Python library directory name (default is `lib`).

Fedora and SuSE use `lib64` on 64-bit platforms.

See `sys.platlibdir`.

Added in version 3.9.

--with-wheel-pkg-dir=PATH

Directory of wheel packages used by the `ensurepip` module (none by default).

Some Linux distribution packaging policies recommend against bundling dependencies. For example, Fedora installs wheel packages in the `/usr/share/python-wheels/` directory and don't install the `ensurepip._bundled` package.

Added in version 3.10.

--with-pkg-config=[`check`|`yes`|`no`]

Whether configure should use `pkg-config` to detect build dependencies.

- `check` (default): `pkg-config` is optional
- `yes`: `pkg-config` is mandatory
- `no`: configure does not use `pkg-config` even when present

Added in version 3.11.

--enable-pystats

Turn on internal Python performance statistics gathering.

By default, statistics gathering is off. Use `python3 -X pystats` command or set `PYTHONSTATS=1` environment variable to turn on statistics gathering at Python startup.

At Python exit, dump statistics if statistics gathering was on and not cleared.

Effects:

- Add `-X pystats` command line option.
- Add `PYTHONSTATS` environment variable.
- Define the `Py_STATS` macro.
- Add functions to the `sys` module:
 - `sys._stats_on()`: Turns on statistics gathering.
 - `sys._stats_off()`: Turns off statistics gathering.
 - `sys._stats_clear()`: Clears the statistics.
 - `sys._stats_dump()`: Dump statistics to file, and clears the statistics.

The statistics will be dumped to a arbitrary (probably unique) file in `/tmp/py_stats/` (Unix) or `C:\temp\py_stats\` (Windows). If that directory does not exist, results will be printed on `stderr`.

Use `Tools/scripts/summarize_stats.py` to read the stats.

Statistics:

- Opcode:
 - Specialization: success, failure, hit, deferred, miss, deopt, failures;
 - Execution count;
 - Pair count.
- Call:
 - Inlined Python calls;
 - PyEval calls;
 - Frames pushed;
 - Frame object created;
 - Eval calls: vector, generator, legacy, function `VECTORCALL`, build class, slot, function «ex», API, method.
- Object:
 - incref and decref;
 - interpreter incref and decref;
 - allocations: all, 512 bytes, 4 kiB, big;
 - free;
 - to/from free lists;
 - dictionary materialized/dematerialized;
 - type cache;
 - optimization attempts;
 - optimization traces created/executed;
 - uops executed.

- Garbage collector:
 - Garbage collections;
 - Objects visited;
 - Objects collected.

Added in version 3.11.

--disable-gil

Enables support for running Python without the *global interpreter lock* (GIL): free threading build.

Defines the `Py_GIL_DISABLED` macro and adds "t" to `sys.abiflags`.

See [whatsnew313-free-threaded-cpython](#) for more detail.

Added in version 3.13.

--enable-experimental-jit=[no|yes|yes-off|interpreter]

Indicate how to integrate the experimental just-in-time compiler.

- no: Don't build the JIT.
- yes: Enable the JIT. To disable it at runtime, set the environment variable `PYTHON_JIT=0`.
- yes-off: Build the JIT, but disable it by default. To enable it at runtime, set the environment variable `PYTHON_JIT=1`.
- interpreter: Enable the «JIT interpreter» (only useful for those debugging the JIT itself). To disable it at runtime, set the environment variable `PYTHON_JIT=0`.

`--enable-experimental-jit=no` is the default behavior if the option is not provided, and `--enable-experimental-jit` is shorthand for `--enable-experimental-jit=yes`. See [Tools/jit/README.md](#) for more information, including how to install the necessary build-time dependencies.

Σημείωση

When building CPython with JIT enabled, ensure that your system has Python 3.11 or later installed.

Added in version 3.13.

PKG_CONFIG

Path to `pkg-config` utility.

PKG_CONFIG_LIBDIR

PKG_CONFIG_PATH

`pkg-config` options.

3.3.2 C compiler options

CC

C compiler command.

CFLAGS

C compiler flags.

CPP

C preprocessor command.

CPPFLAGS

C preprocessor flags, e.g. `-Iinclude_dir`.

3.3.3 Linker options

LDFLAGS

Linker flags, e.g. `-Llibrary_directory`.

LIBS

Libraries to pass to the linker, e.g. `-llibrary`.

MACHDEP

Name for machine-dependent library files.

3.3.4 Options for third-party dependencies

Added in version 3.11.

BZIP2_CFLAGS

BZIP2_LIBS

C compiler and linker flags to link Python to `libbz2`, used by `bz2` module, overriding `pkg-config`.

CURSES_CFLAGS

CURSES_LIBS

C compiler and linker flags for `libncurses` or `libncursesw`, used by `curses` module, overriding `pkg-config`.

GDBM_CFLAGS

GDBM_LIBS

C compiler and linker flags for `gdbm`.

LIBB2_CFLAGS

LIBB2_LIBS

C compiler and linker flags for `libb2` (BLAKE2), used by `hashlib` module, overriding `pkg-config`.

LIBEDIT_CFLAGS

LIBEDIT_LIBS

C compiler and linker flags for `libedit`, used by `readline` module, overriding `pkg-config`.

LIBFFI_CFLAGS

LIBFFI_LIBS

C compiler and linker flags for `libffi`, used by `ctypes` module, overriding `pkg-config`.

LIBMPDEC_CFLAGS

LIBMPDEC_LIBS

C compiler and linker flags for `libmpdec`, used by `decimal` module, overriding `pkg-config`.

Σημείωση

These environment variables have no effect unless `--with-system-libmpdec` is specified.

LIBLZMA_CFLAGS

LIBLZMA_LIBS

C compiler and linker flags for `liblzma`, used by `lzma` module, overriding `pkg-config`.

LIBREADLINE_CFLAGS

LIBREADLINE_LIBS

C compiler and linker flags for `libreadline`, used by `readline` module, overriding `pkg-config`.

LIBSQLITE3_CFLAGS**LIBSQLITE3_LIBS**

C compiler and linker flags for `libsqlite3`, used by `sqlite3` module, overriding `pkg-config`.

LIBUUID_CFLAGS**LIBUUID_LIBS**

C compiler and linker flags for `libuuid`, used by `uuid` module, overriding `pkg-config`.

LIBZSTD_CFLAGS**LIBZSTD_LIBS**

C compiler and linker flags for `libzstd`, used by `compression.zstd` module, overriding `pkg-config`.

Added in version 3.14.

PANEL_CFLAGS**PANEL_LIBS**

C compiler and linker flags for `PANEL`, overriding `pkg-config`.

C compiler and linker flags for `libpanel` or `libpanelw`, used by `curses.panel` module, overriding `pkg-config`.

TCLTK_CFLAGS**TCLTK_LIBS**

C compiler and linker flags for `TCLTK`, overriding `pkg-config`.

ZLIB_CFLAGS**ZLIB_LIBS**

C compiler and linker flags for `libzlib`, used by `gzip` module, overriding `pkg-config`.

3.3.5 WebAssembly Options

--enable-wasm-dynamic-linking

Turn on dynamic linking support for WASM.

Dynamic linking enables `dlopen`. File size of the executable increases due to limited dead code elimination and additional features.

Added in version 3.11.

--enable-wasm-pthreads

Turn on pthreads support for WASM.

Added in version 3.11.

3.3.6 Install Options

--prefix=PREFIX

Install architecture-independent files in `PREFIX`. On Unix, it defaults to `/usr/local`.

This value can be retrieved at runtime using `sys.prefix`.

As an example, one can use `--prefix="$HOME/.local/"` to install a Python in its home directory.

--exec-prefix=EPREFIX

Install architecture-dependent files in EPREFIX, defaults to `--prefix`.

This value can be retrieved at runtime using `sys.exec_prefix`.

--disable-test-modules

Don't build nor install test modules, like the `test` package or the `_testcapi` extension module (built and installed by default).

Added in version 3.10.

--with-ensurepip=[upgrade|install|no]

Select the ensurepip command run on Python installation:

- `upgrade` (default): run `python -m ensurepip --altinstall --upgrade command`.
- `install`: run `python -m ensurepip --altinstall command`;
- `no`: don't run ensurepip;

Added in version 3.6.

3.3.7 Performance options

Configuring Python using `--enable-optimizations --with-lto` (PGO + LTO) is recommended for best performance. The experimental `--enable-bolt` flag can also be used to improve performance.

--enable-optimizations

Enable Profile Guided Optimization (PGO) using `PROFILE_TASK` (disabled by default).

The C compiler Clang requires `llvm-profdata` program for PGO. On macOS, GCC also requires it: GCC is just an alias to Clang on macOS.

Disable also semantic interposition in libpython if `--enable-shared` and GCC is used: add `-fno-semantic-interposition` to the compiler and linker flags.

i Σημείωση

During the build, you may encounter compiler warnings about profile data not being available for some source files. These warnings are harmless, as only a subset of the code is exercised during profile data acquisition. To disable these warnings on Clang, manually suppress them by adding `-Wno-profile-instr-unprofiled` to `CFLAGS`.

Added in version 3.6.

Αλλάξε στην έκδοση 3.10: Use `-fno-semantic-interposition` on GCC.

PROFILE_TASK

Environment variable used in the Makefile: Python command line arguments for the PGO generation task.

Default: `-m test --pgo --timeout=$(TESTTIMEOUT)`.

Added in version 3.8.

Αλλάξε στην έκδοση 3.13: Task failure is no longer ignored silently.

--with-lto=[full|thin|no|yes]

Enable Link Time Optimization (LTO) in any build (disabled by default).

The C compiler Clang requires `llvm-ar` for LTO (`ar` on macOS), as well as an LTO-aware linker (`ld.gold` or `lld`).

Added in version 3.6.

Added in version 3.11: To use ThinLTO feature, use `--with-lto=thin` on Clang.

Αλλάξε στην έκδοση 3.12: Use ThinLTO as the default optimization policy on Clang if the compiler accepts the flag.

--enable-bolt

Enable usage of the [BOLT post-link binary optimizer](#) (disabled by default).

BOLT is part of the LLVM project but is not always included in their binary distributions. This flag requires that `llvm-bolt` and `merge-fdata` are available.

BOLT is still a fairly new project so this flag should be considered experimental for now. Because this tool operates on machine code its success is dependent on a combination of the build environment + the other optimization configure args + the CPU architecture, and not all combinations are supported. BOLT versions before LLVM 16 are known to crash BOLT under some scenarios. Use of LLVM 16 or newer for BOLT optimization is strongly encouraged.

The `BOLT_INSTRUMENT_FLAGS` and `BOLT_APPLY_FLAGS` **configure** variables can be defined to override the default set of arguments for `llvm-bolt` to instrument and apply BOLT data to binaries, respectively.

Added in version 3.12.

BOLT_APPLY_FLAGS

Arguments to `llvm-bolt` when creating a [BOLT optimized binary](#).

Added in version 3.12.

BOLT_INSTRUMENT_FLAGS

Arguments to `llvm-bolt` when instrumenting binaries.

Added in version 3.12.

--with-computed-gotos

Enable computed gotos in evaluation loop (enabled by default on supported compilers).

--with-tail-call-interp

Enable interpreters using tail calls in CPython. If enabled, enabling PGO ([--enable-optimizations](#)) is highly recommended. This option specifically requires a C compiler with proper tail call support, and the [preserve_none](#) calling convention. For example, Clang 19 and newer supports this feature.

Added in version 3.14.

--without-mimalloc

Disable the fast mimalloc allocator (enabled by default).

See also [PYTHONMALLOC](#) environment variable.

--without-pymalloc

Disable the specialized Python memory allocator pymalloc (enabled by default).

See also [PYTHONMALLOC](#) environment variable.

--without-doc-strings

Disable static documentation strings to reduce the memory footprint (enabled by default). Documentation strings defined in Python are not affected.

Don't define the `WITH_DOC_STRINGS` macro.

See the `PyDoc_STRVAR()` macro.

--enable-profiling

Enable C-level code profiling with `gprof` (disabled by default).

--with-strict-overflow

Add `-fstrict-overflow` to the C compiler flags (by default we add `-fno-strict-overflow` instead).

--without-remote-debug

Deactivate remote debugging support described in **PEP 768** (enabled by default). When this flag is provided the code that allows the interpreter to schedule the execution of a Python file in a separate process as described in **PEP 768** is not compiled. This includes both the functionality to schedule code to be executed and the functionality to receive code to be executed.

Py_REMOTE_DEBUG

This macro is defined by default, unless Python is configured with *--without-remote-debug*.

Note that even if the macro is defined, remote debugging may not be available (for example, on an incompatible platform).

Added in version 3.14.

3.3.8 Python Debug Build

A debug build is Python built with the *--with-pydebug* configure option.

Effects of a debug build:

- Display all warnings by default: the list of default warning filters is empty in the warnings module.
- Add `d` to `sys.abiflags`.
- Add `sys.gettotalrefcount()` function.
- Add *-X showrefcount* command line option.
- Add *-d* command line option and *PYTHONDEBUG* environment variable to debug the parser.
- Add support for the `__lltrace__` variable: enable low-level tracing in the bytecode evaluation loop if the variable is defined.
- Install debug hooks on memory allocators to detect buffer overflow and other memory errors.
- Define `Py_DEBUG` and `Py_REF_DEBUG` macros.
- Add runtime checks: code surrounded by `#ifdef Py_DEBUG` and `#endif`. Enable `assert(...)` and `_PyObject_ASSERT(...)` assertions: don't set the `NDEBUG` macro (see also the *--with-assertions* configure option). Main runtime checks:
 - Add sanity checks on the function arguments.
 - Unicode and int objects are created with their memory filled with a pattern to detect usage of uninitialized objects.
 - Ensure that functions which can clear or replace the current exception are not called with an exception raised.
 - Check that deallocator functions don't change the current exception.
 - The garbage collector (`gc.collect()` function) runs some basic checks on objects consistency.
 - The `Py_SAFE_DOWNCAST()` macro checks for integer underflow and overflow when downcasting from wide types to narrow types.

See also the Python Development Mode and the *--with-trace-refs* configure option.

Αλλάξε στην έκδοση 3.8: Release builds and debug builds are now ABI compatible: defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro (see the *--with-trace-refs* option).

3.3.9 Debug options

--with-pydebug

Build Python in debug mode: define the `Py_DEBUG` macro (disabled by default).

--with-trace-refs

Enable tracing references for debugging purpose (disabled by default).

Effects:

- Define the `Py_TRACE_REFS` macro.
- Add `sys.getobjects()` function.
- Add `PYTHONDUMPREFS` environment variable.

The `PYTHONDUMPREFS` environment variable can be used to dump objects and reference counts still alive at Python exit.

Statically allocated objects are not traced.

Added in version 3.8.

Άλλαξε στην έκδοση 3.13: This build is now ABI compatible with release build and *debug build*.

--with-assertions

Build with C assertions enabled (default is no): `assert(...);` and `_PyObject_ASSERT(...);`.

If set, the `NDEBUG` macro is not defined in the `OPT` compiler variable.

See also the `--with-pydebug` option (*debug build*) which also enables assertions.

Added in version 3.6.

--with-valgrind

Enable Valgrind support (default is no).

--with-dtrace

Enable DTrace support (default is no).

See Instrumenting CPython with DTrace and SystemTap.

Added in version 3.6.

--with-address-sanitizer

Enable AddressSanitizer memory error detector, `asan` (default is no). To improve ASan detection capabilities you may also want to combine this with `--without-pymalloc` to disable the specialized small-object allocator whose allocations are not tracked by ASan.

Added in version 3.6.

--with-memory-sanitizer

Enable MemorySanitizer allocation error detector, `msan` (default is no).

Added in version 3.6.

--with-undefined-behavior-sanitizer

Enable UndefinedBehaviorSanitizer undefined behaviour detector, `ubsan` (default is no).

Added in version 3.6.

--with-thread-sanitizer

Enable ThreadSanitizer data race detector, `tsan` (default is no).

Added in version 3.13.

3.3.10 Linker options

--enable-shared

Enable building a shared Python library: `libpython` (default is no).

--without-static-libpython

Do not build `libpythonMAJOR.MINOR.a` and do not install `python.o` (built and enabled by default).
Added in version 3.10.

3.3.11 Libraries options

--with-libs='lib1 ...'

Link against additional libraries (default is no).

--with-system-expat

Build the `pyexpat` module using an installed `expat` library (default is no).

--with-system-libmpdec

Build the `_decimal` extension module using an installed `mpdecimal` library, see the `decimal` module (default is yes).

Added in version 3.3.

Άλλαξε στην έκδοση 3.13: Default to using the installed `mpdecimal` library.

Deprecated since version 3.13, will be removed in version 3.15: A copy of the `mpdecimal` library sources will no longer be distributed with Python 3.15.

 Δείτε επίσης

`LIBMPDEC_CFLAGS` and `LIBMPDEC_LIBS`.

--with-readline=readline|editline

Designate a backend library for the `readline` module.

- `readline`: Use `readline` as the backend.
- `editline`: Use `editline` as the backend.

Added in version 3.10.

--without-readline

Don't build the `readline` module (built by default).

Don't define the `HAVE_LIBREADLINE` macro.

Added in version 3.10.

--with-libm=STRING

Override `libm` math library to *STRING* (default is system-dependent).

--with-libc=STRING

Override `libc` C library to *STRING* (default is system-dependent).

--with-openssl=DIR

Root of the OpenSSL directory.

Added in version 3.7.

--with-openssl-rpath=[no|auto|DIR]

Set runtime library directory (rpath) for OpenSSL libraries:

- `no` (default): don't set rpath;
- `auto`: auto-detect rpath from `--with-openssl` and `pkg-config`;
- *DIR*: set an explicit rpath.

Added in version 3.10.

3.3.12 Security Options

--with-hash-algorithm=[fnv|siphhash13|siphhash24]

Select hash algorithm for use in Python/pyhash.c:

- siphhash13 (default);
- siphhash24;
- fnv.

Added in version 3.4.

Added in version 3.11: siphhash13 is added and it is the new default.

--with-builtin-hashlib-hashes=md5,sha1,sha256,sha512,sha3,blake2

Built-in hash modules:

- md5;
- sha1;
- sha256;
- sha512;
- sha3 (with shake);
- blake2.

Added in version 3.9.

--with-ssl-default-suites=[python|openssl|STRING]

Override the OpenSSL default cipher suites string:

- python (default): use Python's preferred selection;
- openssl: leave OpenSSL's defaults untouched;
- *STRING*: use a custom string

See the `ssl` module.

Added in version 3.7.

Άλλαξε στην έκδοση 3.10: The settings `python` and *STRING* also set TLS 1.2 as minimum protocol version.

--disable-safety

Disable compiler options that are [recommended by OpenSSF](#) for security reasons with no performance overhead. If this option is not enabled, CPython will be built based on safety compiler options with no slow down. When this option is enabled, CPython will not be built with the compiler options listed below.

The following compiler options are disabled with `--disable-safety`:

- [-fstack-protector-strong](#): Enable run-time checks for stack-based buffer overflows.
- [-Wtrampolines](#): Enable warnings about trampolines that require executable stacks.

Added in version 3.14.

--enable-slower-safety

Enable compiler options that are [recommended by OpenSSF](#) for security reasons which require overhead. If this option is not enabled, CPython will not be built based on safety compiler options which performance impact. When this option is enabled, CPython will be built with the compiler options listed below.

The following compiler options are enabled with `--enable-slower-safety`:

- [-D_FORTIFY_SOURCE=3](#): Fortify sources with compile- and run-time checks for unsafe libc usage and buffer overflows.

Added in version 3.14.

3.3.13 macOS Options

See [Mac/README.rst](#).

--enable-universalsdk

--enable-universalsdk=SDKDIR

Create a universal binary build. *SDKDIR* specifies which macOS SDK should be used to perform the build (default is no).

--enable-framework

--enable-framework=INSTALLDIR

Create a Python.framework rather than a traditional Unix install. Optional *INSTALLDIR* specifies the installation path (default is no).

--with-universal-archs=ARCH

Specify the kind of universal binary that should be created. This option is only valid when *--enable-universalsdk* is set.

Options:

- universal2 (x86-64 and arm64);
- 32-bit (PPC and i386);
- 64-bit (PPC64 and x86-64);
- 3-way (i386, PPC and x86-64);
- intel (i386 and x86-64);
- intel-32 (i386);
- intel-64 (x86-64);
- all (PPC, i386, PPC64 and x86-64).

Note that values for this configuration item are *not* the same as the identifiers used for universal binary wheels on macOS. See the Python Packaging User Guide for details on the [packaging platform compatibility tags used on macOS](#)

--with-framework-name=FRAMEWORK

Specify the name for the python framework on macOS only valid when *--enable-framework* is set (default: Python).

--with-app-store-compliance

--with-app-store-compliance=PATCH-FILE

The Python standard library contains strings that are known to trigger automated inspection tool errors when submitted for distribution by the macOS and iOS App Stores. If enabled, this option will apply the list of patches that are known to correct app store compliance. A custom patch file can also be specified. This option is disabled by default.

Added in version 3.13.

3.3.14 iOS Options

See [iOS/README.rst](#).

--enable-framework=INSTALLDIR

Create a Python.framework. Unlike macOS, the *INSTALLDIR* argument specifying the installation path is mandatory.

--with-framework-name=FRAMEWORK

Specify the name for the framework (default: Python).

3.3.15 Cross Compiling Options

Cross compiling, also known as cross building, can be used to build Python for another CPU architecture or platform. Cross compiling requires a Python interpreter for the build platform. The version of the build Python must match the version of the cross compiled host Python.

--build=BUILD

configure for building on BUILD, usually guessed by `config.guess`.

--host=HOST

cross-compile to build programs to run on HOST (target platform)

--with-build-python=path/to/python

path to build python binary for cross compiling

Added in version 3.11.

CONFIG_SITE=file

An environment variable that points to a file with configure overrides.

Example *config.site* file:

```
# config.site-aarch64
ac_cv_buggy_getaddrinfo=no
ac_cv_file__dev_ptmx=yes
ac_cv_file__dev_ptc=no
```

HOSTRUNNER

Program to run CPython for the host platform for cross-compilation.

Added in version 3.11.

Cross compiling example:

```
CONFIG_SITE=config.site-aarch64 ../configure \
--build=x86_64-pc-linux-gnu \
--host=aarch64-unknown-linux-gnu \
--with-build-python=../x86_64/python
```

3.4 Python Build System

3.4.1 Main files of the build system

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (created by `configure`);
- `pyconfig.h` (created by `configure`);
- `Modules/Setup`: C extensions built by the `Makefile` using `Module/makesetup` shell script;

3.4.2 Main build steps

- C files (`.c`) are built as object files (`.o`).
- A static `libpython` library (`.a`) is created from objects files.
- `python.o` and the static `libpython` library are linked into the final `python` program.
- C extensions are built by the `Makefile` (see `Modules/Setup`).

3.4.3 Main Makefile targets

make

For the most part, when rebuilding after editing some code or refreshing your checkout from upstream, all you need to do is execute `make`, which (per Make's semantics) builds the default target, the first one defined in the Makefile. By tradition (including in the CPython project) this is usually the `all` target. The `configure` script expands an `autoconf` variable, `@DEF_MAKE_ALL_RULE@` to describe precisely which targets `make all` will build. The three choices are:

- `profile-opt` (configured with `--enable-optimizations`)
- `build_wasm` (chosen if the host platform matches `wasm32-wasi*` or `wasm32-emscripten`)
- `build_all` (configured without explicitly using either of the others)

Depending on the most recent source file changes, Make will rebuild any targets (object files and executables) deemed out-of-date, including running `configure` again if necessary. Source/target dependencies are many and maintained manually however, so Make sometimes doesn't have all the information necessary to correctly detect all targets which need to be rebuilt. Depending on which targets aren't rebuilt, you might experience a number of problems. If you have build or test problems which you can't otherwise explain, `make clean && make` should work around most dependency problems, at the expense of longer build times.

make platform

Build the `python` program, but don't build the standard library extension modules. This generates a file named `platform` which contains a single line describing the details of the build platform, e.g., `macosx-14.3-arm64-3.12` or `linux-x86_64-3.13`.

make profile-opt

Build Python using profile-guided optimization (PGO). You can use the `configure` `--enable-optimizations` option to make this the default target of the `make` command (`make all` or just `make`).

make clean

Remove built files.

make distclean

In addition to the work done by `make clean`, remove files created by the `configure` script. `configure` will have to be run before building again.¹

make install

Build the `all` target and install Python.

make test

Build the `all` target and run the Python test suite with the `--fast-ci` option without GUI tests. Variables:

- `TESTOPTS`: additional `regtest` command-line options.
- `TESTPYTHONOPTS`: additional Python command-line options.
- `TESTTIMEOUT`: timeout in seconds (default: 10 minutes).

¹ `git clean -fdx` is an even more extreme way to «clean» your checkout. It removes all files not known to Git. When bug hunting using `git bisect`, this is [recommended between probes](#) to guarantee a completely clean build. Use **with care**, as it will delete all files not checked into Git, including your new, uncommitted work.

make ci

This is similar to `make test`, but uses the `-ugui` to also run GUI tests.

Added in version 3.14.

make buildbottest

This is similar to `make test`, but uses the `--slow-ci` option and default timeout of 20 minutes, instead of `--fast-ci` option.

make regen-all

Regenerate (almost) all generated files. These include (but are not limited to) bytecode cases, and parser generator file. `make regen-stdlib-module-names` and `autoconf` must be run separately for the remaining *generated files*.

3.4.4 C extensions

Some C extensions are built as built-in modules, like the `sys` module. They are built with the `Py_BUILD_CORE_BUILTIN` macro defined. Built-in modules have no `__file__` attribute:

```
>>> import sys
>>> sys
<module 'sys' (built-in)>
>>> sys.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'sys' has no attribute '__file__'
```

Other C extensions are built as dynamic libraries, like the `_asyncio` module. They are built with the `Py_BUILD_CORE_MODULE` macro defined. Example on Linux x86-64:

```
>>> import _asyncio
>>> _asyncio
<module '_asyncio' from '/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-
↳ 39-x86_64-linux-gnu.so'>
>>> _asyncio.__file__
'/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_64-linux-gnu.so'
```

`Modules/Setup` is used to generate Makefile targets to build C extensions. At the beginning of the files, C extensions are built as built-in modules. Extensions defined after the `*shared*` marker are built as dynamic libraries.

The `PyAPI_FUNC()`, `PyAPI_DATA()` and `PyMODINIT_FUNC` macros of `Include/exports.h` are defined differently depending if the `Py_BUILD_CORE_MODULE` macro is defined:

- Use `Py_EXPORTED_SYMBOL` if the `Py_BUILD_CORE_MODULE` is defined
- Use `Py_IMPORTED_SYMBOL` otherwise.

If the `Py_BUILD_CORE_BUILTIN` macro is used by mistake on a C extension built as a shared library, its `PyInit_xxx()` function is not exported, causing an `ImportError` on import.

3.5 Compiler and linker flags

Options set by the `./configure` script and environment variables and used by Makefile.

3.5.1 Preprocessor flags

CONFIGURE_CPPFLAGS

Value of *CPPFLAGS* variable passed to the `./configure` script.

Added in version 3.6.

CPPFLAGS

(Objective) C/C++ preprocessor flags, e.g. `-Iinclude_dir` if you have headers in a nonstandard directory *include_dir*.

Both *CPPFLAGS* and *LDLIBRARY* need to contain the shell's value to be able to build extension modules using the directories specified in the environment variables.

BASECPPFLAGS

Added in version 3.4.

PY_CPPFLAGS

Extra preprocessor flags added for building the interpreter object files.

Default: `$(BASECPPFLAGS) -I. -I$(srcdir)/Include $(CONFIGURE_CPPFLAGS) $(CPPFLAGS)`.

Added in version 3.2.

3.5.2 Compiler flags

CC

C compiler command.

Example: `gcc -pthread`.

CXX

C++ compiler command.

Example: `g++ -pthread`.

CFLAGS

C compiler flags.

CFLAGS_NODIST

CFLAGS_NODIST is used for building the interpreter and stdlib C extensions. Use it when a compiler flag should *not* be part of *CFLAGS* once Python is installed ([gh-65320](#)).

In particular, *CFLAGS* should not contain:

- the compiler flag `-I` (for setting the search path for include files). The `-I` flags are processed from left to right, and any flags in *CFLAGS* would take precedence over user- and package-supplied `-I` flags.
- hardening flags such as `-Werror` because distributions cannot control whether packages installed by users conform to such heightened standards.

Added in version 3.5.

COMPILEALL_OPTS

Options passed to the `compileall` command line when building PYC files in `make install`. Default: `-j0`.

Added in version 3.12.

EXTRA_CFLAGS

Extra C compiler flags.

CONFIGURE_CFLAGS

Value of `CFLAGS` variable passed to the `./configure` script.

Added in version 3.2.

CONFIGURE_CFLAGS_NODIST

Value of `CFLAGS_NODIST` variable passed to the `./configure` script.

Added in version 3.5.

BASECFLAGS

Base compiler flags.

OPT

Optimization flags.

CFLAGS_ALIASING

Strict or non-strict aliasing flags used to compile `Python/dtoa.c`.

Added in version 3.7.

CCSHARED

Compiler flags used to build a shared library.

For example, `-fPIC` is used on Linux and on BSD.

CFLAGSFORSHARED

Extra C flags added for building the interpreter object files.

Default: `$(CCSHARED)` when `--enable-shared` is used, or an empty string otherwise.

PY_CFLAGS

Default: `$(BASECFLAGS) $(OPT) $(CONFIGURE_CFLAGS) $(CFLAGS) $(EXTRA_CFLAGS)`.

PY_CFLAGS_NODIST

Default: `$(CONFIGURE_CFLAGS_NODIST) $(CFLAGS_NODIST) -I$(srcdir)/Include/internal`.

Added in version 3.5.

PY_STDMODULE_CFLAGS

C flags used for building the interpreter object files.

Default: `$(PY_CFLAGS) $(PY_CFLAGS_NODIST) $(PY_CPPFLAGS) $(CFLAGSFORSHARED)`.

Added in version 3.7.

PY_CORE_CFLAGS

Default: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`.

Added in version 3.2.

PY_BUILTIN_MODULE_CFLAGS

Compiler flags to build a standard library extension module as a built-in module, like the `posix` module.

Default: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE_BUILTIN`.

Added in version 3.8.

PURIFY

Purify command. Purify is a memory debugger program.

Default: empty string (not used).

3.5.3 Linker flags

LINKCC

Linker command used to build programs like `python` and `_testembed`.

Default: `$(PURIFY) $(CC)`.

CONFIGURE_LDFLAGS

Value of `LD_FLAGS` variable passed to the `./configure` script.

Avoid assigning `C_FLAGS`, `LD_FLAGS`, etc. so users can use them on the command line to append to these values without stomping the pre-set values.

Added in version 3.2.

LD_FLAGS_NODIST

`LD_FLAGS_NODIST` is used in the same manner as `C_FLAGS_NODIST`. Use it when a linker flag should *not* be part of `LD_FLAGS` once Python is installed ([gh-65320](#)).

In particular, `LD_FLAGS` should not contain:

- the compiler flag `-L` (for setting the search path for libraries). The `-L` flags are processed from left to right, and any flags in `LD_FLAGS` would take precedence over user- and package-supplied `-L` flags.

CONFIGURE_LDFLAGS_NODIST

Value of `LD_FLAGS_NODIST` variable passed to the `./configure` script.

Added in version 3.8.

LD_FLAGS

Linker flags, e.g. `-Llib_dir` if you have libraries in a nonstandard directory `lib_dir`.

Both `CPP_FLAGS` and `LD_FLAGS` need to contain the shell's value to be able to build extension modules using the directories specified in the environment variables.

LIBS

Linker flags to pass libraries to the linker when linking the Python executable.

Example: `-lrt`.

LD_SHARED

Command to build a shared library.

Default: `@LD_SHARED@ $(PY_LD_FLAGS)`.

BLD_SHARED

Command to build `libpython` shared library.

Default: `@BLD_SHARED@ $(PY_CORE_LD_FLAGS)`.

PY_LD_FLAGS

Default: `$(CONFIGURE_LD_FLAGS) $(LD_FLAGS)`.

PY_LD_FLAGS_NODIST

Default: `$(CONFIGURE_LD_FLAGS_NODIST) $(LD_FLAGS_NODIST)`.

Added in version 3.8.

PY_CORE_LD_FLAGS

Linker flags used for building the interpreter object files.

Added in version 3.8.

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.

Unlike most Unix systems and services, Windows does not include a system supported installation of Python. Instead, Python can be obtained from a number of distributors, including directly from the CPython team. Each Python distribution will have its own benefits and drawbacks, however, consistency with other tools you are using is generally a worthwhile benefit. Before committing to the process described here, we recommend investigating your existing tools to see if they can provide Python directly.

To obtain Python from the CPython team, use the Python Install Manager. This is a standalone tool that makes Python available as global commands on your Windows machine, integrates with the system, and supports updates over time. You can download the Python Install Manager from python.org/downloads or through the [Microsoft Store app](#).

Once you have installed the Python Install Manager, the global `python` command can be used from any terminal to launch your current latest version of Python. This version may change over time as you add or remove different versions, and the `py list` command will show which is current.

In general, we recommend that you create a virtual environment for each project and run `<env>\Scripts\Activate` in your terminal to use it. This provides isolation between projects, consistency over time, and ensures that additional commands added by packages are also available in your session. Create a virtual environment using `python -m venv <env path>`.

If the `python` or `py` commands do not seem to be working, please see the [Troubleshooting](#) section below. There are sometimes additional manual steps required to configure your PC.

Apart from using the Python install manager, Python can also be obtained as NuGet packages. See [The nuget.org packages](#) below for more information on these packages.

The embeddable distros are minimal packages of Python suitable for embedding into larger applications. They can be installed using the Python install manager. See [The embeddable package](#) below for more information on these packages.

4.1 Python Install Manager

4.1.1 Installation

The Python install manager can be installed from the [Microsoft Store app](#) or downloaded and installed from python.org/downloads. The two versions are identical.

To install through the Store, simply click «Install». After it has completed, open a terminal and type `python` to get started.

To install the file downloaded from `python.org`, either double-click and select «Install», or run `Add-AppxPackage <path to MSIX>` in Windows Powershell.

After installation, the `python`, `py`, and `pymanager` commands should be available. If you have existing installations of Python, or you have modified your `PATH` variable, you may need to remove them or undo the modifications. See [Troubleshooting](#) for more help with fixing non-working commands.

When you first install a runtime, you will likely be prompted to add a directory to your `PATH`. This is optional, if you prefer to use the `py` command, but is offered for those who prefer the full range of aliases (such as `python3.14.exe`) to be available. The directory will be `%LocalAppData%\Python\bin` by default, but may be customized by an administrator. Click Start and search for «Edit environment variables for your account» for the system settings page to add the path.

Each Python runtime you install will have its own directory for scripts. These also need to be added to `PATH` if you want to use them.

The Python install manager will be automatically updated to new releases. This does not affect any installs of Python runtimes. Uninstalling the Python install manager does not uninstall any Python runtimes.

If you are not able to install an MSIX in your context, for example, you are using automated deployment software that does not support it, or are targeting Windows Server 2019, please see [Advanced Installation](#) below for more information.

4.1.2 Basic Use

The recommended command for launching Python is `python`, which will either launch the version requested by the script being launched, an active virtual environment, or the default installed version, which will be the latest stable release unless configured otherwise. If no version is specifically requested and no runtimes are installed at all, the current latest release will be installed automatically.

For all scenarios involving multiple runtime versions, the recommended command is `py`. This may be used anywhere in place of `python` or the older `py.exe` launcher. By default, `py` matches the behaviour of `python`, but also allows command line options to select a specific version as well as subcommands to manage installations. These are detailed below.

Because the `py` command may already be taken by the previous version, there is also an unambiguous `pymanager` command. Scripted installs that are intending to use Python install manager should consider using `pymanager`, due to the lower chance of encountering a conflict with existing installs. The only difference between the two commands is when running without any arguments: `py` will install and launch your default interpreter, while `pymanager` will display help (`pymanager exec ...` provides equivalent behaviour to `py ...`).

Each of these commands also has a windowed version that avoids creating a console window. These are `pyw`, `pythonw` and `pymanagerw`. A `python3` command is also included that mimics the `python` command. It is intended to catch accidental uses of the typical POSIX command on Windows, but is not meant to be widely used or recommended.

To launch your default runtime, run `python` or `py` with the arguments you want to be passed to the runtime (such as script files or the module to launch):

```
$> py
...
$> python my-script.py
...
$> py -m this
...
```

The default runtime can be overridden with the `PYTHON_MANAGER_DEFAULT` environment variable, or a configuration file. See [Configuration](#) for information about configuration settings.

To launch a specific runtime, the `py` command accepts a `-V:<TAG>` option. This option must be specified before any others. The tag is part or all of the identifier for the runtime; for those from the CPython team, it looks like the

version, potentially with the platform. For compatibility, the `V:` may be omitted in cases where the tag refers to an official release and starts with 3.

```
$> py -V:3.14 ...
$> py -V:3-arm64 ...
```

Runtimes from other distributors may require the *company* to be included as well. This should be separated from the tag by a slash, and may be a prefix. Specifying the company is optional when it is `PythonCore`, and specifying the tag is optional (but not the slash) when you want the latest release from a specific company.

```
$> py -V:Distributor\1.0 ...
$> py -V:distrib/ ...
```

If no version is specified, but a script file is passed, the script will be inspected for a *shebang line*. This is a special format for the first line in a file that allows overriding the command. See [Shebang lines](#) for more information. When there is no shebang line, or it cannot be resolved, the script will be launched with the default runtime.

If you are running in an active virtual environment, have not requested a particular version, and there is no shebang line, the default runtime will be that virtual environment. In this scenario, the `python` command was likely already overridden and none of these checks occurred. However, this behaviour ensures that the `py` command can be used interchangeably.

When you launch either `python` or `py` but do not have any runtimes installed, and the requested version is the default, it will be installed automatically and then launched. Otherwise, the requested version will be installed if automatic installation is configured (most likely by setting `PYTHON_MANAGER_AUTOMATIC_INSTALL` to `true`), or if the `py exec` or `pymanager exec` forms of the command were used.

4.1.3 Command Help

The `py help` command will display the full list of supported commands, along with their options. Any command may be passed the `-?` option to display its help, or its name passed to `py help`.

```
$> py help
$> py help install
$> py install /?
```

All commands support some common options, which will be shown by `py help`. These options must be specified after any subcommand. Specifying `-v` or `--verbose` will increase the amount of output shown, and `-vv` will increase it further for debugging purposes. Passing `-q` or `--quiet` will reduce output, and `-qq` will reduce it further.

The `--config=<PATH>` option allows specifying a configuration file to override multiple settings at once. See [Configuration](#) below for more information about these files.

4.1.4 Listing Runtimes

```
$> py list [-f|--format=<FMT>] [-1|--one] [--online|-s|--source=<URL>] [
  ↳<TAG>...]
```

The list of installed runtimes can be seen using `py list`. A filter may be added in the form of one or more tags (with or without company specifier), and each may include a `<`, `<=`, `>=` or `>` prefix to restrict to a range.

A range of formats are supported, and can be passed as the `--format=<FMT>` or `-f <FMT>` option. Formats include `table` (a user friendly table view), `csv` (comma-separated table), `json` (a single JSON blob), `jsonl` (one JSON blob per result), `exe` (just the executable path), `prefix` (just the prefix path).

The `--one` or `-1` option only displays a single result. If the default runtime is included, it will be the one. Otherwise, the `<best>` result is shown (`<best>` is deliberately vaguely defined, but will usually be the most recent version). The result shown by `py list --one <TAG>` will match the runtime that would be launched by `py -V:<TAG>`.

The `--only-managed` option excludes results that were not installed by the Python install manager. This is useful when determining which runtimes may be updated or uninstalled through the `py` command.

The `--online` option is short for passing `--source=<URL>` with the default source. Passing either of these options will search the online index for runtimes that can be installed. The result shown by `py list --online --one <TAG>` will match the runtime that would be installed by `py install <TAG>`.

```
$> py list --online 3.14
```

For compatibility with the old launcher, the `--list`, `--list-paths`, `-0` and `-0p` commands (e.g. `py -0p`) are retained. They do not allow additional options, and will produce legacy formatted output.

4.1.5 Installing Runtimes

```
$> py install [-s|--source=<URL>] [-f|--force] [-u|--update] [--dry-run] [
  ↳<TAG>...]
```

New runtime versions may be added using `py install`. One or more tags may be specified, and the special tag `default` may be used to select the default. Ranges are not supported for installation.

The `--source=<URL>` option allows overriding the online index that is used to obtain runtimes. This may be used with an offline index, as shown in [Offline Installs](#).

Passing `--force` will ignore any cached files and remove any existing install to replace it with the specified one.

Passing `--update` will replace existing installs if the new version is newer. Otherwise, they will be left. If no tags are provided with `--update`, all installs managed by the Python install manager will be updated if newer versions are available. Updates will remove any modifications made to the install, including globally installed packages, but virtual environments will continue to work.

Passing `--dry-run` will generate output and logs, but will not modify any installs.

In addition to the above options, the `--target` option will extract the runtime to the specified directory instead of doing a normal install. This is useful for embedding runtimes into larger applications.

```
$> py install ... [-t|--target=<PATH>] <TAG>
```

4.1.6 Offline Installs

To perform offline installs of Python, you will need to first create an offline index on a machine that has network access.

```
$> py install --download=<PATH> ... <TAG>...
```

The `--download=<PATH>` option will download the packages for the listed tags and create a directory containing them and an `index.json` file suitable for later installation. This entire directory can be moved to the offline machine and used to install one or more of the bundled runtimes:

```
$> py install --source="<PATH>\index.json" <TAG>...
```

The Python install manager can be installed by downloading its installer and moving it to another machine before installing.

Alternatively, the ZIP files in an offline index directory can simply be transferred to another machine and extracted. This will not register the install in any way, and so it must be launched by directly referencing the executables in the extracted directory, but it is sometimes a preferable approach in cases where installing the Python install manager is not possible or convenient.

In this way, Python runtimes can be installed and managed on a machine without access to the internet.

4.1.7 Uninstalling Runtimes

```
$> py uninstall [-y|--yes] <TAG>...
```

Runtimes may be removed using the `py uninstall` command. One or more tags must be specified. Ranges are not supported here.

The `--yes` option bypasses the confirmation prompt before uninstalling.

Instead of passing tags individually, the `--purge` option may be specified. This will remove all runtimes managed by the Python install manager, including cleaning up the Start menu, registry, and any download caches. Runtimes that were not installed by the Python install manager will not be impacted, and neither will manually created configuration files.

```
$> py uninstall [-y|--yes] --purge
```

The Python install manager can be uninstalled through the Windows «Installed apps» settings page. This does not remove any runtimes, and they will still be usable, though the global `python` and `py` commands will be removed. Reinstalling the Python install manager will allow you to manage these runtimes again. To completely clean up all Python runtimes, run with `--purge` before uninstalling the Python install manager.

4.1.8 Configuration

Python install manager is configured with a hierarchy of configuration files, environment variables, command-line options, and registry settings. In general, configuration files have the ability to configure everything, including the location of other configuration files, while registry settings are administrator-only and will override configuration files. Command-line options override all other settings, but not every option is available.

This section will describe the defaults, but be aware that modified or overridden installs may resolve settings differently.

A global configuration file may be configured by an administrator, and would be read first. The user configuration file is stored at `%AppData%\Python\pymanager.json` (by default) and is read next, overwriting any settings from earlier files. An additional configuration file may be specified as the `PYTHON_MANAGER_CONFIG` environment variable or the `--config` command line option (but not both).

The following settings are those that are considered likely to be modified in normal use. Later sections list those that are intended for administrative customization.

Standard configuration options

Config Key	Environment Variable	Description
default_tag	PYTHON_MANAGE	The preferred default version to launch or install. By default, this is interpreted as the most recent non-prerelease version from the CPython team.
default_platform	PYTHON_MANAGE	The preferred default platform to launch or install. This is treated as a suffix to the specified tag, such that <code>py -V:3.14</code> would prefer an install for 3.14-64 if it exists (and <code>default_platform</code> is -64), but will use 3.14 if no tagged install exists.
logs_dir	PYTHON_MANAGE	The location where log files are written. By default, <code>%TEMP%</code> .
automatic	PYTHON_MANAGE	True to allow automatic installs when specifying a particular runtime to launch. By default, true.
include_runtimes	PYTHON_MANAGE	True to allow listing and launching runtimes that were not installed by the Python install manager, or false to exclude them. By default, true.
shebang_command	PYTHON_MANAGE	True to allow shebangs in <code>.py</code> files to launch applications other than Python runtimes, or false to prevent it. By default, true.
log_level	PYMANAGER_VERBOSE PYMANAGER_DEBUG	Set the default level of output (0-50). By default, 20. Lower values produce more output. The environment variables are boolean, and may produce additional output during startup that is later suppressed by other configuration.
confirm	PYTHON_MANAGE	True to confirm certain actions before taking them (such as uninstall), or false to skip the confirmation. By default, true.
install_source	PYTHON_MANAGE	Override the index feed to obtain new installs from.
list_format	PYTHON_MANAGE	Specify the default format used by the <code>py list</code> command. By default, <code>table</code> .

Dotted names should be nested inside JSON objects, for example, `list.format` would be specified as `{"list": {"format": "table"}}`.

4.1.9 Shebang lines

If the first line of a script file starts with `#!`, it is known as a «shebang» line. Linux and other Unix like operating systems have native support for such lines and they are commonly used on such systems to indicate how a script should be executed. The `python` and `py` commands allow the same facilities to be used with Python scripts on Windows.

To allow shebang lines in Python scripts to be portable between Unix and Windows, a number of “virtual” commands are supported to specify which interpreter to use. The supported virtual commands are:

- `/usr/bin/env <ALIAS>`
- `/usr/bin/env -S <ALIAS>`
- `/usr/bin/<ALIAS>`
- `/usr/local/bin/<ALIAS>`
- `<ALIAS>`

For example, if the first line of your script starts with

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

The default Python or an active virtual environment will be located and used. As many Python scripts written to work on Unix will already have this line, you should find these scripts can be used by the launcher without modification. If you are writing a new script on Windows which you hope will be useful on Unix, you should use one of the shebang lines starting with `/usr`.

Any of the above virtual commands can have `<ALIAS>` replaced by an alias from an installed runtime. That is, any command generated in the global aliases directory (which you may have added to your `PATH` environment variable)

can be used in a shebang, even if it is not on your `PATH`. This allows the use of shebangs like `/usr/bin/python3.12` to select a particular runtime.

If no runtimes are installed, or if automatic installation is enabled, the requested runtime will be installed if necessary. See [Configuration](#) for information about configuration settings.

The `/usr/bin/env` form of shebang line will also search the `PATH` environment variable for unrecognized commands. This corresponds to the behaviour of the Unix `env` program, which performs the same search, but prefers launching known Python commands. A warning may be displayed when searching for arbitrary executables, and this search may be disabled by the `shebang_can_run_anything` configuration option.

Shebang lines that do not match any of patterns are treated as *Windows* executable paths that are absolute or relative to the directory containing the script file. This is a convenience for Windows-only scripts, such as those generated by an installer, since the behavior is not compatible with Unix-style shells. These paths may be quoted, and may include multiple arguments, after which the path to the script and any additional arguments will be appended. This functionality may be disabled by the `shebang_can_run_anything` configuration option.

Σημείωση

The behaviour of shebangs in the Python install manager is subtly different from the previous `py.exe` launcher, and the old configuration options no longer apply. If you are specifically reliant on the old behaviour or configuration, we recommend keeping the legacy launcher. It may be [downloaded independently](#) and installed on its own. The legacy launcher's `py` command will override PyManager's one, and you will need to use `pymanager` commands for installing and uninstalling.

4.1.10 Advanced Installation

For situations where an MSIX cannot be installed, such as some older administrative distribution platforms, there is an MSI available from the [python.org](#) downloads page. This MSI has no user interface, and can only perform per-machine installs to its default location in Program Files. It will attempt to modify the system `PATH` environment variable to include this install location, but be sure to validate this on your configuration.

Σημείωση

Windows Server 2019 is the only version of Windows that CPython supports that does not support MSIX. For Windows Server 2019, you should use the MSI.

Be aware that the MSI package does not bundle any runtimes, and so is not suitable for installs into offline environments without also creating an offline install index. See [Offline Installs](#) and [Administrative Configuration](#) for information on handling these scenarios.

Runtimes installed by the MSI are shared with those installed by the MSIX, and are all per-user only. The Python install manager does not support installing runtimes per-machine. To emulate a per-machine install, you can use `py install --target=<shared location>` as administrator and add your own system-wide modifications to `PATH`, the registry, or the Start menu.

When the MSIX is installed, but commands are not available in the `PATH` environment variable, they can be found under `%LocalAppData%\Microsoft\WindowsApps\PythonSoftwareFoundation.PythonManager_3847v3x7pw1km` or `%LocalAppData%\Microsoft\WindowsApps\PythonSoftwareFoundation.PythonManager_qbz5n2kfra8p0`, depending on whether it was installed from [python.org](#) or through the Windows Store. Attempting to run the executable directly from Program Files is not recommended.

To programmatically install the Python install manager, it is easiest to use WinGet, which is included with all supported versions of Windows:

```
$> winget install 9NQ7512CXL7T -e --accept-package-agreements --disable-
↪ interactivity
```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
# Optionally run the configuration checker and accept all changes
$> py install --configure -y
```

To download the Python install manager and install on another machine, the following WinGet command will download the required files from the Store to your Downloads directory (add `-d <location>` to customize the output location). This also generates a YAML file that appears to be unnecessary, as the downloaded MSIX can be installed by launching or using the commands below.

```
$> winget download 9NQ7512CXL7T -e --skip-license --accept-package-
    agreements --accept-source-agreements
```

To programmatically install or uninstall an MSIX using only PowerShell, the `Add-AppxPackage` and `Remove-AppxPackage` PowerShell cmdlets are recommended:

```
$> Add-AppxPackage C:\Downloads\python-manager-25.0.msix
...
$> Get-AppxPackage PythonSoftwareFoundation.PythonManager | Remove-
    AppxPackage
```

The latest release can be downloaded and installed by Windows by passing the AppInstaller file to the `Add-AppxPackage` command. This installs using the MSIX on python.org, and is only recommended for cases where installing via the Store (interactively or using WinGet) is not possible.

```
$> Add-AppxPackage -AppInstallerFile https://www.python.org/ftp/python/
    pymanager/pymanager.appinstaller
```

Other tools and APIs may also be used to provision an MSIX package for all users on a machine, but Python does not consider this a supported scenario. We suggest looking into the PowerShell `Add-AppxProvisionedPackage` cmdlet, the native Windows `PackageManager` class, or the documentation and support for your deployment tool.

Regardless of the install method, users will still need to install their own copies of Python itself, as there is no way to trigger those installs without being a logged in user. When using the MSIX, the latest version of Python will be available for all users to install without network access.

Note that the MSIX downloadable from the Store and from the Python website are subtly different and cannot be installed at the same time. Wherever possible, we suggest using the above WinGet commands to download the package from the Store to reduce the risk of setting up conflicting installs. There are no licensing restrictions on the Python install manager that would prevent using the Store package in this way.

4.1.11 Administrative Configuration

There are a number of options that may be useful for administrators to override configuration of the Python install manager. These can be used to provide local caching, disable certain shortcut types, override bundled content. All of the above configuration options may be set, as well as those below.

Configuration options may be overridden in the registry by setting values under `HKEY_LOCAL_MACHINE\Software\Policies\Python\PyManager`, where the value name matches the configuration key and the value type is `REG_SZ`. Note that this key can itself be customized, but only by modifying the core config file distributed with the Python install manager. We recommend, however, that registry values are used only to set `base_config` to a JSON file containing the full set of overrides. Registry key overrides will replace any other configured setting, while `base_config` allows users to further modify settings they may need.

Note that most settings with environment variables support those variables because their default setting specifies the variable. If you override them, the environment variable will no longer work, unless you override it with another one. For example, the default value of `confirm` is literally `%PYTHON_MANAGER_CONFIRM%`, which will resolve the variable at load time. If you override the value to `yes`, then the environment variable will no longer be used. If you override the value to `%CONFIRM%`, then that environment variable will be used instead.

Configuration settings that are paths are interpreted as relative to the directory containing the configuration file that specified them.

Administrative configuration options

Config Key	Description
<code>base_config</code>	The highest priority configuration file to read. Note that only the built-in configuration file and the registry can modify this setting.
<code>user_config</code>	The second configuration file to read.
<code>additional_config</code>	The third configuration file to read.
<code>registry_override_k</code>	Registry location to check for overrides. Note that only the built-in configuration file can modify this setting.
<code>bundled_dir</code>	Read-only directory containing locally cached files.
<code>install.fallback_source</code>	Path or URL to an index to consult when the main index cannot be accessed.
<code>install.enable_shortcut_kinds</code>	Comma-separated list of shortcut kinds to allow (e.g. "pep514,start"). Enabled shortcuts may still be disabled by <code>disable_shortcut_kinds</code> .
<code>install.disable_shortcut_kinds</code>	Comma-separated list of shortcut kinds to exclude (e.g. "pep514,start"). Disabled shortcuts are not reactivated by <code>enable_shortcut_kinds</code> .
<code>pep514_root</code>	Registry location to read and write PEP 514 entries into. By default, <code>HKEY_CURRENT_USER\Software\Python</code> .
<code>start_folder</code>	Start menu folder to write shortcuts into. By default, <code>Python</code> . This path is relative to the user's Programs folder.
<code>virtual_env</code>	Path to the active virtual environment. By default, this is <code>%VIRTUAL_ENV%</code> , but may be set empty to disable venv detection.
<code>shebang_can_run_any</code>	True to suppress visible warnings when a shebang launches an application other than a Python runtime.

4.1.12 Installing Free-threaded Binaries

Added in version 3.13: (Experimental)

Σημείωση

Everything described in this section is considered experimental, and should be expected to change in future releases.

Pre-built distributions of the experimental free-threaded build are available by installing tags with the `t` suffix.

```
$> py install 3.14t
$> py install 3.14t-arm64
$> py install 3.14t-32
```

This will install and register as normal. If you have no other runtimes installed, then `python` will launch this one. Otherwise, you will need to use `py -V:3.14t ...` or, if you have added the global aliases directory to your `PATH` environment variable, the `python3.14t.exe` commands.

4.1.13 Troubleshooting

If your Python install manager does not seem to be working correctly, please work through these tests and fixes to see if it helps. If not, please report an issue at [our bug tracker](#), including any relevant log files (written to your `%TEMP%` directory by default).

Troubleshooting

Symptom	Things to try
<code>python</code> gives me a «command not found» error or opens the Store app when I type it in my terminal.	<p>Did you <i>install the Python install manager</i>?</p> <p>Click Start, open «Manage app execution aliases», and check that the aliases for «Python (default)» are enabled. If they already are, try disabling and re-enabling to refresh the command. The «Python (default windowed)» and «Python install manager» commands may also need refreshing.</p> <p>Check that the <code>py</code> and <code>pymanager</code> commands work.</p>
<code>py</code> gives me a «command not found» error when I type it in my terminal.	<p>Did you <i>install the Python install manager</i>?</p> <p>Click Start, open «Manage app execution aliases», and check that the aliases for «Python (default)» are enabled. If they already are, try disabling and re-enabling to refresh the command. The «Python (default windowed)» and «Python install manager» commands may also need refreshing.</p>
<code>py</code> gives me a «can't open file» error when I type commands in my terminal.	This usually means you have the legacy launcher installed and it has priority over the Python install manager. To remove, click Start, open «Installed apps», search for «Python launcher» and uninstall it.
<code>python</code> doesn't launch the same runtime as <code>py</code>	<p>Click Start, open «Installed apps», look for any existing Python runtimes, and either remove them or Modify and disable the <code>PATH</code> options.</p> <p>Click Start, open «Manage app execution aliases», and check that your <code>python.exe</code> alias is set to «Python (default)»</p>
<code>python</code> and <code>py</code> don't launch the runtime I expect	<p>Check your <code>PYTHON_MANAGER_DEFAULT</code> environment variable or <code>default_tag</code> configuration. The <code>py list</code> command will show your default based on these settings.</p> <p>Installs that are managed by the Python install manager will be chosen ahead of unmanaged installs. Use <code>py install</code> to install the runtime you expect, or configure your default tag.</p> <p>Prerelease and experimental installs that are not managed by the Python install manager may be chosen ahead of stable releases. Configure your default tag or uninstall the prerelease runtime and reinstall using <code>py install</code>.</p>
<code>pythonw</code> or <code>pyw</code> don't launch the same runtime as <code>python</code> or <code>py</code>	Click Start, open «Manage app execution aliases», and check that your <code>pythonw.exe</code> and <code>pyw.exe</code> aliases are consistent with your others.
<code>pip</code> gives me a «command not found» error when I type it in my terminal.	<p>Have you activated a virtual environment? Run the <code>.venv\Scripts\activate</code> script in your terminal to activate.</p> <p>The package may be available but missing the generated executable. We recommend using the <code>python -m pip</code> command instead, or alternatively the <code>python -m pip install --force pip</code> command will recreate the executables and show you the path to add to <code>PATH</code>. These scripts are separated for each runtime, and so you may need to add multiple paths.</p>

4.2 The embeddable package

Added in version 3.5.

The embedded distribution is a ZIP file containing a minimal Python environment. It is intended for acting as part of another application, rather than being directly accessed by end-users.

To install an embedded distribution, we recommend using `py install` with the `--target` option:

```
$> py install 3.14-embed --target=runtime
```

When extracted, the embedded distribution is (almost) fully isolated from the user's system, including environment variables, system registry settings, and installed packages. The standard library is included as pre-compiled and optimized `.pyc` files in a ZIP, and `python3.dll`, `python313.dll`, `python.exe` and `pythonw.exe` are all provided. Tcl/tk (including all dependents, such as Idle), pip and the Python documentation are not included.

A default `._pth` file is included, which further restricts the default search paths (as described below in [Finding modules](#)). This file is intended for embedders to modify as necessary.

Third-party packages should be installed by the application installer alongside the embedded distribution. Using pip to manage dependencies as for a regular Python installation is not supported with this distribution, though with some care it may be possible to include and use pip for automatic updates. In general, third-party packages should be treated as part of the application («vendoring») so that the developer can ensure compatibility with newer versions before providing updates to users.

The two recommended use cases for this distribution are described below.

4.2.1 Python Application

An application written in Python does not necessarily require users to be aware of that fact. The embedded distribution may be used in this case to include a private version of Python in an install package. Depending on how transparent it should be (or conversely, how professional it should appear), there are two options.

Using a specialized executable as a launcher requires some coding, but provides the most transparent experience for users. With a customized launcher, there are no obvious indications that the program is running on Python: icons can be customized, company and version information can be specified, and file associations behave properly. In most cases, a custom launcher should simply be able to call `Py_Main` with a hard-coded command line.

The simpler approach is to provide a batch file or generated shortcut that directly calls the `python.exe` or `pythonw.exe` with the required command-line arguments. In this case, the application will appear to be Python and not its actual name, and users may have trouble distinguishing it from other running Python processes or file associations.

With the latter approach, packages should be installed as directories alongside the Python executable to ensure they are available on the path. With the specialized launcher, packages can be located in other locations as there is an opportunity to specify the search path before launching the application.

4.2.2 Embedding Python

Applications written in native code often require some form of scripting language, and the embedded Python distribution can be used for this purpose. In general, the majority of the application is in native code, and some part will either invoke `python.exe` or directly use `python3.dll`. For either case, extracting the embedded distribution to a subdirectory of the application installation is sufficient to provide a loadable Python interpreter.

As with the application use, packages can be installed to any location as there is an opportunity to specify search paths before initializing the interpreter. Otherwise, there is no fundamental differences between using the embedded distribution and a regular installation.

4.3 The nuget.org packages

Added in version 3.5.2.

The nuget.org package is a reduced size Python environment intended for use on continuous integration and build systems that do not have a system-wide install of Python. While nuget is «the package manager for .NET», it also works perfectly fine for packages containing build-time tools.

Visit nuget.org for the most up-to-date information on using nuget. What follows is a summary that is sufficient for Python developers.

The `nuget.exe` command line tool may be downloaded directly from <https://aka.ms/nugetclidl>, for example, using curl or PowerShell. With the tool, the latest version of Python for 64-bit or 32-bit machines is installed using:

```
nuget.exe install python -ExcludeVersion -OutputDirectory .
nuget.exe install pythonx86 -ExcludeVersion -OutputDirectory .
```

To select a particular version, add a `-Version 3.x.y`. The output directory may be changed from `.`, and the package will be installed into a subdirectory. By default, the subdirectory is named the same as the package, and without the `-ExcludeVersion` option this name will include the specific version installed. Inside the subdirectory is a `tools` directory that contains the Python installation:

```
# Without -ExcludeVersion
> .\python.3.5.2\tools\python.exe -V
Python 3.5.2

# With -ExcludeVersion
> .\python\tools\python.exe -V
Python 3.5.2
```

In general, nuget packages are not upgradeable, and newer versions should be installed side-by-side and referenced using the full path. Alternatively, delete the package directory manually and install it again. Many CI systems will do this automatically if they do not preserve files between builds.

Alongside the `tools` directory is a `build\native` directory. This contains a MSBuild properties file `python.props` that can be used in a C++ project to reference the Python install. Including the settings will automatically use the headers and import libraries in your build.

The package information pages on nuget.org are www.nuget.org/packages/python for the 64-bit version, www.nuget.org/packages/pythonx86 for the 32-bit version, and www.nuget.org/packages/pythonarm64 for the ARM64 version

4.3.1 Free-threaded packages

Added in version 3.13: (Experimental)

Σημείωση

Everything described in this section is considered experimental, and should be expected to change in future releases.

Packages containing free-threaded binaries are named `python-freethreaded` for the 64-bit version, `pythonx86-freethreaded` for the 32-bit version, and `pythonarm64-freethreaded` for the ARM64 version. These packages contain both the `python3.13t.exe` and `python.exe` entry points, both of which run free threaded.

4.4 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

Anaconda

Popular scientific modules (such as numpy, scipy and pandas) and the `conda` package manager.

Enthought Deployment Manager

«The Next Generation Python Environment and Package Manager».

Previously Enthought provided Canopy, but it [reached end of life in 2016](#).

WinPython

Windows-specific distribution with prebuilt scientific packages and tools for building packages.

Note that these packages may not include the latest versions of Python or other libraries, and are not maintained or supported by the core Python team.

4.5 Supported Windows versions

As specified in [PEP 11](#), a Python release only supports a Windows platform while Microsoft considers the platform under extended support. This means that Python 3.14 supports Windows 10 and newer. If you require Windows 7 support, please install Python 3.8. If you require Windows 8.1 support, please install Python 3.12.

4.6 Removing the MAX_PATH Limitation

Windows historically has limited path lengths to 260 characters. This meant that paths longer than this would not resolve and errors would result.

In the latest versions of Windows, this limitation can be expanded to over 32,000 characters. Your administrator will need to activate the «Enable Win32 long paths» group policy, or set `LongPathsEnabled` to 1 in the registry key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`.

This allows the `open()` function, the `os` module and most other path functionality to accept and return paths longer than 260 characters.

After changing the above option and rebooting, no further configuration is required.

4.7 UTF-8 mode

Added in version 3.7.

Windows still uses legacy encodings for the system encoding (the ANSI Code Page). Python uses it for the default encoding of text files (e.g. `locale.getencoding()`).

This may cause issues because UTF-8 is widely used on the internet and most Unix systems, including WSL (Windows Subsystem for Linux).

You can use the Python UTF-8 Mode to change the default text encoding to UTF-8. You can enable the Python UTF-8 Mode via the `-X utf8` command line option, or the `PYTHONUTF8=1` environment variable. See [PYTHONUTF8](#) for enabling UTF-8 mode, and [Python Install Manager](#) for how to modify environment variables.

When the Python UTF-8 Mode is enabled, you can still use the system encoding (the ANSI Code Page) via the «mbscs» codec.

Note that adding `PYTHONUTF8=1` to the default environment variables will affect all Python 3.7+ applications on your system. If you have any Python 3.7+ applications which rely on the legacy system encoding, it is recommended to set the environment variable temporarily or use the `-X utf8` command line option.

i Σημείωση

Even when UTF-8 mode is disabled, Python uses UTF-8 by default on Windows for:

- Console I/O including standard I/O (see [PEP 528](#) for details).
- The *filesystem encoding* (see [PEP 529](#) for details).

4.8 Finding modules

These notes supplement the description at `sys.path-init` with detailed Windows notes.

When no `._pth` file is found, this is how `sys.path` is populated on Windows:

- An empty entry is added at the start, which corresponds to the current directory.
- If the environment variable `PYTHONPATH` exists, as described in *Environment variables*, its entries are added next. Note that on Windows, paths in this variable must be separated by semicolons, to distinguish them from the colon used in drive identifiers (`C:\` etc.).
- Additional «application paths» can be added in the registry as subkeys of `\SOFTWARE\Python\PythonCore{version}\PythonPath` under both the `HKEY_CURRENT_USER` and `HKEY_LOCAL_MACHINE` hives. Subkeys which have semicolon-delimited path strings as their default value will cause each path to be added to `sys.path`. (Note that all known installers only use `HKLM`, so `HKCU` is typically empty.)
- If the environment variable `PYTHONHOME` is set, it is assumed as «Python Home». Otherwise, the path of the main Python executable is used to locate a «landmark file» (either `Lib\os.py` or `pythonXY.zip`) to deduce the «Python Home». If a Python home is found, the relevant sub-directories added to `sys.path` (`Lib`, `plat-win`, etc) are based on that folder. Otherwise, the core Python path is constructed from the `PythonPath` stored in the registry.
- If the Python Home cannot be located, no `PYTHONPATH` is specified in the environment, and no registry entries can be found, a default path with relative entries is used (e.g. `.\Lib;.\plat-win`, etc).

If a `pyvenv.cfg` file is found alongside the main executable or in the directory one level above the executable, the following variations apply:

- If `home` is an absolute path and `PYTHONHOME` is not set, this path is used instead of the path to the main executable when deducing the home location.

The end result of all this is:

- When running `python.exe`, or any other `.exe` in the main Python directory (either an installed version, or directly from the PCbuild directory), the core path is deduced, and the core paths in the registry are ignored. Other «application paths» in the registry are always read.
- When Python is hosted in another `.exe` (different directory, embedded via COM, etc), the «Python Home» will not be deduced, so the core path from the registry is used. Other «application paths» in the registry are always read.
- If Python can't find its home and there are no registry value (frozen `.exe`, some very strange installation setup) you get a path with some default, but relative, paths.

For those who want to bundle Python into their application or distribution, the following advice will prevent conflicts with other installations:

- Include a `._pth` file alongside your executable containing the directories to include. This will ignore paths listed in the registry and environment variables, and also ignore `site` unless `import site` is listed.
- If you are loading `python3.dll` or `python37.dll` in your own executable, explicitly set `PyConfig.module_search_paths` before `Py_InitializeFromConfig()`.
- Clear and/or overwrite `PYTHONPATH` and set `PYTHONHOME` before launching `python.exe` from your application.

- If you cannot use the previous suggestions (for example, you are a distribution that allows people to run `python.exe` directly), ensure that the landmark file (`Lib\os.py`) exists in your install directory. (Note that it will not be detected inside a ZIP file, but a correctly named ZIP file will be detected instead.)

These will ensure that the files in a system-wide installation will not take precedence over the copy of the standard library bundled with your application. Otherwise, your users may experience problems using your application. Note that the first suggestion is the best, as the others may still be susceptible to non-standard paths in the registry and user site-packages.

Άλλαξε στην έκδοση 3.6: Add `._pth` file support and removes `applocal` option from `pyvenv.cfg`.

Άλλαξε στην έκδοση 3.6: Add `pythonXX.zip` as a potential landmark when directly adjacent to the executable.

Αποσύρθηκε στην έκδοση 3.6: Modules specified in the registry under `Modules` (not `PythonPath`) may be imported by `importlib.machinery.WindowsRegistryFinder`. This finder is enabled on Windows in 3.6.0 and earlier, but may need to be explicitly added to `sys.meta_path` in the future.

4.9 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 `mswin-specific-services`.

4.9.1 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.

Δείτε επίσης

Win32 How Do I...?

by Tim Golden

Python and COM

by David and Paul Boddie

4.9.2 cx_Freeze

`cx_Freeze` 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.

4.10 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](#).

The source tree contains a build solution and project files for Microsoft Visual Studio, which is the compiler used to build the official Python releases. These files are in the `PCbuild` directory.

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

For extension modules, consult `building-on-windows`.

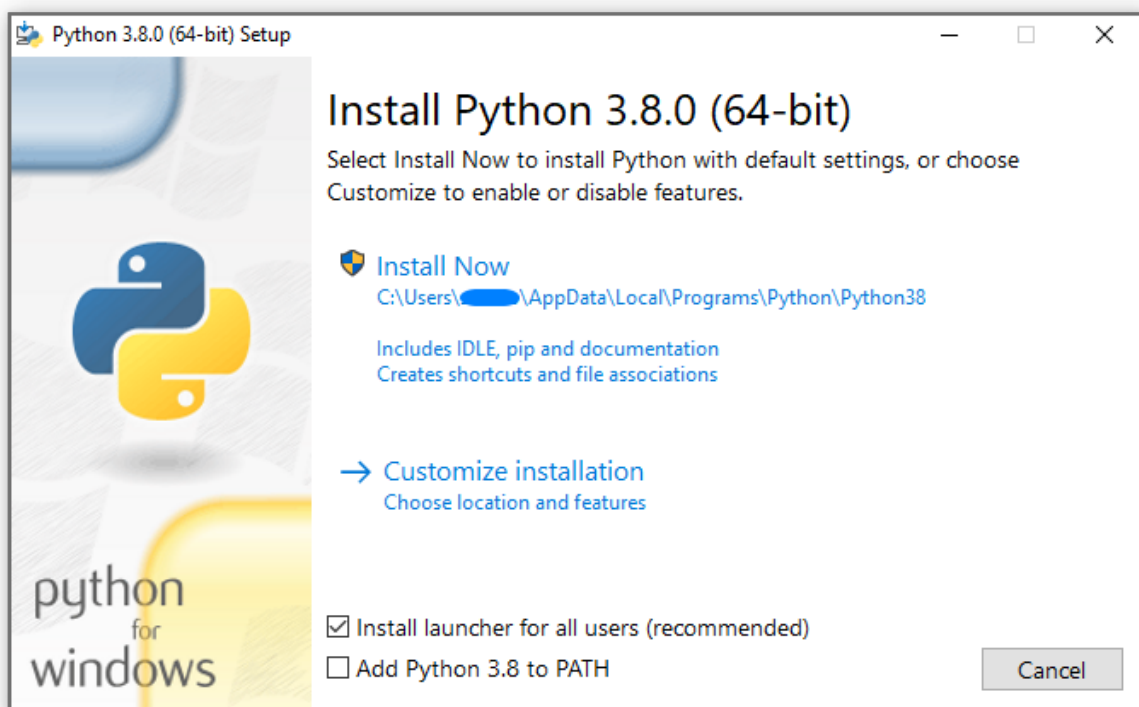
4.11 The full installer (deprecated)

Αποσύρθηκε στην έκδοση 3.14: This installer is deprecated since 3.14 and will not be produced for Python 3.16 or later. See *Python Install Manager* for the modern installer.

4.11.1 Installation steps

Four Python 3.14 installers are available for download - two each for the 32-bit and 64-bit versions of the interpreter. The *web installer* is a small initial download, and it will automatically download the required components as necessary. The *offline installer* includes the components necessary for a default installation and only requires an internet connection for optional features. See *Installing Without Downloading* for other ways to avoid downloading during installation.

After starting the installer, one of two options may be selected:



If you select «Install Now»:

- You will *not* need to be an administrator (unless a system update for the C Runtime Library is required or you install the *Python Install Manager* for all users)
- Python will be installed into your user directory
- The *Python Install Manager* will be installed according to the option at the bottom of the first page
- The standard library, test suite, launcher and pip will be installed
- If selected, the install directory will be added to your `PATH`
- Shortcuts will only be visible for the current user

Selecting «Customize installation» will allow you to select the features to install, the installation location and other options or post-install actions. To install debugging symbols or binaries, you will need to use this option.

To perform an all-users installation, you should select «Customize installation». In this case:

- You may be required to provide administrative credentials or approval
- Python will be installed into the Program Files directory
- The *Python Install Manager* will be installed into the Windows directory
- Optional features may be selected during installation
- The standard library can be pre-compiled to bytecode
- If selected, the install directory will be added to the system `PATH`
- Shortcuts are available for all users

4.11.2 Removing the MAX_PATH Limitation

Windows historically has limited path lengths to 260 characters. This meant that paths longer than this would not resolve and errors would result.

In the latest versions of Windows, this limitation can be expanded to approximately 32,000 characters. Your administrator will need to activate the «Enable Win32 long paths» group policy, or set `LongPathsEnabled` to 1 in the registry key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`.

This allows the `open()` function, the `os` module and most other path functionality to accept and return paths longer than 260 characters.

After changing the above option, no further configuration is required.

Άλλαξε στην έκδοση 3.6: Support for long paths was enabled in Python.

4.11.3 Installing Without UI

All of the options available in the installer UI can also be specified from the command line, allowing scripted installers to replicate an installation on many machines without user interaction. These options may also be set without suppressing the UI in order to change some of the defaults.

The following options (found by executing the installer with `/?`) can be passed into the installer:

Name	Description
<code>/passive</code>	to display progress without requiring user interaction
<code>/quiet</code>	to install/uninstall without displaying any UI
<code>/simple</code>	to prevent user customization
<code>/uninstall</code>	to remove Python (without confirmation)
<code>/layout [directory]</code>	to pre-download all components
<code>/log [filename]</code>	to specify log files location

All other options are passed as `name=value`, where the value is usually 0 to disable a feature, 1 to enable a feature, or a path. The full list of available options is shown below.

Name	Description	Default
InstallAllU	Perform a system-wide installation.	0
TargetDir	The installation directory	Selected based on InstallAllUsers
DefaultAll	The default installation directory for all-user installs	%ProgramFiles%\Python X.Y or %ProgramFiles(x86)\Python X.Y
DefaultJus	The default install directory for just-for-me installs	%LocalAppData%\Programs\Python\PythonXY or %LocalAppData%\Programs\Python\PythonXY-32 or %LocalAppData%\Programs\Python\PythonXY-64
DefaultCus	The default custom install directory displayed in the UI	(empty)
AssociateF	Create file associations if the launcher is also installed.	1
CompileA	Compile all .py files to .pyc.	0
PrependPa	Prepend install and Scripts directories to PATH and add .PY to PATHEXT	0
AppendPa	Append install and Scripts directories to PATH and add .PY to PATHEXT	0
Shortcuts	Create shortcuts for the interpreter, documentation and IDLE if installed.	1
Include_dc	Install Python manual	1
Include_de	Install debug binaries	0
Include_de	Install developer headers and libraries. Omitting this may lead to an unusable installation.	1
Include_ex	Install python.exe and related files. Omitting this may lead to an unusable installation.	1
Include_la	Install <i>Python Install Manager</i> .	1
InstallLaur	Installs the launcher for all users. Also requires Include_launcher to be set to 1	1
Include_lit	Install standard library and extension modules. Omitting this may lead to an unusable installation.	1
Include_pi	Install bundled pip and setuptools	1
Include_sy	Install debugging symbols (*.pdb)	0
Include_tc	Install Tcl/Tk support and IDLE	1
Include_te	Install standard library test suite	1
Include_to	Install utility scripts	1
LauncherC	Only installs the launcher. This will override most other options.	0
SimpleInst	Disable most install UI	0
SimpleInst	A custom message to display when the simplified install UI is used.	(empty)

For example, to silently install a default, system-wide Python installation, you could use the following command (from

an elevated command prompt):

```
python-3.9.0.exe /quiet InstallAllUsers=1 PrependPath=1 Include_test=0
```

To allow users to easily install a personal copy of Python without the test suite, you could provide a shortcut with the following command. This will display a simplified initial page and disallow customization:

```
python-3.9.0.exe InstallAllUsers=0 Include_launcher=0 Include_test=0
SimpleInstall=1 SimpleInstallDescription="Just for me, no test suite."
```

(Note that omitting the launcher also omits file associations, and is only recommended for per-user installs when there is also a system-wide installation that included the launcher.)

The options listed above can also be provided in a file named `unattend.xml` alongside the executable. This file specifies a list of options and values. When a value is provided as an attribute, it will be converted to a number if possible. Values provided as element text are always left as strings. This example file sets the same options as the previous example:

```
<Options>
  <Option Name="InstallAllUsers" Value="no" />
  <Option Name="Include_launcher" Value="0" />
  <Option Name="Include_test" Value="no" />
  <Option Name="SimpleInstall" Value="yes" />
  <Option Name="SimpleInstallDescription">Just for me, no test suite</
  <Option>
</Options>
```

4.11.4 Installing Without Downloading

As some features of Python are not included in the initial installer download, selecting those features may require an internet connection. To avoid this need, all possible components may be downloaded on-demand to create a complete *layout* that will no longer require an internet connection regardless of the selected features. Note that this download may be bigger than required, but where a large number of installations are going to be performed it is very useful to have a locally cached copy.

Execute the following command from Command Prompt to download all possible required files. Remember to substitute `python-3.9.0.exe` for the actual name of your installer, and to create layouts in their own directories to avoid collisions between files with the same name.

```
python-3.9.0.exe /layout [optional target directory]
```

You may also specify the `/quiet` option to hide the progress display.

4.11.5 Modifying an install

Once Python has been installed, you can add or remove features through the Programs and Features tool that is part of Windows. Select the Python entry and choose «Uninstall/Change» to open the installer in maintenance mode.

«Modify» allows you to add or remove features by modifying the checkboxes - unchanged checkboxes will not install or remove anything. Some options cannot be changed in this mode, such as the install directory; to modify these, you will need to remove and then reinstall Python completely.

«Repair» will verify all the files that should be installed using the current settings and replace any that have been removed or modified.

«Uninstall» will remove Python entirely, with the exception of the *Python Install Manager*, which has its own entry in Programs and Features.

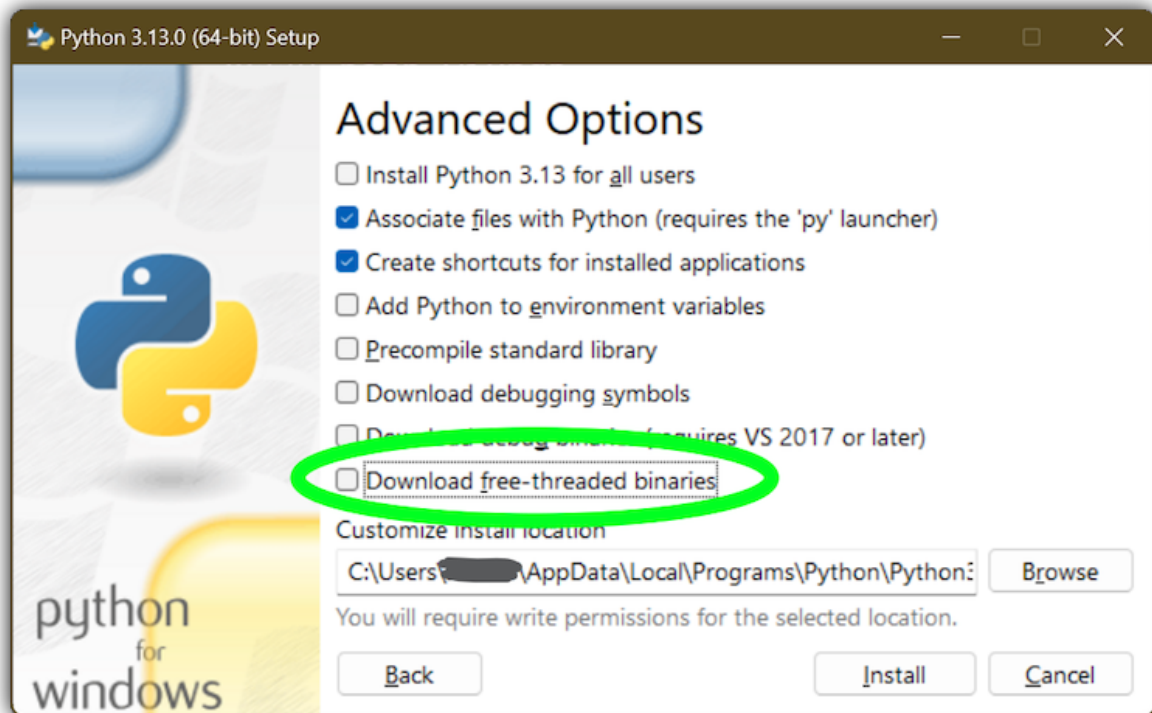
4.11.6 Installing Free-threaded Binaries

Added in version 3.13: (Experimental)

Σημείωση

Everything described in this section is considered experimental, and should be expected to change in future releases.

To install pre-built binaries with free-threading enabled (see [PEP 703](#)), you should select «Customize installation». The second page of options includes the «Download free-threaded binaries» checkbox.



Selecting this option will download and install additional binaries to the same location as the main Python install. The main executable is called `python3.13t.exe`, and other binaries either receive a `t` suffix or a full ABI suffix. Python source files and bundled third-party dependencies are shared with the main install.

The free-threaded version is registered as a regular Python install with the tag `3.13t` (with a `-32` or `-arm64` suffix as normal for those platforms). This allows tools to discover it, and for the *Python Install Manager* to support `py.exe -3.13t`. Note that the launcher will interpret `py.exe -3` (or a `python3` shebang) as «the latest 3.x install», which will prefer the free-threaded binaries over the regular ones, while `py.exe -3.13` will not. If you use the short style of option, you may prefer to not install the free-threaded binaries at this time.

To specify the install option at the command line, use `Include_freethreaded=1`. See *Installing Without Downloading* for instructions on pre-emptively downloading the additional binaries for offline install. The options to include debug symbols and binaries also apply to the free-threaded builds.

Free-threaded binaries are also available [on nuget.org](https://nuget.org).

4.12 Python Launcher for Windows (Deprecated)

Αποσύρθηκε στην έκδοση 3.14: The launcher and this documentation have been superseded by the Python Install Manager described above. This is preserved temporarily for historical interest.

Added in version 3.3.

The Python launcher for Windows is a utility which aids in locating and executing of different Python versions. It allows scripts (or the command-line) to indicate a preference for a specific Python version, and will locate and execute that version.

Unlike the `PATH` variable, the launcher will correctly select the most appropriate version of Python. It will prefer per-user installations over system-wide ones, and orders by language version rather than using the most recently installed version.

The launcher was originally specified in [PEP 397](#).

4.12.1 Getting started

From the command-line

Αλλάξε στην έκδοση 3.6.

System-wide installations of Python 3.3 and later will put the launcher on your `PATH`. The launcher is compatible with all available versions of Python, so it does not matter which version is installed. To check that the launcher is available, execute the following command in Command Prompt:

```
py
```

You should find that the latest version of Python you have installed is started - it can be exited as normal, and any additional command-line arguments specified will be sent directly to Python.

If you have multiple versions of Python installed (e.g., 3.7 and 3.14) you will have noticed that Python 3.14 was started - to launch Python 3.7, try the command:

```
py -3.7
```

If you want the latest version of Python 2 you have installed, try the command:

```
py -2
```

If you see the following error, you do not have the launcher installed:

```
'py' is not recognized as an internal or external command,  
operable program or batch file.
```

The command:

```
py --list
```

displays the currently installed version(s) of Python.

The `-x.y` argument is the short form of the `-V:Company/Tag` argument, which allows selecting a specific Python runtime, including those that may have come from somewhere other than python.org. Any runtime registered by following [PEP 514](#) will be discoverable. The `--list` command lists all available runtimes using the `-V:` format.

When using the `-V:` argument, specifying the Company will limit selection to runtimes from that provider, while specifying only the Tag will select from all providers. Note that omitting the slash implies a tag:

```
# Select any '3.*' tagged runtime
py -V:3

# Select any 'PythonCore' released runtime
py -V:PythonCore/

# Select PythonCore's latest Python 3 runtime
py -V:PythonCore/3
```

The short form of the argument (`-3`) only ever selects from core Python releases, and not other distributions. However, the longer form (`-V:3`) will select from any.

The Company is matched on the full string, case-insensitive. The Tag is matched on either the full string, or a prefix, provided the next character is a dot or a hyphen. This allows `-V:3.1` to match `3.1-32`, but not `3.10`. Tags are sorted using numerical ordering (`3.10` is newer than `3.1`), but are compared using text (`-V:3.01` does not match `3.1`).

Virtual environments

Added in version 3.5.

If the launcher is run with no explicit Python version specification, and a virtual environment (created with the standard library `venv` module or the external `virtualenv` tool) active, the launcher will run the virtual environment's interpreter rather than the global one. To run the global interpreter, either deactivate the virtual environment, or explicitly specify the global Python version.

From a script

Let's create a test Python script - create a file called `hello.py` with the following contents

```
#!/python
import sys
sys.stdout.write("hello from Python %s\n" % (sys.version,))
```

From the directory in which `hello.py` lives, execute the command:

```
py hello.py
```

You should notice the version number of your latest Python 2.x installation is printed. Now try changing the first line to be:

```
#!/python3
```

Re-executing the command should now print the latest Python 3.x information. As with the above command-line examples, you can specify a more explicit version qualifier. Assuming you have Python 3.7 installed, try changing the first line to `#!/python3.7` and you should find the 3.7 version information printed.

Note that unlike interactive use, a bare `<python>` will use the latest version of Python 2.x that you have installed. This is for backward compatibility and for compatibility with Unix, where the command `python` typically refers to Python 2.

From file associations

The launcher should have been associated with Python files (i.e. `.py`, `.pyw`, `.pyc` files) when it was installed. This means that when you double-click on one of these files from Windows explorer the launcher will be used, and therefore you can use the same facilities described above to have the script specify the version which should be used.

The key benefit of this is that a single launcher can support multiple Python versions at the same time depending on the contents of the first line.

4.12.2 Shebang Lines

If the first line of a script file starts with `#!`, it is known as a «shebang» line. Linux and other Unix like operating systems have native support for such lines and they are commonly used on such systems to indicate how a script should be executed. This launcher allows the same facilities to be used with Python scripts on Windows and the examples above demonstrate their use.

To allow shebang lines in Python scripts to be portable between Unix and Windows, this launcher supports a number of “virtual” commands to specify which interpreter to use. The supported virtual commands are:

- `/usr/bin/env`
- `/usr/bin/python`
- `/usr/local/bin/python`

- python

For example, if the first line of your script starts with

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

The default Python or an active virtual environment will be located and used. As many Python scripts written to work on Unix will already have this line, you should find these scripts can be used by the launcher without modification. If you are writing a new script on Windows which you hope will be useful on Unix, you should use one of the shebang lines starting with `/usr`.

Any of the above virtual commands can be suffixed with an explicit version (either just the major version, or the major and minor version). Furthermore the 32-bit version can be requested by adding «-32» after the minor version. I.e. `/usr/bin/python3.7-32` will request usage of the 32-bit Python 3.7. If a virtual environment is active, the version will be ignored and the environment will be used.

Added in version 3.7: Beginning with python launcher 3.7 it is possible to request 64-bit version by the «-64» suffix. Furthermore it is possible to specify a major and architecture without minor (i.e. `/usr/bin/python3-64`).

Αλλάξε στην έκδοση 3.11: The «-64» suffix is deprecated, and now implies «any architecture that is not provably i386/32-bit». To request a specific environment, use the new `-V:TAG` argument with the complete tag.

Αλλάξε στην έκδοση 3.13: Virtual commands referencing `python` now prefer an active virtual environment rather than searching `PATH`. This handles cases where the shebang specifies `/usr/bin/env python3` but `python3.exe` is not present in the active environment.

The `/usr/bin/env` form of shebang line has one further special property. Before looking for installed Python interpreters, this form will search the executable `PATH` for a Python executable matching the name provided as the first argument. This corresponds to the behaviour of the Unix `env` program, which performs a `PATH` search. If an executable matching the first argument after the `env` command cannot be found, but the argument starts with `python`, it will be handled as described for the other virtual commands. The environment variable `PYLAUNCHER_NO_SEARCH_PATH` may be set (to any value) to skip this search of `PATH`.

Shebang lines that do not match any of these patterns are looked up in the `[commands]` section of the launcher's `.INI` file. This may be used to handle certain commands in a way that makes sense for your system. The name of the command must be a single argument (no spaces in the shebang executable), and the value substituted is the full path to the executable (additional arguments specified in the `.INI` will be quoted as part of the filename).

[commands]

```
/bin/xpython=C:\Program Files\XPython\python.exe
```

Any commands not found in the `.INI` file are treated as **Windows** executable paths that are absolute or relative to the directory containing the script file. This is a convenience for Windows-only scripts, such as those generated by an installer, since the behavior is not compatible with Unix-style shells. These paths may be quoted, and may include multiple arguments, after which the path to the script and any additional arguments will be appended.

4.12.3 Arguments in shebang lines

The shebang lines can also specify additional options to be passed to the Python interpreter. For example, if you have a shebang line:

```
#!/usr/bin/python -v
```

Then Python will be started with the `-v` option

4.12.4 Customization

Customization via INI files

Two `.ini` files will be searched by the launcher - `py.ini` in the current user's application data directory (`%LOCALAPPDATA%` or `$env:LocalAppData`) and `py.ini` in the same directory as the launcher. The same `.ini` files are used for both the “console” version of the launcher (i.e. `py.exe`) and for the “windows” version (i.e. `pyw.exe`).

Customization specified in the «application directory» will have precedence over the one next to the executable, so a user, who may not have write access to the .ini file next to the launcher, can override commands in that global .ini file.

Customizing default Python versions

In some cases, a version qualifier can be included in a command to dictate which version of Python will be used by the command. A version qualifier starts with a major version number and can optionally be followed by a period (".") and a minor version specifier. Furthermore it is possible to specify if a 32 or 64 bit implementation shall be requested by adding «-32» or «-64».

For example, a shebang line of `#!/python` has no version qualifier, while `#!/python3` has a version qualifier which specifies only a major version.

If no version qualifiers are found in a command, the environment variable `PY_PYTHON` can be set to specify the default version qualifier. If it is not set, the default is «3». The variable can specify any value that may be passed on the command line, such as «3», «3.7», «3.7-32» or «3.7-64». (Note that the «-64» option is only available with the launcher included with Python 3.7 or newer.)

If no minor version qualifiers are found, the environment variable `PY_PYTHON{major}` (where {major} is the current major version qualifier as determined above) can be set to specify the full version. If no such option is found, the launcher will enumerate the installed Python versions and use the latest minor release found for the major version, which is likely, although not guaranteed, to be the most recently installed version in that family.

On 64-bit Windows with both 32-bit and 64-bit implementations of the same (major.minor) Python version installed, the 64-bit version will always be preferred. This will be true for both 32-bit and 64-bit implementations of the launcher - a 32-bit launcher will prefer to execute a 64-bit Python installation of the specified version if available. This is so the behavior of the launcher can be predicted knowing only what versions are installed on the PC and without regard to the order in which they were installed (i.e., without knowing whether a 32 or 64-bit version of Python and corresponding launcher was installed last). As noted above, an optional «-32» or «-64» suffix can be used on a version specifier to change this behaviour.

Examples:

- If no relevant options are set, the commands `python` and `python2` will use the latest Python 2.x version installed and the command `python3` will use the latest Python 3.x installed.
- The command `python3.7` will not consult any options at all as the versions are fully specified.
- If `PY_PYTHON=3`, the commands `python` and `python3` will both use the latest installed Python 3 version.
- If `PY_PYTHON=3.7-32`, the command `python` will use the 32-bit implementation of 3.7 whereas the command `python3` will use the latest installed Python (`PY_PYTHON` was not considered at all as a major version was specified.)
- If `PY_PYTHON=3` and `PY_PYTHON3=3.7`, the commands `python` and `python3` will both use specifically 3.7

In addition to environment variables, the same settings can be configured in the .INI file used by the launcher. The section in the INI file is called `[defaults]` and the key name will be the same as the environment variables without the leading `PY_` prefix (and note that the key names in the INI file are case insensitive.) The contents of an environment variable will override things specified in the INI file.

For example:

- Setting `PY_PYTHON=3.7` is equivalent to the INI file containing:

```
[defaults]
python=3.7
```

- Setting `PY_PYTHON=3` and `PY_PYTHON3=3.7` is equivalent to the INI file containing:

```
[defaults]
python=3
python3=3.7
```


4.12.5 Diagnostics

If an environment variable `PYLAUNCHER_DEBUG` is set (to any value), the launcher will print diagnostic information to `stderr` (i.e. to the console). While this information manages to be simultaneously verbose *and* terse, it should allow you to see what versions of Python were located, why a particular version was chosen and the exact command-line used to execute the target Python. It is primarily intended for testing and debugging.

4.12.6 Dry Run

If an environment variable `PYLAUNCHER_DRYRUN` is set (to any value), the launcher will output the command it would have run, but will not actually launch Python. This may be useful for tools that want to use the launcher to detect and then launch Python directly. Note that the command written to standard output is always encoded using UTF-8, and may not render correctly in the console.

4.12.7 Install on demand

If an environment variable `PYLAUNCHER_ALLOW_INSTALL` is set (to any value), and the requested Python version is not installed but is available on the Microsoft Store, the launcher will attempt to install it. This may require user interaction to complete, and you may need to run the command again.

An additional `PYLAUNCHER_ALWAYS_INSTALL` variable causes the launcher to always try to install Python, even if it is detected. This is mainly intended for testing (and should be used with `PYLAUNCHER_DRYRUN`).

4.12.8 Return codes

The following exit codes may be returned by the Python launcher. Unfortunately, there is no way to distinguish these from the exit code of Python itself.

The names of codes are as used in the sources, and are only for reference. There is no way to access or resolve them apart from reading this page. Entries are listed in alphabetical order of names.

Name	Value	Description
<code>RC_BAD_VENV_CFG</code>	107	A <code>pyvenv.cfg</code> was found but is corrupt.
<code>RC_CREATE_PROCESS</code>	101	Failed to launch Python.
<code>RC_INSTALLING</code>	111	An install was started, but the command will need to be re-run after it completes.
<code>RC_INTERNAL_ERROR</code>	109	Unexpected error. Please report a bug.
<code>RC_NO_COMMANDLINE</code>	108	Unable to obtain command line from the operating system.
<code>RC_NO_PYTHON</code>	103	Unable to locate the requested version.
<code>RC_NO_VENV_CFG</code>	106	A <code>pyvenv.cfg</code> was required but not found.

Χρησιμοποιώντας Python σε macOS

Αυτό το έγγραφο αποσκοπεί στο να δώσει μια επισκόπηση της συμπεριφοράς που σχετίζεται με το macOS που θα πρέπει να γνωρίζετε για να ξεκινήσετε με την Python σε υπολογιστές Mac. Η Python σε Mac που εκτελεί macOS είναι πολύ παρόμοια με την Python σε άλλες πλατφόρμες που προέρχονται από το Unix, αλλά υπάρχουν κάποιες διαφορές στην εγκατάσταση και κάποιες δυνατότητες.

Υπάρχουν διάφοροι τρόποι για να αποκτήσετε και να εγκαταστήσετε την Python για macOS. Προκατασκευασμένες εκδόσεις των πιο πρόσφατων εκδόσεων της Python είναι διαθέσιμες από έναν αριθμό διανομέων. Πολλές από αυτές τις πληροφορίες περιγράφουν τη χρήση της Python που παρέχονται από την ομάδα κυκλοφορίας CPython για λήψη από την ιστοσελίδα python.org. Δείτε *Εναλλακτικές Διανομές* για μερικές άλλες επιλογές.

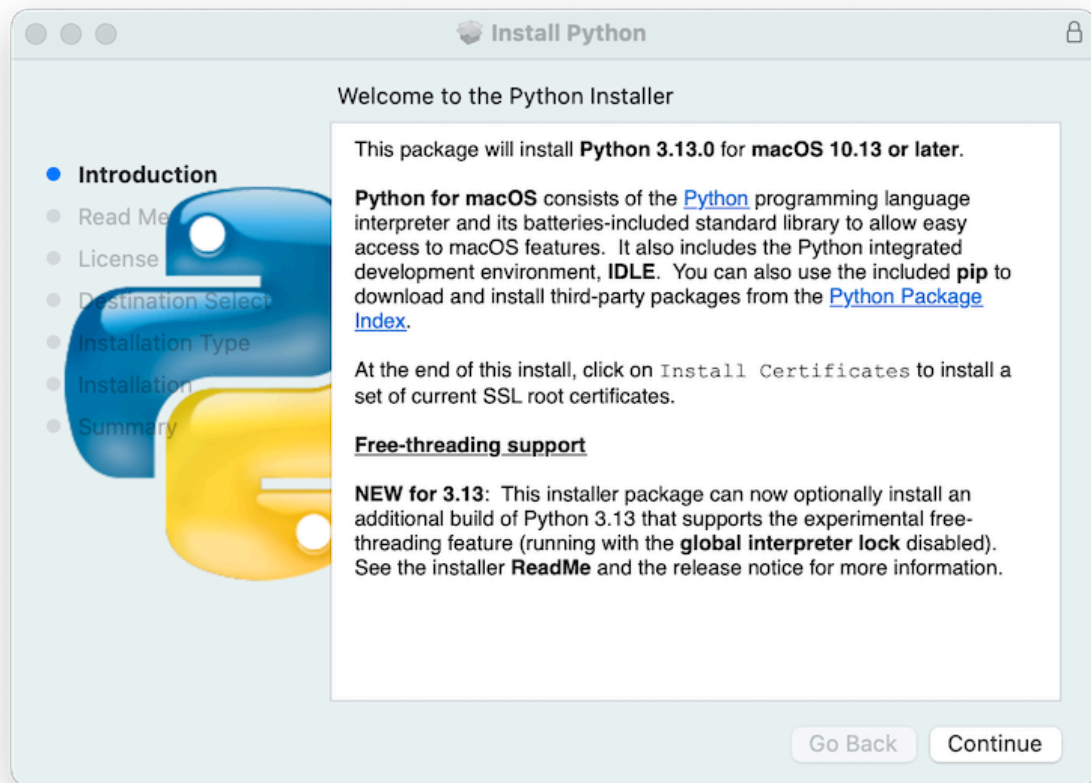
5.1 Χρησιμοποιώντας Python για macOS από το `python.org`

5.1.1 Βήματα εγκατάστασης

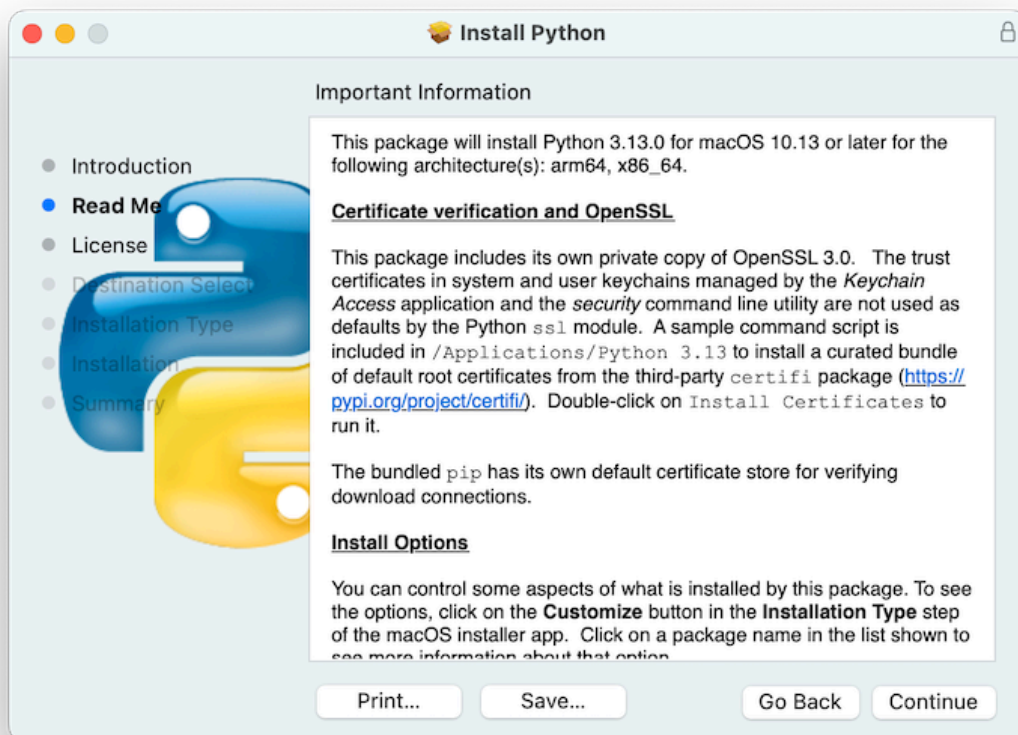
For [current Python versions](#) (other than those in `security status`), the release team produces a **Python for macOS** installer package for each new release. A list of available installers is available [here](#). We recommend using the most recent supported Python version where possible. Current installers provide a `universal2 binary` build of Python which runs natively on all Macs (Apple Silicon and Intel) that are supported by a wide range of macOS versions, currently typically from at least **macOS 10.15 Catalina** on.

Το αρχείο που κατεβάσατε είναι ένα τυπικό αρχείο πακέτου εγκατάστασης macOS (`.pkg`). Οι πληροφορίες ακεραιότητας αρχείου (έλεγχος ταυτότητας, μέγεθος, υπογραφή sigstore κ.λπ.) για κάθε αρχείο περιλαμβάνονται στη σελίδα λήψης κυκλοφορίας. Τα πακέτα εγκατάστασης και το περιεχόμενό τους είναι υπογεγραμμένα και επικυρωμένα με Python Software Foundation Apple Developer ID πιστοποιητικά για να ικανοποιούν τις απαιτήσεις `macOS Gatekeeper`.

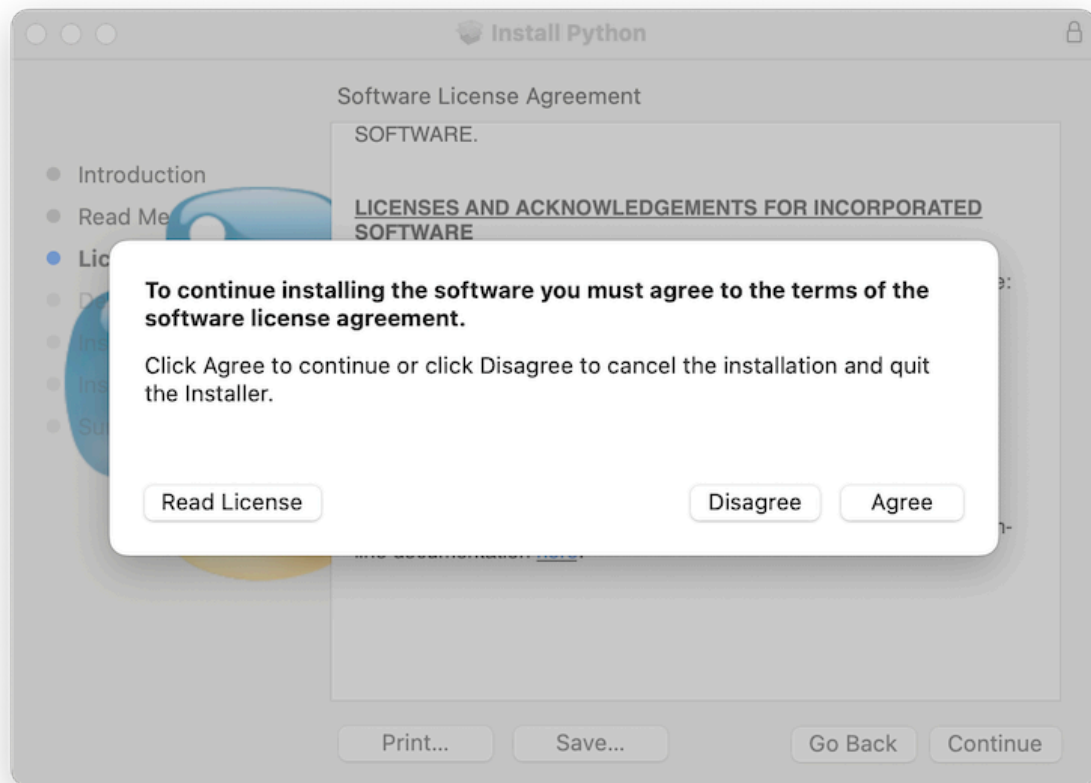
Για μια τυπική εγκατάσταση, κάντε διπλό κλικ στο αρχείο πακέτου εγκατάστασης που κατεβάσατε. Αυτό θα πρέπει να εκκινήσει την τυπική εφαρμογή εγκατάστασης macOS και να εμφανίσει το πρώτο από αρκετά βήματα παραθύρου εγκατάστασης.



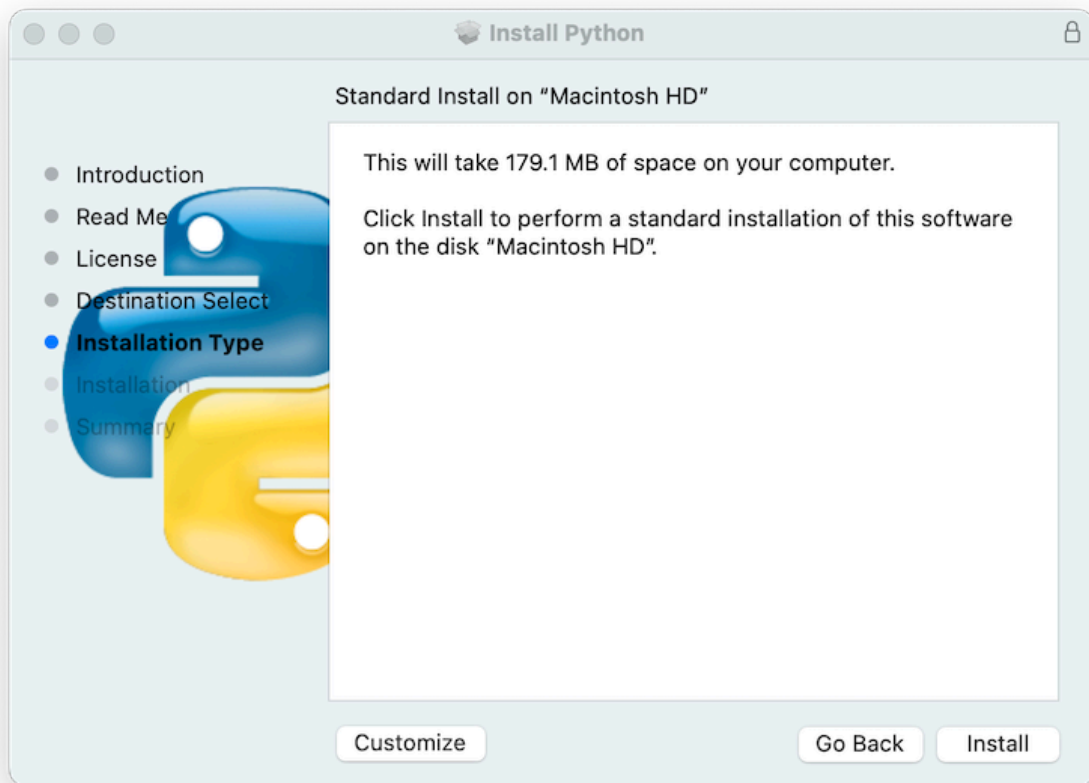
Κάνοντας κλικ στο κουμπί **Continue** εμφανίζεται το **Read Me** για αυτόν τον εγκαταστάτη. Εκτός από άλλες σημαντικές πληροφορίες, το **Read Me** καταγράφει ποια έκδοση Python πρόκειται να εγκατασταθεί και σε ποιες εκδόσεις macOS υποστηρίζεται. Ίσως χρειαστεί να κάνετε κύλιση για να διαβάσετε ολόκληρο το αρχείο. Από προεπιλογή, αυτό το **Read Me** θα εγκατασταθεί επίσης στο `/Applications/Python 3.14/` και θα είναι διαθέσιμο για ανάγνωση οποιαδήποτε στιγμή.



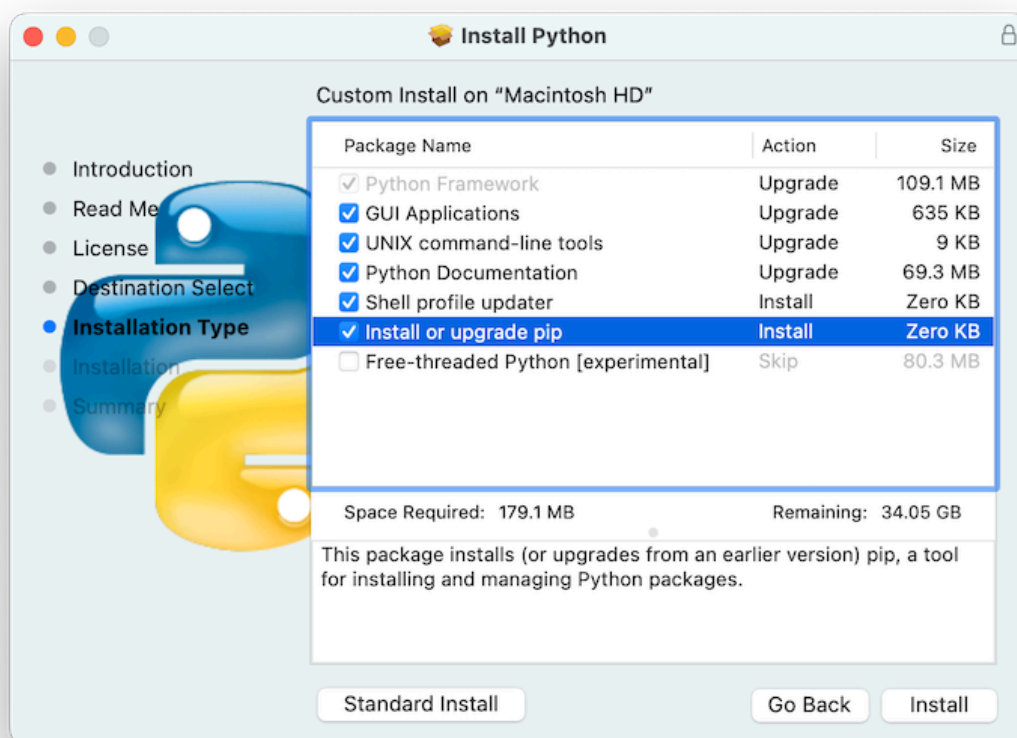
Κάνοντας κλικ στο **Continue** προχωρά στην εμφάνιση της άδειας για την Python και για άλλο περιλαμβανόμενο λογισμικό. Στη συνέχεια θα χρειαστεί να **Συμφωνήσετε** με τους όρους άδειας πριν προχωρήσετε στο επόμενο βήμα. Αυτό το αρχείο άδειας θα εγκατασταθεί επίσης και θα είναι διαθέσιμο για ανάγνωση αργότερα.



Αφού γίνουν αποδεκτοί οι όροι άδειας, το επόμενο βήμα είναι η εμφάνιση **Τύπος Εγκατάστασης**. Για τις περισσότερες χρήσεις, το τυπικό σύνολο λειτουργιών εγκατάστασης είναι κατάλληλο.



Πατώντας το κουμπί **Customize**, μπορείτε να επιλέξετε να παραλείψετε ή να επιλέξετε ορισμένα πακέτα του εγκαταστάτη. Κάντε κλικ σε κάθε όνομα πακέτου για να δείτε μια περιγραφή του τι εγκαθιστά. Για να εγκαταστήσετε επίσης υποστήριξη για την προαιρετική δυνατότητα χωρίς νήματα, δείτε [Εγκατάσταση εκδόσεων Free-threaded](#).

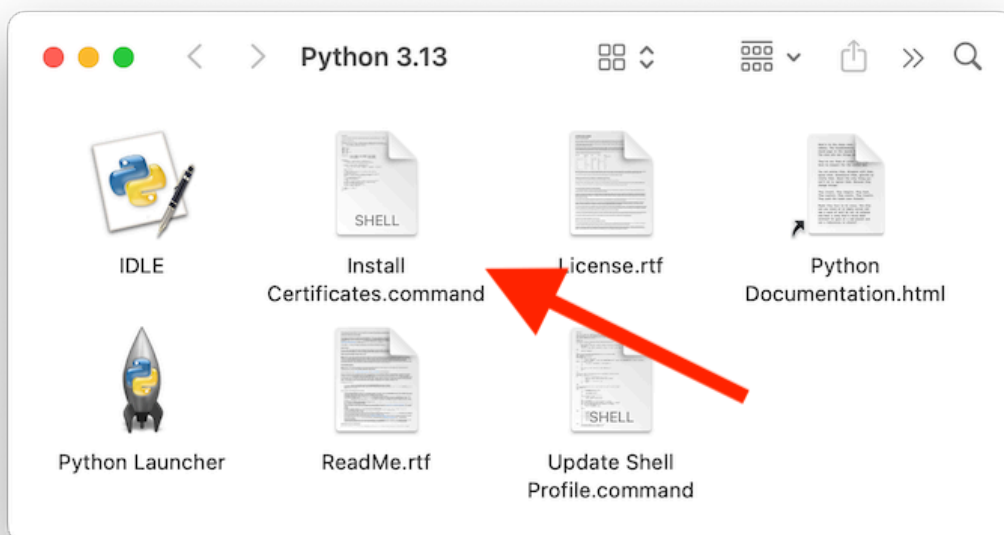


Σε κάθε περίπτωση, κάνοντας κλικ στο **Install** θα ξεκινήσει η διαδικασία εγκατάστασης ζητώντας άδεια για την εγκατάσταση νέου λογισμικού. Ένα όνομα χρήστη macOS με Administrator δικαιώματα είναι απαραίτητο καθώς η εγκατεστημένη Python θα είναι διαθέσιμη σε όλους τους χρήστες του Mac.

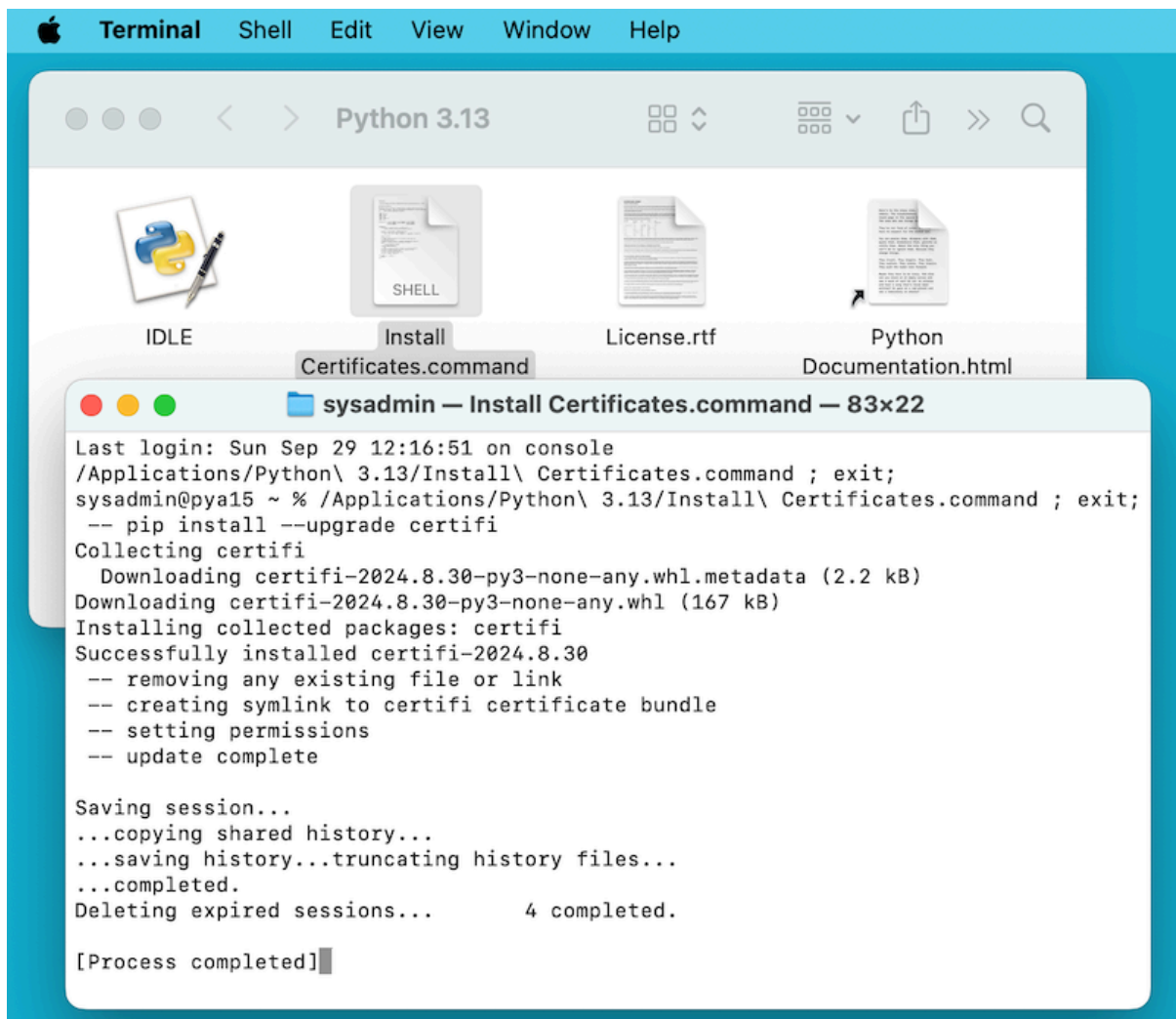
Όταν ολοκληρωθεί η εγκατάσταση, θα εμφανιστεί το παράθυρο **Περίληψη**.



Διπλό κλικ στο εικονίδιο ή το αρχείο **Install Certificates.command** στο παράθυρο / Applications/Python 3.14/ για να ολοκληρώσετε την εγκατάσταση.



Αυτό θα ανοίξει ένα προσωρινό παράθυρο κελύφους **Terminal** που θα χρησιμοποιήσει τη νέα Python για να κατεβάσει και να εγκαταστήσει ριζικούς πιστοποιητικούς SSL για τη χρήση του.



Εάν εμφανιστεί `Successfully installed certifi` και `update complete` στο παράθυρο τερματικού, η εγκατάσταση είναι ολοκληρωμένη. Κλείστε αυτό το παράθυρο τερματικού και το παράθυρο εγκατάστασης.

Μια προεπιλεγμένη εγκατάσταση θα περιλαμβάνει:

- Ένας Python 3.14 φάκελος στον φάκελο Applications. Εδώ θα βρείτε το **IDLE**, το περιβάλλον ανάπτυξης που είναι τυπικό μέρος των επίσημων διανομών Python• και το **Python Launcher**, το οποίο χειρίζεται τα διπλά κλικ σε Python scripts από το macOS Finder.
- Ένα πλαίσιο `/Library/Frameworks/Python.framework`, το οποίο περιλαμβάνει τον εκτελέσιμο κώδικα Python και τις βιβλιοθήκες. Το πρόγραμμα εγκατάστασης προσθέτει αυτήν την τοποθεσία στο shell path σας. Για να απεγκαταστήσετε την Python, μπορείτε να αφαιρέσετε αυτά τα τρία πράγματα. Symlinks στον εκτελέσιμο κώδικα Python τοποθετούνται στο `/usr/local/bin/`.

Σημείωση

Οι πρόσφατες εκδόσεις του macOS περιλαμβάνουν μια εντολή **python3** στο `/usr/bin/python3` που συνδέεται με μια συνήθως παλαιότερη και ελλιπή έκδοση της Python που παρέχεται από και για χρήση από τα εργαλεία ανάπτυξης της Apple, **Xcode** ή τα **Command Line Tools for Xcode**. Δεν θα πρέπει ποτέ να τροποποιήσετε ή να προσπαθήσετε να διαγράψετε αυτήν την εγκατάσταση, καθώς ελέγχεται από την Apple και χρησιμοποιείται από το λογισμικό που παρέχεται από την Apple ή από τρίτους. Εάν επιλέξετε να εγκαταστήσετε μια νεότερη έκδοση Python από το `python.org`, θα έχετε δύο διαφορετικές αλλά λειτουργικές εγκαταστάσεις Python στον υπολογιστή σας που μπορούν να συνυπάρχουν. Οι προεπιλεγμένες επιλογές εγκατάστασης θα πρέπει να διασφαλίσουν ότι η **python3** θα χρησιμοποιηθεί

αντί της συστήματος **python3**.

5.1.2 Πώς να εκτελέσετε ένα σενάριο Python

Υπάρχουν δύο τρόποι για να καλέσετε τον διερμηνέα Python. Εάν είστε εξοικειωμένοι με τη χρήση ενός κελύφους Unix σε ένα παράθυρο τερματικού, μπορείτε να καλέσετε `python3.14` ή `python3` προαιρετικά ακολουθούμενο από μία ή περισσότερες επιλογές γραμμής εντολών (περιγράφεται στο *Command line and environment*). Το εγχειρίδιο Python έχει επίσης μια χρήσιμη ενότητα σχετικά με `using Python interactively from a shell`.

Μπορείτε επίσης να καλέσετε τον ερμηνευτή μέσω ενός ενσωματωμένου αναπτυξιακού περιβάλλοντος. `idle` είναι ένας βασικός επεξεργαστής και περιβάλλον ερμηνευτή που περιλαμβάνεται με την τυπική διανομή Python. Το **IDLE** περιλαμβάνει ένα μενού Βοήθειας που σας επιτρέπει να αποκτήσετε πρόσβαση στην τεκμηρίωση Python. Εάν είστε εντελώς νέοι στην Python, μπορείτε να διαβάσετε την εισαγωγή του εγχειριδίου σε αυτό το έγγραφο.

Υπάρχουν πολλοί άλλοι επεξεργαστές και IDE διαθέσιμοι, δείτε *Επεξεργαστές Κειμένου και IDEs* για περισσότερες πληροφορίες.

Για να εκτελέσετε ένα αρχείο script Python από το παράθυρο τερματικού, μπορείτε να καλέσετε τον ερμηνευτή με το όνομα του αρχείου script:

```
python3.14 myscript.py
```

Για να εκτελέσετε το script σας από το Finder, μπορείτε είτε:

- Σύρετε το στο **Python Launcher**.
- Επιλέξτε το **Python Launcher** ως την προεπιλεγμένη εφαρμογή για να ανοίξετε το script σας (ή οποιοδήποτε `.py` script) μέσω του παραθύρου πληροφοριών Finder και κάντε διπλό κλικ σε αυτό. Το **Python Launcher** έχει διάφορες προτιμήσεις για να ελέγξει πώς εκκινείται το script σας. Η επιλογή-μεταφορά σας επιτρέπει να αλλάξετε αυτά για μία κλήση ή χρησιμοποιήστε το μενού Προτιμήσεις για να αλλάξετε τα πράγματα παγκοσμίως.

Να είστε προσεκτικοί ότι η εκτέλεση του script απευθείας από το macOS Finder μπορεί να παράγει διαφορετικά αποτελέσματα από την εκτέλεση από ένα παράθυρο τερματικού καθώς το script δεν θα εκτελείται στο συνήθη περιβάλλον κελύφους που περιλαμβάνει οποιαδήποτε ρύθμιση μεταβλητών περιβάλλοντος σε προφίλ κελύφους. Και, όπως με οποιοδήποτε άλλο script ή πρόγραμμα, να είστε σίγουροι για το τι πρόκειται να εκτελέσετε.

5.2 Εναλλακτικές Διανομές

Εκτός από τον τυπικό εγκαταστάτη `python.org` για macOS, υπάρχουν τρίτες διανομές για macOS που μπορεί να περιλαμβάνουν πρόσθετη λειτουργικότητα. Ορισμένες δημοφιλείς διανομές και τα βασικά χαρακτηριστικά τους:

ActivePython

Εγκαταστάτης με συμβατότητα πολλών πλατφορμών, τεκμηρίωση

Anaconda

Δημοφιλή επιστημονικά modules (όπως `numpy`, `scipy` και `pandas`) και ο `conda` διαχειριστής πακέτων.

Homebrew

Διαχειριστής πακέτων για macOS που περιλαμβάνει πολλές εκδόσεις Python και πολλά τρίτα πακέτα Python (συμπεριλαμβανομένων των `numpy`, `scipy` και `pandas`).

MacPorts

Ένας άλλος διαχειριστής πακέτων για macOS που περιλαμβάνει πολλές εκδόσεις Python και πολλά τρίτα πακέτα Python. Μπορεί να περιλαμβάνει προ-κατασκευασμένες εκδόσεις της Python και πολλά πακέτα για παλαιότερες εκδόσεις του macOS.

Σημειώστε ότι οι διανομές ενδέχεται να μην περιλαμβάνουν τις τελευταίες εκδόσεις της Python ή άλλων βιβλιοθηκών και δεν υποστηρίζονται ή συντηρούνται από την κύρια ομάδα Python.

5.3 Εγκατάσταση πρόσθετων πακέτων Python

Ανατρέξτε στον [Python Packaging User Guide](#) για περισσότερες πληροφορίες.

5.4 Προγραμματισμός GUI

Υπάρχουν πολλές επιλογές για την κατασκευή εφαρμογών GUI στον Mac με Python.

Η τυπική βιβλιοθήκη GUI Python είναι `tkinter`, βασισμένη στο διαλεειτουργικό εργαλείο Tk (<https://www.tcl.tk>). Μια εγγενής έκδοση Tk macOS περιλαμβάνεται με τον εγκαταστάτη.

Το `PyObjC` είναι ένα Python binding στο πλαίσιο Objective-C/Cocoa της Apple. Πληροφορίες σχετικά με το `PyObjC` είναι διαθέσιμες από [pyobjc](#).

Ένας αριθμός εναλλακτικών εργαλείων GUI macOS είναι διαθέσιμος, συμπεριλαμβανομένων:

- **PySide**: Επίσημα Python bindings στο Qt GUI toolkit.
- **PyQt**: Εναλλακτικά Python bindings στο Qt.
- **Kivy**: Ένα διαλεειτουργικό εργαλείο GUI που υποστηρίζει desktop και mobile πλατφόρμες.
- **Toga**: Μέρος του [BeeWare Project](#); υποστηρίζει εφαρμογές desktop, mobile, web και κονσόλας εφαρμογές.
- **wxPython**: Ένα διαλεειτουργικό εργαλείο που υποστηρίζει desktop πλατφόρμες.

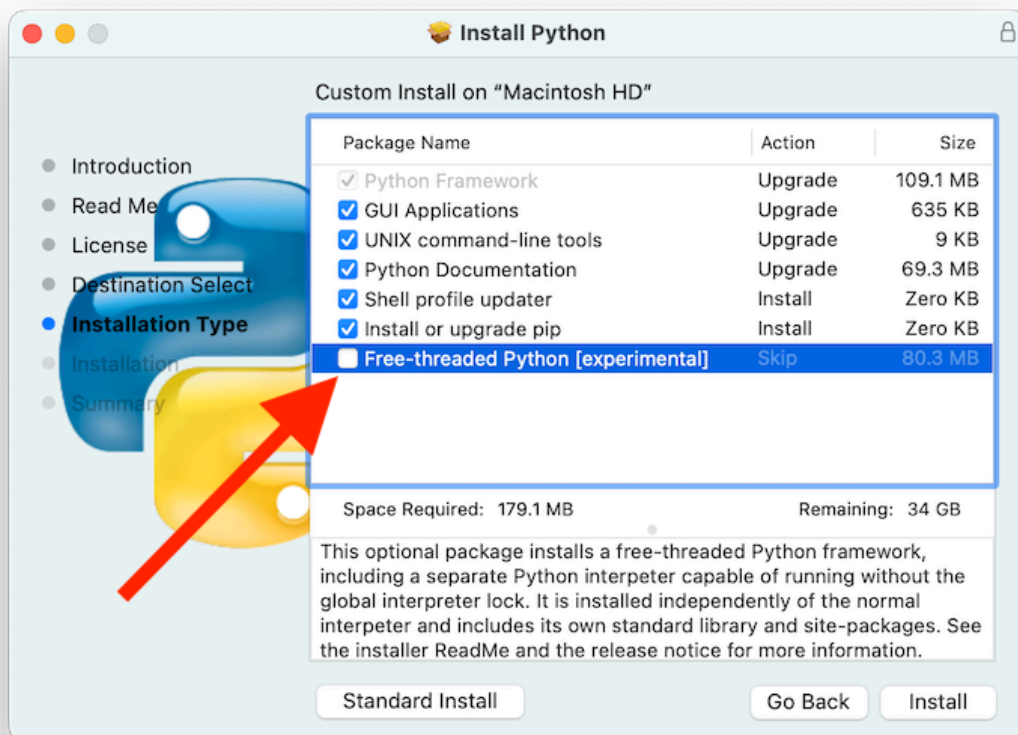
5.5 Προχωρημένα θέματα

5.5.1 Εγκατάσταση εκδόσεων Free-threaded

Added in version 3.13.

Το πακέτο εγκατάστασης `python.org Python for macOS` μπορεί προαιρετικά να εγκαταστήσει μια επιπλέον έκδοση της Python 3.14 που υποστηρίζει **PEP 703**, την πειραματική δυνατότητα ελεύθερης νηματοποίησης (threading) (εκτέλεση με το *global interpreter lock* απενεργοποιημένο). Ελέγξτε τη σελίδα κυκλοφορίας στο `python.org` για πιθανές ενημερωμένες πληροφορίες.

Η λειτουργία ελεύθερου νήματος λειτουργεί και συνεχίζει να βελτιώνεται, αλλά υπάρχει κάποια πρόσθετη επιβάρυνση στα φορτία εργασίας μονού νήματος σε σύγκριση με την κανονική έκδοση. Επιπλέον, τα πακέτα τρίτων κατασκευαστών, ιδίως αυτά με *extension module*, ενδέχεται να μην είναι έτοιμα για χρήση σε μια έκδοση ελεύθερου νήματος και θα ενεργοποιήσουν ξανά το *GIL*. Επομένως, η υποστήριξη για ελεύθερη νηματοποίησης δεν εγκαθίστανται από προεπιλογή. Είναι συσκευασμένη ως ξεχωριστή επιλογή εγκατάστασης, διαθέσιμη κάνοντας κλικ το κουμπί **Customize** στο βήμα **Installation Type** του προγράμματος εγκατάστασης όπως περιγράφεται παραπάνω.



Εάν το πλαίσιο δίπλα στο όνομα πακέτου **Free-threaded Python** είναι επιλεγμένο, θα εγκατασταθεί επίσης ένα ξεχωριστό `PythonT.framework` δίπλα στο κανονικό `Python.framework` στο `/Library/Frameworks`. Αυτή η διαμόρφωση επιτρέπει σε μια έκδοση Python 3.14 ελεύθερης νηματοποίησης να συγκατοικεί στο σύστημά σας με μια παραδοσιακή (μόνο GIL) Python 3.14 έκδοση με ελάχιστο κίνδυνο κατά την εγκατάσταση ή τη δοκιμή. Αυτή η διάταξη εγκατάστασης είναι πειραματική και υπόκειται σε αλλαγές σε μελλοντικές εκδόσεις.

Γνωστές προφυλάξεις και περιορισμοί:

- Το **UNIX command-line tools** πακέτο, το οποίο επιλέγεται από προεπιλογή, θα εγκαταστήσει συνδέσμους στο `/usr/local/bin` για `python3.14t`, τον ερμηνευτή ελεύθερης νηματοποίησης και `python3.14t-config`, ένα βοηθητικό πρόγραμμα διαμόρφωσης που μπορεί να είναι χρήσιμο για κατασκευαστές πακέτων. Δεδομένου ότι το `/usr/local/bin` περιλαμβάνεται συνήθως στη μεταβλητή περιβάλλοντος `PATH` του κελύφους σας, στις περισσότερες περιπτώσεις δεν θα χρειαστούν αλλαγές στις μεταβλητές περιβάλλοντος `PATH` για να χρησιμοποιήσετε `python3.14t`.
- Για αυτήν την έκδοση, το πακέτο **Shell profile updater** και το `Update Shell Profile.command` στο `/Applications/Python 3.14/` δεν υποστηρίζουν το πακέτο ελεύθερης νηματοποίησης.
- Η έκδοση ελεύθερης νηματοποίησης και η παραδοσιακή έκδοση έχουν ξεχωριστές διαδρομές αναζήτησης και ξεχωριστούς καταλόγους `site-packages`, οπότε, από προεπιλογή, εάν χρειάζεστε ένα πακέτο διαθέσιμο και στις δύο εκδόσεις, μπορεί να χρειαστεί να εγκατασταθεί και στις δύο. Το πακέτο ελεύθερης νηματοποίησης θα εγκαταστήσει μια ξεχωριστή έκδοση του **pip** για χρήση με `python3.14t`.
 - Για να εγκαταστήσετε ένα πακέτο χρησιμοποιώντας **pip** χωρίς **venv**:


```
python3.14t -m pip install <package_name>
```
- Όταν εργάζεστε με πολλές Python περιβάλλοντα, είναι συνήθως ασφαλέστερο και ευκολότερο να `create and use virtual environments`. Αυτό μπορεί να αποφύγει πιθανά ονόματα εντολών συγκρούσεων και σύγχυση σχετικά με το ποια Python είναι σε χρήση:


```
python3.14t -m venv <venv_name>
```

τότε **activate**.

- Για να εκτελέσετε μια έκδοση ελεύθερης νηματοποίησης του IDLE:

```
python3.14t -m idlelib
```

- Οι ερμηνευτές και στις δύο εκδόσεις ανταγωνίζονται στις ίδιες *PYTHON environment variables* που μπορεί να έχουν απροσδόκητα αποτελέσματα, για παράδειγμα, εάν έχετε ρυθμίσει το PYTHONPATH σε ένα προφίλ κελύφους. Εάν είναι απαραίτητο, υπάρχουν *command line options* όπως `-E` για να αγνοήσετε αυτές τις μεταβλητές περιβάλλοντος.
- Η έκδοση ελεύθερης νηματοποίησης συνδέεται με τις βιβλιοθήκες κοινής χρήσης τρίτων, όπως OpenSSL και Tk, που είναι εγκατεστημένες στο παραδοσιακό πλαίσιο. Αυτό σημαίνει ότι και οι δύο εκδόσεις μοιράζονται επίσης ένα σύνολο πιστοποιητικών εμπιστοσύνης όπως εγκαθίστανται από το **Install Certificates.command** script, επομένως χρειάζεται να εκτελείται μόνο μία φορά.
- Εάν δεν μπορείτε να εξαρτηθείτε από το σύνδεσμο στο `/usr/local/bin` που δείχνει στην έκδοση `python3.14t` ελεύθερης νηματοποίησης (για παράδειγμα, εάν θέλετε να εγκαταστήσετε τη δική σας έκδοση εκεί ή κάποια άλλη διανομή το κάνει), μπορείτε να ορίσετε ρητά τη μεταβλητή περιβάλλοντος PATH του κελύφους σας για να περιλαμβάνει το κατάλογο bin του πλαισίου PythonT:

```
export PATH="/Library/Frameworks/PythonT.framework/Versions/3.14/bin":  
↪$PATH"
```

Η παραδοσιακή εγκατάσταση πλαισίου από προεπιλογή κάνει κάτι παρόμοιο, εκτός από το `Python.framework`. Να είστε προσεκτικοί ότι η παρουσία και των δύο καταλόγων bin πλαισίου στο PATH μπορεί να οδηγήσει σε σύγχυση εάν υπάρχουν διπλά ονόματα όπως `python3.14` και στις δύο• ποιο χρησιμοποιείται στην πραγματικότητα εξαρτάται από τη σειρά που εμφανίζονται στο PATH. Οι εντολές `which python3.x` ή `which python3.xt` μπορούν να δείξουν ποια διαδρομή χρησιμοποιείται. Η χρήση εικονικών περιβαλλόντων μπορεί να βοηθήσει στην αποφυγή τέτοιων αμφισβημιών. Μια άλλη επιλογή μπορεί να είναι η δημιουργία ενός κελύφους **alias** στον επιθυμητό διεργαστή, όπως:

```
alias py3.14="/Library/Frameworks/Python.framework/Versions/3.14/bin/  
↪python3.14"  
alias py3.14t="/Library/Frameworks/PythonT.framework/Versions/3.14/bin/  
↪python3.14t"
```

5.5.2 Εγκατάσταση χρησιμοποιώντας τη γραμμή εντολών

Εάν θέλετε να χρησιμοποιήσετε αυτοματισμό για να εγκαταστήσετε το πακέτο εγκατάστασης `python.org` (παρά να χρησιμοποιήσετε την οικεία εφαρμογή **Installer** GUI macOS), το βοηθητικό πρόγραμμα **installer** γραμμής εντολών macOS σας επιτρέπει να επιλέξετε μη προεπιλεγμένες επιλογές, επίσης. Εάν δεν είστε εξοικειωμένοι με το **installer**, μπορεί να είναι κάπως κρυπτικό (βλ. **man installer** για περισσότερες πληροφορίες). Ως παράδειγμα, το παρακάτω απόσπασμα κώδικα δείχνει έναν τρόπο να το κάνετε, χρησιμοποιώντας την έκδοση 3.14.0b2 και επιλέγοντας την επιλογή διεργαστή ελεύθερης νηματοποίησης:

```
RELEASE="python-3.140b2-macos11.pkg"
```

```
# download installer pkg
```

```
curl -O https://www.python.org/ftp/python/3.14.0/${RELEASE}
```

```
# create installer choicechanges to customize the install:
```

```
# enable the PythonTFramework-3.14 package
```

```
# while accepting the other defaults (install all other packages)
```

```
cat > ./choicechanges.plist <<EOF
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/  
↪DTDs/PropertyList-1.0.dtd">
```

```
<plist version="1.0">
```

```
<array>
```

```

<dict>
  <key>attributeSetting</key>
  <integer>1</integer>
  <key>choiceAttribute</key>
  <string>selected</string>
  <key>choiceIdentifier</key>
  <string>org.python.Python.PythonTFramework-3.14</string>
</dict>
</array>
</plist>
EOF

```

```

sudo installer -pkg ./${RELEASE} -applyChoiceChangesXML ./choicechanges.
→plist -target /

```

Μπορείτε στη συνέχεια να ελέγξετε ότι και οι δύο εγκαταστάτες είναι διαθέσιμοι με κάτι σαν:

```

$ # test that the free-threaded interpreter was installed if the Unix_
→Command Tools package was enabled
$ /usr/local/bin/python3.14t -VV
Python 3.14.0b2 free-threading build (v3.14.0b2:3a83b172af, Jun  5 2024, _
→12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ # and the traditional interpreter
$ /usr/local/bin/python3.14 -VV
Python 3.14.0b2 (v3.14.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.
→0 (clang-1500.3.9.4)]
$ # test that they are also available without the prefix if /usr/local/bin_
→is on $PATH
$ python3.14t -VV
Python 3.14.0b2 free-threading build (v3.14.0b2:3a83b172af, Jun  5 2024, _
→12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ python3.14 -VV
Python 3.14.0b2 (v3.14.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.
→0 (clang-1500.3.9.4)]

```

Σημείωση

Τα τρέχοντα προγράμματα εγκατάστασης python.org εγκαθίστανται μόνο σε καθορισμένες τοποθεσίες όπως /Library/Frameworks/, /Applications και /usr/local/bin. Δεν μπορείτε να χρησιμοποιήσετε την επιλογή `-domain` του **installer** για να εγκαταστήσετε σε άλλες τοποθεσίες.

5.5.3 Διανομή εφαρμογών Python

Υπάρχει μια σειρά εργαλείων για τη μετατροπή του κώδικα Python σας σε μια αυτόνομη διανεμόμενη εφαρμογή:

- **py2app**: Υποστηρίζει τη δημιουργία macOS `.app` πακέτων από ένα έργο Python.
- **Briefcase**: Μέρος του [BeeWare Project](#); ένα εργαλείο διανομής πολλαπλών πλατφορμών που υποστηρίζει τη δημιουργία `.app` πακέτων σε macOS, καθώς και τη διαχείριση υπογραφής και πιστοποίησης.
- **PyInstaller**: Ένα εργαλείο διανομής πολλαπλών πλατφορμών που δημιουργεί ένα μόνο αρχείο ή φάκελο ως διανεμόμενο αντικείμενο.

5.5.4 Συμμόρφωση App Store

Οι εφαρμογές που υποβάλλονται για διανομή μέσω του macOS App Store πρέπει να περάσουν τη διαδικασία αναθεώρησης εφαρμογών της Apple. Αυτή η διαδικασία περιλαμβάνει ένα σύνολο αυτοματοποιημένων κανόνων επικύρωσης που ελέγχουν το υποβληθέν πακέτο εφαρμογής για προβληματικό κώδικα.

Η βιβλιοθήκη προτύπων Python περιέχει κάποιον κώδικα που είναι γνωστό ότι παραβιάζει αυτούς τους αυτοματοποιημένους κανόνες. Ενώ αυτές οι παραβιάσεις φαίνεται να είναι ψευδώς θετικές, οι κανόνες αναθεώρησης της Apple δεν μπορούν να αμφισβητηθούν. Επομένως, είναι απαραίτητο να τροποποιηθεί η βιβλιοθήκη προτύπων Python για να περάσει μια εφαρμογή την αναθεώρηση του App Store.

Το δέντρο πηγών Python περιέχει *ένα αρχείο patch* που θα αφαιρέσει όλο τον κώδικα που είναι γνωστό ότι προκαλεί προβλήματα με τη διαδικασία αναθεώρησης του App Store. Αυτή η επιδιόρθωση εφαρμόζεται αυτόματα όταν η CPython διαμορφώνεται με την `--with-app-store-compliance` επιλογή.

Αυτή η επιδιόρθωση δεν απαιτείται κανονικά για τη χρήση της CPython σε Mac. Ούτε απαιτείται εάν διανέμετε μια εφαρμογή *εκτός* του macOS App Store. Απαιτείται *μόνο* εάν χρησιμοποιείτε το macOS App Store ως κανάλι διανομής.

5.6 Άλλοι πόροι

Η [python.org Help page](#) έχει συνδέσμους σε πολλούς χρήσιμους πόρους. Η [Pythonmac-SIG mailing list](#) είναι ένας άλλος πόρος υποστήριξης ειδικά για χρήστες και προγραμματιστές Python στο Mac.

Using Python on Android

Python on Android is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a **python** executable and entering commands at an interactive prompt, or by running a Python script.

On Android, there is no concept of installing as a system resource. The only unit of software distribution is an «app». There is also no console where you could run a **python** executable, or interact with a Python REPL.

As a result, the only way you can use Python on Android is in embedded mode – that is, by writing a native Android application, embedding a Python interpreter using `libpython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged into your app for its own private use.

The Python standard library has some notable omissions and restrictions on Android. See the API availability guide for details.

6.1 Adding Python to an Android app

Most app developers should use one of the following tools, which will provide a much easier experience:

- [Briefcase](#), from the BeeWare project
- [Buildozer](#), from the Kivy project
- [Chaquopy](#)
- [pyqtdeploy](#)
- [Termux](#)

If you're sure you want to do all of this manually, read on. You can use the [testbed app](#) as a guide; each step below contains a link to the relevant file.

- Build Python by following the instructions in [Android/README.md](#). This will create the directory `cross-build/HOST/prefix`.
- Add code to your [build.gradle](#) file to copy the following items into your project. All except your own Python code can be copied from `prefix/lib`:
 - In your JNI libraries:
 - * `libpython*.so`
 - * `lib*_python.so` (external libraries such as OpenSSL)

- In your assets:
 - * `python*.*` (the Python standard library)
 - * `python*/site-packages` (your own Python code)
- Add code to your app to [extract the assets to the filesystem](#).
- Add code to your app to [start Python in embedded mode](#). This will need to be C code called via JNI.

6.2 Building a Python package for Android

Python packages can be built for Android as wheels and released on PyPI. The recommended tool for doing this is [cibuildwheel](#), which automates all the details of setting up a cross-compilation environment, building the wheel, and testing it on an emulator.

Using Python on iOS

Authors

Russell Keith-Magee (2024-03)

Python on iOS is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a **python** executable and entering commands at an interactive prompt, or by running a Python script.

On iOS, there is no concept of installing as a system resource. The only unit of software distribution is an «app». There is also no console where you could run a **python** executable, or interact with a Python REPL.

As a result, the only way you can use Python on iOS is in embedded mode - that is, by writing a native iOS application, and embedding a Python interpreter using `libPython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged as a standalone bundle that can be distributed via the iOS App Store.

If you're looking to experiment for the first time with writing an iOS app in Python, projects such as [BeeWare](#) and [Kivy](#) will provide a much more approachable user experience. These projects manage the complexities associated with getting an iOS project running, so you only need to deal with the Python code itself.

7.1 Python at runtime on iOS

7.1.1 iOS version compatibility

The minimum supported iOS version is specified at compile time, using the `--host` option to `configure`. By default, when compiled for iOS, Python will be compiled with a minimum supported iOS version of 13.0. To use a different minimum iOS version, provide the version number as part of the `--host` argument - for example, `--host=arm64-apple-ios15.4-simulator` would compile an ARM64 simulator build with a deployment target of 15.4.

7.1.2 Platform identification

When executing on iOS, `sys.platform` will report as `ios`. This value will be returned on an iPhone or iPad, regardless of whether the app is running on the simulator or a physical device.

Information about the specific runtime environment, including the iOS version, device model, and whether the device is a simulator, can be obtained using `platform.ios_ver()`. `platform.system()` will report `ios` or `iPadOS`, depending on the device.

`os.uname()` reports kernel-level details; it will report a name of Darwin.

7.1.3 Standard library availability

The Python standard library has some notable omissions and restrictions on iOS. See the API availability guide for iOS for details.

7.1.4 Binary extension modules

One notable difference about iOS as a platform is that App Store distribution imposes hard requirements on the packaging of an application. One of these requirements governs how binary extension modules are distributed.

The iOS App Store requires that *all* binary modules in an iOS app must be dynamic libraries, contained in a framework with appropriate metadata, stored in the `Frameworks` folder of the packaged app. There can be only a single binary per framework, and there can be no executable binary material outside the `Frameworks` folder.

This conflicts with the usual Python approach for distributing binaries, which allows a binary extension module to be loaded from any location on `sys.path`. To ensure compliance with App Store policies, an iOS project must post-process any Python packages, converting `.so` binary modules into individual standalone frameworks with appropriate metadata and signing. For details on how to perform this post-processing, see the guide for [adding Python to your project](#).

To help Python discover binaries in their new location, the original `.so` file on `sys.path` is replaced with a `.framework` file. This file is a text file containing the location of the framework binary, relative to the app bundle. To allow the framework to resolve back to the original location, the framework must contain a `.origin` file that contains the location of the `.framework` file, relative to the app bundle.

For example, consider the case of an import from `foo.bar import _whiz`, where `_whiz` is implemented with the binary module `sources/foo/bar/_whiz.abi3.so`, with `sources` being the location registered on `sys.path`, relative to the application bundle. This module *must* be distributed as `Frameworks/foo.bar._whiz.framework/foo.bar._whiz` (creating the framework name from the full import path of the module), with an `Info.plist` file in the `.framework` directory identifying the binary as a framework. The `foo.bar._whiz` module would be represented in the original location with a `sources/foo/bar/_whiz.abi3.framework` marker file, containing the path `Frameworks/foo.bar._whiz.framework/foo.bar._whiz`. The framework would also contain `Frameworks/foo.bar._whiz.framework/foo.bar._whiz.origin`, containing the path to the `.framework` file.

When running on iOS, the Python interpreter will install an `AppleFrameworkLoader` that is able to read and import `.framework` files. Once imported, the `__file__` attribute of the binary module will report as the location of the `.framework` file. However, the `ModuleSpec` for the loaded module will report the `origin` as the location of the binary in the framework folder.

7.1.5 Compiler stub binaries

Xcode doesn't expose explicit compilers for iOS; instead, it uses an `xcrun` script that resolves to a full compiler path (e.g., `xcrun --sdk iphoneos clang` to get the `clang` for an iPhone device). However, using this script poses two problems:

- The output of `xcrun` includes paths that are machine specific, resulting in a `sysconfig` module that cannot be shared between users; and
- It results in `CC/CPP/LD/AR` definitions that include spaces. There is a lot of C ecosystem tooling that assumes that you can split a command line at the first space to get the path to the compiler executable; this isn't the case when using `xcrun`.

To avoid these problems, Python provided stubs for these tools. These stubs are shell script wrappers around the underlying `xcrun` tools, distributed in a `bin` folder distributed alongside the compiled iOS framework. These scripts are relocatable, and will always resolve to the appropriate local system paths. By including these scripts in the `bin` folder that accompanies a framework, the contents of the `sysconfig` module becomes useful for end-users to compile their own modules. When compiling third-party Python modules for iOS, you should ensure these stub binaries are on your path.

7.2 Installing Python on iOS

7.2.1 Tools for building iOS apps

Building for iOS requires the use of Apple's Xcode tooling. It is strongly recommended that you use the most recent stable release of Xcode. This will require the use of the most (or second-most) recently released macOS version, as Apple does not maintain Xcode for older macOS versions. The Xcode Command Line Tools are not sufficient for iOS development; you need a *full* Xcode install.

If you want to run your code on the iOS simulator, you'll also need to install an iOS Simulator Platform. You should be prompted to select an iOS Simulator Platform when you first run Xcode. Alternatively, you can add an iOS Simulator Platform by selecting from the Platforms tab of the Xcode Settings panel.

7.2.2 Adding Python to an iOS project

Python can be added to any iOS project, using either Swift or Objective C. The following examples will use Objective C; if you are using Swift, you may find a library like [PythonKit](#) to be helpful.

To add Python to an iOS Xcode project:

1. Build or obtain a Python XCFramework. See the instructions in [iOS/README.rst](#) (in the CPython source distribution) for details on how to build a Python XCFramework. At a minimum, you will need a build that supports `arm64-apple-ios`, plus one of either `arm64-apple-ios-simulator` or `x86_64-apple-ios-simulator`.
2. Drag the XCframework into your iOS project. In the following instructions, we'll assume you've dropped the XCframework into the root of your project; however, you can use any other location that you want by adjusting paths as needed.
3. Drag the `iOS/Resources/dylib-Info-template.plist` file into your project, and ensure it is associated with the app target.
4. Add your application code as a folder in your Xcode project. In the following instructions, we'll assume that your user code is in a folder named `app` in the root of your project; you can use any other location by adjusting paths as needed. Ensure that this folder is associated with your app target.
5. Select the app target by selecting the root node of your Xcode project, then the target name in the sidebar that appears.
6. In the «General» settings, under «Frameworks, Libraries and Embedded Content», add `Python.xcframework`, with «Embed & Sign» selected.
7. In the «Build Settings» tab, modify the following:
 - Build Options
 - User Script Sandboxing: No
 - Enable Testability: Yes
 - Search Paths
 - Framework Search Paths: `$(PROJECT_DIR)`
 - Header Search Paths: `"$(BUILT_PRODUCTS_DIR)/Python.framework/Headers"`
 - Apple Clang - Warnings - All languages
 - Quoted Include In Framework Header: No
8. Add a build step that copies the Python standard library into your app. In the «Build Phases» tab, add a new «Run Script» build step *before* the «Embed Frameworks» step, but *after* the «Copy Bundle Resources» step. Name the step «Install Target Specific Python Standard Library», disable the «Based on dependency analysis» checkbox, and set the script content to:

```

set -e

mkdir -p "$CODESIGNING_FOLDER_PATH/python/lib"
if [ "$EFFECTIVE_PLATFORM_NAME" = "-iphonesimulator" ]; then
    echo "Installing Python modules for iOS Simulator"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64_x86_
↳ 64-simulator/lib/" "$CODESIGNING_FOLDER_PATH/python/lib/"
else
    echo "Installing Python modules for iOS Device"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64/lib/
↳ " "$CODESIGNING_FOLDER_PATH/python/lib/"
fi

```

Note that the name of the simulator «slice» in the XCframework may be different, depending the CPU architectures your XCFramework supports.

9. Add a second build step that processes the binary extension modules in the standard library into «Framework» format. Add a «Run Script» build step *directly after* the one you added in step 8, named «Prepare Python Binary Modules». It should also have «Based on dependency analysis» unchecked, with the following script content:

```

set -e

install_dylib () {
    INSTALL_BASE=$1
    FULL_EXT=$2

    # The name of the extension file
    EXT=$(basename "$FULL_EXT")
    # The location of the extension file, relative to the bundle
    RELATIVE_EXT=${FULL_EXT#$CODESIGNING_FOLDER_PATH/}
    # The path to the extension file, relative to the install base
    PYTHON_EXT=${RELATIVE_EXT/$INSTALL_BASE/}
    # The full dotted name of the extension module, constructed from
↳ the file path.
    FULL_MODULE_NAME=$(echo $PYTHON_EXT | cut -d "." -f 1 | tr "/" ".
↳ ");
    # A bundle identifier; not actually used, but required by Xcode
↳ framework packaging
    FRAMEWORK_BUNDLE_ID=$(echo $PRODUCT_BUNDLE_IDENTIFIER.$FULL_MODULE_
↳ NAME | tr "_" "-")
    # The name of the framework folder.
    FRAMEWORK_FOLDER="Frameworks/$FULL_MODULE_NAME.framework"

    # If the framework folder doesn't exist, create it.
    if [ ! -d "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER" ]; then
        echo "Creating framework for $RELATIVE_EXT"
        mkdir -p "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER"
        cp "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist" "
↳ $CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleExecutable -string "$FULL_MODULE_NAME"
↳ "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleIdentifier -string "$FRAMEWORK_BUNDLE_
↳ ID" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
    fi

    echo "Installing binary for $FRAMEWORK_FOLDER/$FULL_MODULE_NAME"

```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```

mv "$FULL_EXT" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/$FULL_
↪MODULE_NAME"
# Create a placeholder .fwork file where the .so was
echo "$FRAMEWORK_FOLDER/$FULL_MODULE_NAME" > ${FULL_EXT%.so}.fwork
# Create a back reference to the .so file location in the framework
echo "${RELATIVE_EXT%.so}.fwork" > "$CODESIGNING_FOLDER_PATH/
↪$FRAMEWORK_FOLDER/$FULL_MODULE_NAME.origin"
}

PYTHON_VER=$(ls -1 "$CODESIGNING_FOLDER_PATH/python/lib")
echo "Install Python $PYTHON_VER standard library extension modules...
↪"
find "$CODESIGNING_FOLDER_PATH/python/lib/$PYTHON_VER/lib-dynload" -
↪name "*.so" | while read FULL_EXT; do
    install_dylib python/lib/$PYTHON_VER/lib-dynload/ "$FULL_EXT"
done

# Clean up dylib template
rm -f "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist"

echo "Signing frameworks as $EXPANDED_CODE_SIGN_IDENTITY_NAME (
↪$EXPANDED_CODE_SIGN_IDENTITY)..."
find "$CODESIGNING_FOLDER_PATH/Frameworks" -name "*.framework" -exec /
↪usr/bin/codesign --force --sign "$EXPANDED_CODE_SIGN_IDENTITY" $
↪{OTHER_CODE_SIGN_FLAGS:-} -o runtime --timestamp=none --preserve-
↪metadata=identifier,entitlements,flags --generate-entitlement-der "{}
↪" \;
```

10. Add Objective C code to initialize and use a Python interpreter in embedded mode. You should ensure that:

- UTF-8 mode (PyPreConfig.utf8_mode) is *enabled*;
- Buffered stdio (PyConfig.buffered_stdio) is *disabled*;
- Writing bytecode (PyConfig.write_bytecode) is *disabled*;
- Signal handlers (PyConfig.install_signal_handlers) are *enabled*;
- System logging (PyConfig.use_system_logger) is *enabled* (optional, but strongly recommended; this is enabled by default);
- `PYTHONHOME` for the interpreter is configured to point at the python subfolder of your app's bundle; and
- The `PYTHONPATH` for the interpreter includes:
 - the python/lib/python3.X subfolder of your app's bundle,
 - the python/lib/python3.X/lib-dynload subfolder of your app's bundle, and
 - the app subfolder of your app's bundle

Your app's bundle location can be determined using `[[NSBundle mainBundle] resourcePath]`.

Steps 8, 9 and 10 of these instructions assume that you have a single folder of pure Python application code, named app. If you have third-party binary modules in your app, some additional steps will be required:

- You need to ensure that any folders containing third-party binaries are either associated with the app target, or copied in as part of step 8. Step 8 should also purge any binaries that are not appropriate for the platform a specific build is targeting (i.e., delete any device binaries if you're building an app targeting the simulator).
- Any folders that contain third-party binaries must be processed into framework form by step 9. The invocation of `install_dylib` that processes the `lib-dynload` folder can be copied and adapted for this purpose.

- If you're using a separate folder for third-party packages, ensure that folder is included as part of the `PYTHONPATH` configuration in step 10.
- If any of the folders that contain third-party packages will contain `.pth` files, you should add that folder as a *site directory* (using `site.addsitedir()`), rather than adding to `PYTHONPATH` or `sys.path` directly.

7.2.3 Testing a Python package

The CPython source tree contains a [testbed project](#) that is used to run the CPython test suite on the iOS simulator. This testbed can also be used as a testbed project for running your Python library's test suite on iOS.

After building or obtaining an iOS XCFramework (See [iOS/README.rst](#) for details), create a clone of the Python iOS testbed project by running:

```
$ python iOS/testbed clone --framework <path/to/Python.xcframework> --app  
→<path/to/module1> --app <path/to/module2> app-testbed
```

You will need to modify the `iOS/testbed` reference to point to that directory in the CPython source tree; any folders specified with the `--app` flag will be copied into the cloned testbed project. The resulting testbed will be created in the `app-testbed` folder. In this example, the `module1` and `module2` would be importable modules at runtime. If your project has additional dependencies, they can be installed into the `app-testbed/iOSTestbed/app_packages` folder (using `pip install --target app-testbed/iOSTestbed/app_packages` or similar).

You can then use the `app-testbed` folder to run the test suite for your app. For example, if `module1.tests` was the entry point to your test suite, you could run:

```
$ python app-testbed run -- module1.tests
```

This is the equivalent of running `python -m module1.tests` on a desktop Python build. Any arguments after the `--` will be passed to the testbed as if they were arguments to `python -m` on a desktop machine.

You can also open the testbed project in Xcode by running:

```
$ open app-testbed/iOSTestbed.xcodeproj
```

This will allow you to use the full Xcode suite of tools for debugging.

The arguments used to run the test suite are defined as part of the test plan. To modify the test plan, select the test plan node of the project tree (it should be the first child of the root node), and select the «Configurations» tab. Modify the «Arguments Passed On Launch» value to change the testing arguments.

The test plan also disables parallel testing, and specifies the use of the `iOSTestbed.lldbinit` file for providing configuration of the debugger. The default debugger configuration disables automatic breakpoints on the `SIGINT`, `SIGUSR1`, `SIGUSR2`, and `SIGXFSZ` signals.

7.3 App Store Compliance

The only mechanism for distributing apps to third-party iOS devices is to submit the app to the iOS App Store; apps submitted for distribution must pass Apple's app review process. This process includes a set of automated validation rules that inspect the submitted application bundle for problematic code.

The Python standard library contains some code that is known to violate these automated rules. While these violations appear to be false positives, Apple's review rules cannot be challenged; so, it is necessary to modify the Python standard library for an app to pass App Store review.

The Python source tree contains a [patch file](#) that will remove all code that is known to cause issues with the App Store review process. This patch is applied automatically when building for iOS.

Επεξεργαστές Κειμένου και IDEs

Υπάρχουν αρκετά IDEs που υποστηρίζουν τη γλώσσα προγραμματισμού Python. Αρκετοί επεξεργαστές κειμένου και IDEs παρέχουν επισήμανση σύνταξης, εργαλεία αποσφαλμάτωσης και ελέγχους [PEP 8](#).

8.1 IDLE — Επεξεργαστής και shell της Python

Το IDLE είναι το Ολοκληρωμένο Περιβάλλον Ανάπτυξης και Εκμάθησης της Python και γενικά συνοδεύει τις εγκαταστάσεις της Python. Αν χρησιμοποιείτε Linux και δεν έχετε εγκατεστημένο το IDLE δείτε [Installing IDLE on Linux](#). Για περισσότερες πληροφορίες δείτε το IDLE docs.

8.2 Άλλοι Επεξεργαστές Κειμένου και IDEs

Το wiki της κοινότητας της Python έχει πληροφορίες που έχουν υποβληθεί από την κοινότητα σχετικά με τους επεξεργαστές κειμένου και IDEs. Παρακαλούμε μεταβείτε στο [Python Editors](#) και [Integrated Development Environments](#) για ένα πλήρη κατάλογο.

>>>

Η προεπιλεγμένη Python εντολή του *interactive* shell. Συχνά εμφανίζεται για παραδείγματα κώδικα που μπορούν να εκτελεστούν διαδραστικά στον interpreter.

...

Μπορεί να αναφέρεται σε:

- Η προεπιλεγμένη Python εντολή του *interactive* shell κατά την εισαγωγή του κώδικα για ένα μπλοκ κώδικα με εσοχή, όταν βρίσκεται μέσα σε ένα ζεύγος ταιριασμένων αριστερών και δεξιών delimiters (παρενθέσεις, αγκύλες, άγκιστρα ή τριπλά εισαγωγικά), ή μετά τον καθορισμό ενός decorator.
- Η ενσωματωμένη σταθερά Ellipsis.

αφηρημένη βασική κλάση

Οι αφηρημένες βασικές κλάσεις συμπληρώνουν το *duck-typing* παρέχοντας έναν τρόπο ορισμού interfaces όταν άλλες τεχνικές όπως η `hasattr()` θα ήταν αδέξιες ή ανεπαίσθητα λανθασμένες (για παράδειγμα με magic methods). Τα ABC (abstract base class) εισάγουν εικονικές υποκλάσεις, οι οποίες είναι κλάσεις που δεν κληρονομούνται από μια κλάση, αλλά εξακολουθούν να αναγνωρίζονται από το `isinstance()` και από το `issubclass()` βλ. την τεκμηρίωση του module `abc`. Η Python διαθέτει πολλά ενσωματωμένα ABC για δομές δεδομένων (στο module `collections.abc`), αριθμούς (στο module `numbers`), ροές (στο module μονάδα `io`), εισαγωγή finders και loaders (στο module `importlib.abc`). Μπορείτε να δημιουργήσετε τα δικά σας ABC με το module `abc`.

συνάρτηση annotate

Μια συνάρτηση που μπορεί να κληθεί για να ανακτήσει το *annotations* ενός αντικειμένου. Αυτή η συνάρτηση είναι προσβάσιμη ως το χαρακτηριστικό `__annotate__` των συναρτήσεων, των κλάσεων και των modules. Οι συναρτήσεις `annotate` είναι ένα υποσύνολο του *evaluate functions*.

annotation

Μια ετικέτα που σχετίζεται με μια μεταβλητή, ένα χαρακτηριστικό κλάσης ή μια παράμετρος συνάρτησης ή τιμή που επιστρέφεται, που χρησιμοποιείται κατά σύμβαση ως *type hint*.

Δεν είναι δυνατή η πρόσβαση στα annotations των τοπικών μεταβλητών κατά το χρόνο εκτέλεσης, αλλά τα annotations των global μεταβλητών, των χαρακτηριστικών κλάσης και των συναρτήσεων μπορούν να ανακτηθούν καλώντας την εντολή `annotationlib.get_annotations()` σε modules, κλάσεις και συναρτήσεις, αντίστοιχα.

Βλ. τα *variable annotation*, *function annotation*, **PEP 484**, **PEP 526** και **PEP 649**, τα οποία περιγράφουν την λειτουργικότητα. Δείτε επίσης τα annotations-howto για τις βέλτιστες πρακτικές δουλεύοντας με

annotations.

όρισμα

Μια τιμή μεταβιβάζεται σε μία *function* (ή *method*) κατά την κλήση της συνάρτησης. Υπάρχουν δύο είδη ορισμάτων:

- *keyword argument*: ένα όρισμα πριν από ένα αναγνωριστικό (π.χ. `name=`) σε μια κλήση συνάρτησης ή περνώντας το ως τιμή σε ένα λεξικό πριν από `**`. Για παράδειγμα, το 3 και το 5 αποτελούν ορίσματα λέξεων-κλειδιών στις ακόλουθες κλήσεις προς `complex()`:

```
complex(real=3, imag=5)
complex(**{'real': 3, 'imag': 5})
```

- *positional argument*: ένα όρισμα που δεν είναι όρισμα keyword. Τα ορίσματα θέσης μπορούν να εμφανίζονται στην αρχή μιας λίστας ορισμάτων ή/και να μεταβιβάζονται ως στοιχεία ενός *iterable* πριν από `*`. Για παράδειγμα, το 3 και το 5 αποτελούν ορίσματα θέσης στις παρακάτω κλήσεις:

```
complex(3, 5)
complex(*(3, 5))
```

Τα ορίσματα εκχωρούνται στις ονομασμένες τοπικές μεταβλητές στο σώμα μια συνάρτησης. Βλ. την ενότητα *calls* για τους κανόνες που διέπουν αυτήν την εκχώρηση. Συντακτικά, οποιαδήποτε έκφραση μπορεί να χρησιμοποιηθεί για να αναπαραστήσει ένα όρισμα η αξιολογούμενη τιμή εκχωρείται σε μια τοπική μεταβλητή.

Βλ. επίσης την εγγραφή του γλωσσarium για το *parameter*, την FAQ ερώτηση στο η διαφορά μεταξύ ορισμάτων και παραμέτρων, και **PEP 362**.

ασύγχρονος διαχειριστής context

Ένα αντικείμενο που ελέγχει το ορατό περιβάλλον σε μια δήλωση `async with` ορίζοντας τις μεθόδους `__aenter__()` και `__aexit__()`. Που εισήχθη από **PEP 492**.

ασύγχρονος generator

Μια συνάρτηση που επιστρέφει έναν *asynchronous generator iterator*. Μοιάζει με μια συνάρτηση *coroutine* που ορίζεται με `async def` εκτός από ότι περιέχει εκφράσεις `yield` για την παραγωγή μιας σειράς τιμών που μπορούν να χρησιμοποιηθούν σε έναν `async for` βρόχο.

Συνήθως αναφέρεται σε μια συνάρτηση ασύγχρονου generator, αλλά μπορεί να αναφέρεται σε έναν *ασύγχρονο generator iterator* σε ορισμένα contexts. Σε περιπτώσεις όπου το επιδιωκόμενο νόημα δεν είναι σαφές, με την χρήση των πλήρων όρων αποφεύγεται η ασάφεια.

Μια συνάρτηση ασύγχρονου generator μπορεί να περιέχει εκφράσεις `await`, καθώς και δηλώσεις `async for`, και `async with`.

ασύγχρονος generator iterator

An object created by an *asynchronous generator* function.

Αυτός είναι ένας *asynchronous iterator* που όταν καλείται χρησιμοποιώντας την μέθοδο `__anext__()` επιστρέφει ένα αναμενόμενο αντικείμενο που θα εκτελέσει στο σώμα της συνάρτησης του ασύγχρονου generator μέχρι την επόμενη `yield` έκφραση.

Κάθε `yield` αναστέλλει προσωρινά την επεξεργασία, θυμάται την κατάσταση εκτέλεσης (συμπεριλαμβανομένων των τοπικών μεταβλητών και των δηλώσεων `try` σε εκκρεμότητα). Όταν ο *ασύγχρονος generator iterator* συνεχίσει αποτελεσματικά με άλλο αναμενόμενο που επιστρέφεται από `:pep: 492()` και **PEP 525**.

ασύγχρονος iterable

Ένα αντικείμενο, που μπορεί να χρησιμοποιηθεί σε μια δήλωση `async for`. Πρέπει να επιστρέφει ένα *asynchronous iterator* από την μέθοδο `__aiter__()`. Που εισήχθη από **PEP 492**.

ασύγχρονος iterator

Ένα αντικείμενο που υλοποιεί τις μεθόδους `__aiter__()` και `__anext__()`. Η μέθοδος `__anext__()` πρέπει να επιστρέφει ένα *awaitable* αντικείμενο. Το `async for` επιλύει τα αναμενόμενα που επιστρέφονται από τη μέθοδο `__anext__()` ενός ασύγχρονου iterator έως ότου εγείρει μια εξαίρεση `StopAsyncIteration`. Εισήχθη από **PEP 492**.

κατάσταση συνδεδεμένου νήματος

Ένα *thread state* που είναι ενεργή για το τρέχον νήμα του λειτουργικού συστήματος.

Όταν επισυνάπτεται ένας *thread state*, το νήμα του λειτουργικού συστήματος έχει πρόσβαση στο πλήρες Python C API και μπορεί να καλέσει με ασφάλεια τον διερμηνέα bytecode.

Εκτός εάν μια συνάρτηση αναφέρει ρητά το αντίθετο, η προσπάθεια κλήσης του C API χωρίς μια συνημμένη κατάσταση νήματος θα οδηγήσει ένα μοιραίο σφάλμα ή σε απροσδιόριστη συμπεριφορά. Μια κατάσταση νήματος μπορεί να συνδεθεί και να αποσυνδεθεί ρητά από τον χρήστη μέσω του C API ή έμμεσα από τον χρόνο εκτέλεσης, συμπεριλαμβανομένων των κλήσεων αποκλεισμού C και από τον διερμηνέα bytecode μεταξύ των κλήσεων.

Στις περισσότερες εκδόσεις της Python, η ύπαρξη μιας κατάσταση συνδεδεμένου νήματος υπονοεί ότι ο καλών διατηρεί την *GIL* για τον τρέχοντα διερμηνέα, επομένως μόνο ένα νήμα λειτουργικού συστήματος μπορεί να έχει μια κατάσταση συνδεδεμένου νήματος σε μια δεδομένη στιγμή. Στις εκδόσεις *free-threaded* της Python, τα νήματα μπορούν να διατηρούν ταυτόχρονα μια κατάσταση συνδεδεμένου νήματος, επιτρέποντας την πραγματική παραλληλία του διερμηνέα bytecode.

χαρακτηριστικό

Μια τιμή που σχετίζεται με ένα αντικείμενο που συνήθως αναφέρεται με όνομα χρησιμοποιώντας εκφράσεις με κουκκίδες. Για παράδειγμα, εάν ένα αντικείμενο *o* έχει ένα χαρακτηριστικό *a* θα αναφέρεται ως *o.a*.

Είναι δυνατό να δώσουμε σε ένα αντικείμενο ένα χαρακτηριστικό που το όνομα του δεν είναι αναγνωριστικό όπως ορίζεται από identifiers, για παράδειγμα χρησιμοποιώντας `setattr()`, αν επιτρέπεται από το αντικείμενο. Ένα τέτοιο χαρακτηριστικό δεν θα είναι προσβάσιμο χρησιμοποιώντας τις τελείες, και αντί αυτού θα πρέπει να ανακτηθεί χρησιμοποιώντας `getattr()`.

awaitable

Ένα αντικείμενο που μπορεί να χρησιμοποιηθεί στην έκφραση `await`. Μπορεί να είναι *coroutine* ή ένα αντικείμενο με μια `__await__()` μέθοδο. Βλ. επίσης [PEP 492](#).

BDFL

Ακρωνύμιο του *Benevolent Dictator For Life*, καλοκάγαθος δικτάτορας της ζωής, δηλαδή [Guido van Rossum](#), ο δημιουργός της Python.

δυναδικό αρχείο

Ένα *file object* ικανό να διαβάσει και να γράφει *δυναδικού τύπου αντικείμενα*. Παραδείγματα δυναδικών αρχείων είναι αρχεία που ανοίγουν σε δυναδική λειτουργία ('rb', 'wb' ή 'rb+'), `sys.stdin.buffer`, `sys.stdout.buffer`, και στιγμιотύπων των `io.BytesIO` και `gzip.GzipFile`.

Βλ. επίσης *text file* για ένα αντικείμενο τύπου αρχείο ικανό να διαβάσει και να γράψει `str` αντικείμενα.

δανεική αναφορά

Στο C API της Python, μια δανεική αναφορά είναι μια αναφορά σε ένα αντικείμενο, όπου ο κώδικας που χρησιμοποιεί το αντικείμενο δεν κατέχει την αναφορά. Γίνεται ένας αχρησιμοποίητος δείκτης εάν το αντικείμενο καταστραφεί. Για παράδειγμα, μια διαδικασία *garbage collection* μπορεί να αφαιρέσει το τελευταίο *strong reference* από το αντικείμενο και έτσι να το καταστρέψει.

Συνίσταται η κλήση του `Py_INCREF()` στο *δανεική αναφορά* με σκοπό να μετατραπεί σε ένα *ισχυρή αναφορά* επιτόπου, εκτός όταν το αντικείμενο δεν μπορεί να καταστραφεί πριν από την τελευταία χρήση της δανεικής αναφοράς. Η συνάρτηση `Py_NewRef()` μπορεί να χρησιμοποιηθεί ώστε να δημιουργηθεί ένα *ισχυρή αναφορά*.

bytes-like αντικείμενα

Ένα αντικείμενο που υποστηρίζει το `bufferobjects` και μπορεί να εξάγει ένα *C-contiguous* buffer. Αυτό περιλαμβάνει όλα τα αντικείμενα `bytes`, `bytearray`, και `array.array`, καθώς και πολλά κοινά `memoryview` αντικείμενα. Τα δυναδικού τύπου (bytes-like) αντικείμενα μπορούν να χρησιμοποιηθούν για διάφορες λειτουργίες που διαχειρίζονται δυναδικά δεδομένα—αυτά περιλαμβάνουν συμπίεση αποθήκευση σε δυναδικό αρχείο και αποστολή μέσω `socket`.

Ορισμένες λειτουργίες χρειάζονται τα δυναδικά δεδομένα να είναι μεταβλητά. Η τεκμηρίωση συχνά αναφέρεται σε αυτά ως «δυναδικά αντικείμενα ανάγνωσης-εγγραφής» (read-write bytes-like objects). Παραδείγματα μεταβλητών αντικειμένων προσωρινής αποθήκευσης περιέχουν `bytearray` και ένα

`memoryview` ενός `bytearray`. Άλλες λειτουργίες απαιτούν την αποθήκευση των δυαδικών δεδομένων σε αμετάβλητα αντικείμενα («δυαδικά αντικείμενα μόνο ανάγνωσης» (read-only bytes-like objects) παραδείγματα αυτών περιέχουν `bytes` και ένα `memoryview` ενός `bytes` αντικειμένου.

bytecode

Ο πηγαίος κώδικας της Python μεταγλωττίζεται σε *bytecode*, η εσωτερική αναπαράσταση ενός προγράμματος Python στον διερμηνέα CPython. Το *bytecode* αποθηκεύεται επίσης προσωρινά ως `.pyc` αρχεία ώστε η εκτέλεση του ίδιου αρχείου να είναι γρηγορότερη την δεύτερη φορά εκτέλεσης (μπορεί να αποφευχθεί η εκ νέου μεταγλώττιση από τον πηγαίο κώδικα σε *bytecode*). Αυτή η «ενδιάμεση γλώσσα» λέγεται ότι τρέχει σε μια *virtual machine* που εκτελεί τον κώδικα μηχανής που αντιστοιχεί σε κάθε *bytecode*. Λάβετε υπόψη ότι τα *bytecode* δεν αναμένεται να λειτουργούν μεταξύ διαφορετικών εικονικών μηχανών Python, ούτε να είναι σταθερά μεταξύ των εκδόσεων της Python.

Μια λίστα από οδηγίες σχετικά με τα *bytecode* μπορεί να βρεθεί στην τεκμηρίωση για το module `dis`.

callable

Ένα callable είναι ένα αντικείμενο που μπορεί να καλεστεί, πιθανά με ένα σύνολο ορισμάτων (βλ. *argument*), με την παρακάτω σύνταξη:

```
callable(argument1, argument2, argumentN)
```

Μια *function*, και κατ' επέκταση μια *method* είναι callable. Ένα στιγμιότυπο μια κλάσης που υλοποιεί τη μέθοδο `__call__()` είναι επίσης callable.

callback

Μια subroutine συνάρτηση η οποία μεταβιβάζεται ως όρισμα που θα εκτελεστεί κάποια στιγμή στο μέλλον.

κλάση

Ένα πρότυπο για τη δημιουργία αντικειμένων που ορίζονται από το χρήστη. Οι ορισμοί κλάσεων συνήθως περιέχουν ορισμούς μεθόδων που λειτουργούν σε στιγμιότυπα της κλάσης.

μεταβλητή κλάσης

Μια μεταβλητή που ορίζεται σε μια κλάση και προορίζεται να τροποποιηθεί μόνο σε επίπεδο κλάσης (δηλ. όχι σε ένα στιγμιότυπο μιας κλάσης).

μεταβλητή κλεισίματος

Ένας *free variable* που αναφέρεται από ένα *nested scope* και ορίζεται σε μια εξωτερική περιοχή, αντί να επιλύεται δυναμικά κατά την εκτέλεση από τα καθολικά ή ενσωματωμένα namespaces. Μπορεί να δηλωθεί ρητά με τη δεσμευμένη λέξη-κλειδί `nonlocal` ώστε να επιτραπεί η εγγραφή, ή να θεωρηθεί ότι ορίζεται έμμεσα όταν η μεταβλητή χρησιμοποιείται μόνο για ανάγνωση.

Για παράδειγμα, η συνάρτηση `inner` του παρακάτω κώδικα, τόσο η `x` όσο και η `print` είναι *free variables*, αλλά μόνο η `x` είναι μια μεταβλητή κλεισίματος:

```
def outer():
    x = 0
    def inner():
        nonlocal x
        x += 1
        print(x)
    return inner
```

Λόγο του χαρακτηριστικού `codeobject.co_freevars` (το οποίο, παρά την ονομασία του, περιλαμβάνει μόνο τα ονόματα των μεταβλητών κλεισίματος και όχι όλες τις αναφερόμενες ελεύθερες μεταβλητές), χρησιμοποιείται μερικές φορές ο πιο γενικός όρος *free variable* ακόμη και όταν γίνεται ειδική αναφορά σε μεταβλητές κλεισίματος.

μυγαδικός αριθμός

Μια επέκταση του γνωστού συστήματος πραγματικών αριθμών στο οποίο όλοι οι αριθμοί εκφράζονται ως άθροισμα ενός πραγματικού μέρους και ενός φανταστικού μέρους. Οι φανταστικοί αριθμοί είναι πραγματικά πολλαπλάσια της φανταστικής μονάδα (η τετραγωνική ρίζα του -1), που συχνά γράφονται i στα μαθηματικά ή j στη μηχανική. Η Python έχει ενσωματωμένη υποστήριξη για μυγαδικούς

αριθμούς, οι οποίοι γράφονται με αυτόν τον τελευταίο συμβολισμό” το φανταστικό μέρος γράφεται με το επίθημα `j`, π.χ., `3+1j`. Για να αποκτήσετε πρόσβαση σε σύνθετα ισοδύναμα το `module math`, χρησιμοποιήστε το `cmath`. Η χρήση μιγαδικών αριθμών είναι ένα αρκετά προηγμένο μαθηματικό χαρακτηριστικό, εάν δεν γνωρίζετε την ανάγκη τους, είναι σχεδόν σίγουρο ότι μπορείτε να τα αγνοήσετε με ασφάλεια.

context

Αυτό ο όρος έχει διαφορετικές σημασίες ανάλογα με το πού και πώς χρησιμοποιείται. Μερικές κοινές έννοιες:

- Η προσωρινή κατάσταση ή το περιβάλλον που δημιουργείται από έναν *context manager* μέσω μιας δήλωσης `with`.
- Το σύνολο των δεσμευμένων κλειδιού-τιμής που σχετίζονται με ένα συγκεκριμένο αντικείμενο `contextvars.Context` και προσπελάζονται μέσω αντικειμένων `ContextVar`. Βλ. επίσης *context variable*.
- Ένα αντικείμενο `contextvars.Context`. Βλ. επίσης *current context*.

πρωτόκολλο διαχείρισης περιβάλλοντος

Οι μέθοδοι `__enter__()` και `__exit__()` καλούνται από τη δήλωση `with`. Βλ. **PEP 343**.

διαχειριστής context

Ένα αντικείμενο που υλοποιεί το *context management protocol* και ελέγχει το περιβάλλον που είσαι ορατό μέσα σε μια δήλωση `with`. Βλ. **PEP 343**.

context μεταβλητή

Μια μεταβλητή της οποίας η τιμή εξαρτάται από το ποιο είναι το *current context*. Οι τιμές προσπελάζονται μέσω των αντικειμένων `contextvars.ContextVar`. Οι μεταβλητές συμφραζόμενων χρησιμοποιούνται κυρίως για να απομονώσουν την κατάσταση μεταξύ ταυτόχρονων ασύγχρονων εργασιών.

contiguous

Ένα buffer θεωρείται *contiguous* ακριβώς εάν είναι είτε *C-contiguous* είτε *Fortran contiguous*. Το buffer μηδενικών διαστάσεων είναι C και Fortran contiguous. Σε μονοδιάστατους πίνακες, τα στοιχεία πρέπει να τοποθετούνται στη μνήμη το ένα δίπλα στο άλλο, με σειρά αύξησης των δεικτών ξεκινώντας από το μηδέν. Σε πολυδιάστατους C-contiguous πίνακες, ο τελευταίος δείκτης μεταβάλλεται ταχύτερα όταν επισκέπτονται τα στοιχεία σε σειρά διεύθυνσης μνήμης. Ωστόσο, σε Fortran contiguous πίνακες, ο πρώτος δείκτης μεταβάλλεται πιο γρήγορα.

coroutine

Οι *coroutines* είναι μια πιο γενικευμένη μορφή *subroutines*. Οι *subroutines* εισάγονται σε ένα σημείο και εξάγονται σε άλλο σημείο. Οι *coroutines* μπορεί να εισαχθούν, να εξαχθούν και να συνεχιστούν σε πολλά διαφορετικά σημεία. Μπορούν να υλοποιηθούν με την δήλωση `async def`. Βλ. επίσης **PEP 492**.

coroutine συνάρτηση

Μια συνάρτηση που επιστρέφει ένα *coroutine* αντικείμενο. Μια συνάρτηση *coroutine* μπορεί να ορίζεται από τη δήλωση `async def`, και μπορεί να περιέχει `await`, `async for`, και `async with` λέξεις κλειδιά. Αυτές εισήχθησαν από το **PEP 492**.

CPython

Η κανονική υλοποίηση της γλώσσας προγραμματισμού Python, όπως διανέμεται στο python.org. Ο όρος «CPython» χρησιμοποιείται όταν είναι απαραίτητο για την διάκριση αυτής της υλοποίησης από άλλες όπως η *Jython* ή η *IronPython*.

τρέχον πλαίσιο

Το *context* (`contextvars.Context` αντικείμενο) που χρησιμοποιείται αυτή τη στιγμή από τα αντικείμενα `ContextVar` για να προσπελάσει (να πάρει ή να ορίσει) τις τιμές των *context variables*. Κάθε νήμα έχει το δικό του τρέχον συμφραζόμενο Τα πλαίσια για την εκτέλεση ασύγχρονων εργασιών (βλ. *asyncio*) συνδέουν κάθε εργασία με ένα συμφραζόμενο, το οποίο γίνεται το τρέχον συμφραζόμενο όποτε η εργασία ξεκινά ή συνεχίζει την εκτέλεση.

κυκλική απομόνωση

Μια υποομάδα ενός ή περισσότερων αντικειμένων που αναφέρονται μεταξύ τους σχηματίζοντας έναν κύκλο αναφορών, αλλά δεν αναφέρονται από άλλα αντικείμενα εκτός της ομάδας. Ο σκοπός του *cyclic*

garbage collector είναι να εντοπίζει αυτές τις ομάδες και να σπάει τον κύκλο αναφορών ώστε να μπορεί να αποδεσμευτεί η μνήμη.

decorator

Μια συνάρτηση που επιστρέφει μια άλλη συνάρτηση, συνήθως εφαρμόζεται ως μετασχηματισμός συνάρτησης χρησιμοποιώντας την `@wrapper` σύνταξη. Συνηθισμένα παραδείγματα για τους decorators είναι `classmethod()` και `staticmethod()`.

Η σύνταξη του decorator είναι απλώς καλλωπιστική, οι ακόλουθοι δύο ορισμοί συναρτήσεων είναι σημασιολογικά ισοδύναμοι:

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

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

Η ίδια έννοια υπάρχει για τις κλάσεις, αλλά χρησιμοποιείται λιγότερο συχνά εκεί. Βλ. την τεκμηρίωση για function definitions και class definitions για περισσότερα σχετικά με τους decorators.

descriptor

Κάθε αντικείμενο που ορίζει τις μεθόδους `__get__()`, `__set__()`, ή `__delete__()`. Όταν ένα χαρακτηριστικό κλάσης είναι descriptor, η ειδική δεσμευτική του συμπεριφορά ενεργοποιείται κατά την αναζήτηση χαρακτηριστικών. Κανονικά, χρησιμοποιώντας `a.b` για να λάβετε, να ορίσετε ή να διαγράψετε ένα χαρακτηριστικό αναζητά το αντικείμενο με το όνομα `b` στο λεξικό της κλάσης για `a`, αλλά εάν το `b` είναι descriptor, καλείται η αντίστοιχη μέθοδος descriptor. Η κατανόηση των descriptors είναι το κλειδί για την καλύτερη κατανόηση της Python γιατί αυτό αποτελεί την βάση για πολλά χαρακτηριστικά όπως συναρτήσεις, μεθόδους, ιδιότητες, μέθοδοι κλάσης στατικές μέθοδοι, και αναφορά σε σούπερ κλάσεις.

Για περισσότερες πληροφορίες αναφορικά με τις μεθόδους των descriptors, βλ. see descriptors ή το Πρακτικός οδηγός για τη χρήση του Descriptor.

λεξικό

Ένα προσαρμοστικός πίνακα, όπου αυθαίρετα κλειδιά αντιστοιχίζονται σε τιμές. Τα κλειδιά μπορεί να είναι οποιοδήποτε αντικείμενο με μεθόδους `__hash__()` και `__eq__()`. Ονομάζεται ως hash στο Perl.

κατανόηση λεξικού

Ένα συμπαγής τρόπος για να επεξεργαστείτε όλα ή μέρος των στοιχείων σε ένα επαναληπτικό και να επιστραφεί ένα με λεξικό με τα αποτελέσματα. `results = {n: n ** 2 for n in range(10)}` δημιουργεί ένα λεξικό που περιέχει το κλειδί `n` που αντιστοιχίζεται με την τιμή `n ** 2`. Βλ. comprehensions.

όψη λεξικού

Τα αντικείμενα που επιστρέφονται από `dict.keys()`, `dict.values()`, και `dict.items()` καλούνται όψεις λεξικού. Αυτές παρέχουν μια δυναμική όψη των των εγγραφών του λεξικού, που σημαίνει ότι όταν το λεξικό μεταβάλλεται, η όψη αντικατοπτρίζει αυτές τις αλλαγές. Για να αναγκάσετε την όψη λεξικού να γίνει μια πλήρης λίστα χρησιμοποιήστε το `list(dictview)`. Βλ. dict-views.

docstring

Μια literal συμβολοσειρά που εμφανίζεται ως η πρώτη έκφραση σε μια κλάση, συνάρτηση ή module. Ενώ αγνοείται κατά την εκτέλεση της σουίτας, αναγνωρίζεται από τον μεταγλωττιστή και τοποθετείται στο χαρακτηριστικό `__doc__` της κλάσης, της συνάρτησης ή του module που περικλείει. Δεδομένου ότι είναι διαθέσιμο μέσω ενδοσκόπησης, το κανονικό μέρος για την τεκμηρίωση του αντικειμένου.

duck-typing

Ένα στυλ προγραμματισμού που δεν εξετάζει τον τύπο ενός αντικειμένου για να προσδιορίσει αν έχει τη σωστή διεπαφή αντίθετα, η μέθοδος ή το χαρακτηριστικό καλείται απλώς ή χρησιμοποιείται («If it looks like a duck and quacks like a duck, it must be a duck.») Δίνοντας έμφαση στις διεπαφές και όχι σε συγκεκριμένους τύπους, ο καλά σχεδιασμένος κώδικας βελτιώνει την ευελιξία του επιτρέποντας

την πολυμορφική υποκατάσταση. Ο τύπος *duck-typing* αποφεύγει δοκιμές χρησιμοποιώντας `type()` ή `isinstance()`. (Σημείωση, ωστόσο, ότι ο τύπος πάπιας *duck-typing* μπορεί να συμπληρωθεί με *abstract base classes*.) Αντί αυτού, συνήθως χρησιμοποιεί δοκιμές `hasattr()` ή προγραμματισμό *EAFP*.

dunder

An informal short-hand for «double underscore», used when talking about a *special method*. For example, `__init__` is often pronounced «dunder init».

EAFP

Πιο εύκολο να ζητήσεις συγχώρεση παρά άδεια. Αυτό το κοινό στυλ προγραμματισμού σε Python προϋποθέτει την ύπαρξη έγκυρων κλειδιών ή χαρακτηριστικών και συλλαμβάνει εξαιρέσεις εάν η υπόθεση αποδοχθεί εσφαλμένη. Αυτό το καθαρό και γρήγορο στυλ χαρακτηρίζεται από την παρουσία πολλών δηλώσεων `try` και `except`. Η τεχνική έρχεται σε αντίθεση με το στυλ που είναι *LBYL* κοινό σε πολλές άλλες γλώσσες, όπως η C.

αξιολόγηση συνάρτησης

Μια συνάρτηση που μπορεί να κληθεί για να αξιολογήσει ένα αδρανές χαρακτηριστικό ενός αντικειμένου, όπως η τιμή των ψευδωνύμων τύπου που δημιουργούνται με την πρόταση `type`.

έκφραση

Ένα κομμάτι σύνταξης που μπορεί να αξιολογηθεί σε κάποια τιμή. Με άλλα λόγια, μια έκφραση είναι μια συσσώρευση στοιχείων έκφρασης όπως κυριολεξία, ονόματα, πρόσβαση χαρακτηριστικών, τελεστές ή κλήσεις συναρτήσεων που όλες επιστρέφουν μια τιμή. Σε αντίθεση με πολλές άλλες γλώσσες, δεν είναι όλες οι γλωσσικές δομές εκφράσεις. Υπάρχουν επίσης *statements* που δεν μπορούν να χρησιμοποιηθούν ως εκφράσεις, όπως το `while`. Οι αναθέσεις τιμών είναι επίσης δηλώσεις όχι εκφράσεις.

module επέκτασης

Ένα module γραμμένο σε C ή C++, που χρησιμοποιείται από το C API της Python για να αλληλεπιδράσουν με τον πυρήνα και με τον κώδικα του χρήστη.

f-string

f-strings

String literals prefixed with `f` or `F` are commonly called «f-strings» which is short for formatted string literals. See also [PEP 498](#).

αντικείμενο αρχείου

Ένα αντικείμενο που εκθέτει ένα API προσανατολισμένο σε αρχείο (με μεθόδους όπως `read()` ή `write()`) σε έναν υποκείμενο πόρο. Ανάλογα με τον τρόπο που δημιουργήθηκε, ένα αντικείμενο αρχείου μπορεί να μεσολαβήσει στην πρόσβαση σε ένα πραγματικό αρχείο στο δίσκο ή σε άλλο τύπο συσκευής αποθήκευσης ή επικοινωνίας (για παράδειγμα τυπική είσοδος/ έξοδος, in-memory buffers, sockets, pipes, κλπ.). Αντικείμενο αρχείου ονομάζονται επίσης *file-like objects* ή *streams*.

Στην πραγματικότητα υπάρχουν τρεις κατηγορίες αντικειμένων αρχείου *raw δυαδικά αρχεία*, *buffered δυαδικά αρχεία* και *αρχεία κειμένου*. Οι διεπαφές τους ορίζονται στην ενότητα `io`. Ο κανονικός τρόπος για να δημιουργήσετε ένα αντικείμενο αρχείου είναι χρησιμοποιώντας την συνάρτηση `open()`.

αντικείμενο που μοιάζει με αρχείο

Ένα συνώνυμο με το *file object*.

κωδικοποίηση συστήματος αρχείων και χειριστής σφαλμάτων

Η κωδικοποίηση και ο χειριστής σφαλμάτων χρησιμοποιείται από την Python για την αποκωδικοποίηση των bytes από το λειτουργικό σύστημα και την κωδικοποίηση σε Unicode για το λειτουργικό σύστημα.

Η κωδικοποίηση συστήματος αρχείων μπορεί να εγγραφεί την επιτυχημένη αποκωδικοποίηση όλων των bytes κάτω από 128. Εάν η κωδικοποίηση συστήματος αρχείων δεν παρέχει αυτήν την εγγύηση, οι συναρτήσεις API μπορούν να εγείρουν ένα `UnicodeError`.

Οι συναρτήσεις `sys.getfilesystemencoding()` και `sys.getfilesystemencodeerrors()` μπορούν να χρησιμοποιηθούν για να λάβετε την κωδικοποίηση του συστήματος αρχείων και του χειριστή σφαλμάτων.

Ο *filesystem encoding and error handler* διαμορφώνονται κατά την εκκίνηση της Python από τη συνάρτηση `PyConfig_Read()` βλ. `filesystem_encoding` και `filesystem_errors` μέλη του `PyConfig`.

Βλ. επίσης το *locale encoding*.

finder

Ένα αντικείμενο που προσπαθεί να βρει το *loader* για ένα module που εισήχθη.

Υπάρχουν δύο τύποι finder: *finders μετα διαδρομής* για χρήση με `sys.meta_path`, και *finders εισόδου διαδρομής* για χρήση με `sys.path_hooks`.

Βλ. `finders-and-loaders` και `importlib` για περισσότερες λεπτομέρειες.

ακέραια διαίρεση

Η μαθηματική διαίρεση που στρογγυλοποιεί προς τα κάτω στον κοντινότερο ακέραιο. Ο τελεστής ακεράιας διαίρεσης είναι `//`. Για παράδειγμα, η έκφραση `11 // 4` αξιολογείται σε 2 σε αντίθεση με την τιμή 2.75 που επιστρέφεται από την διαίρεση με υποδιαστολή. Σημείωση ότι `(-11) // 4` κάνει -3 επειδή αυτή είναι η στρογγυλοποίηση προς τα κάτω του -2.75. Βλ. **PEP 238**.

δωρεάν νήμα

Ένα μοντέλο νημάτων όπου πολλά νήματα μπορούν να εκτελούν Python bytecode ταυτόχρονα μέσα στον ίδιο διερμηνέα. Αυτό έρχεται σε αντίθεση με το *global interpreter lock*, το οποίο επιτρέπει σε ένα μόνο νήμα να εκτελεί Python bytecode κάθε φορά. Δείτε το **PEP 703**.

δωρεάν μεταβλητή

Τυπικά, όπως ορίζεται στο language execution model, μια ελεύθερη μεταβλητή είναι οποιαδήποτε μεταβλητή χρησιμοποιείται σε ένα namespace που δεν είναι τοπική μεταβλητή σε εκείνο το namespace. Δείτε το *closure variable* για παράδειγμα. Πρακτικά, λόγω του ονόματος του χαρακτηριστικού `codeobject.co_freevars`, ο όρος χρησιμοποιείται επίσης μερικές φορές ως συνώνυμο της *closure variable*.

συνάρτηση

Μια σειρά από δηλώσεις που επιστρέφουν κάποια τιμή σε αυτόν που την κάλεσε. Σε αυτές μπορούν να περαστούν κανένα ή περισσότερα *ορίσματα* που μπορεί να χρησιμοποιηθεί για την εκτέλεση. Βλ. επίσης τις ενότητες *parameter*, *method*, και *the function*.

συνάρτηση annotation

Ένας *annotation* μιας παραμέτρου συνάρτησης ή μιας τιμής επιστροφής.

Οι συναρτήσεις annotations συχνά χρησιμοποιούνται για *υποδείξεις τύπου*: για παράδειγμα, αυτή η συνάρτηση αναμένεται να πάρει δύο ορίσματα `int` και επίσης αναμένεται να έχει μία επιστρεφόμενη τιμή `int`:

```
def sum_two_numbers(a: int, b: int) -> int:
    return a + b
```

Η σύνταξη συνάρτησης annotation αναλύεται στην ενότητα *function*.

Βλ. *variable annotation* και **PEP 484**, που περιγράφει αυτή την λειτουργικότητα. Επίσης βλ. *annotations-howto* για τις καλύτερες πρακτικές δουλεύοντας με annotations.

__future__

Ένα future statement, `from __future__ import <feature>`, καθοδηγεί τον μεταγλωττιστή να μεταγλωττίσει το τρέχον module χρησιμοποιώντας σύνταξη ή σημασιολογία που θα γίνει η τυπική σε μελλοντική έκδοση της Python. Το module `__future__` τεκμηριώνει τις πιθανές τιμές του *feature*. Με την εισαγωγή αυτής της λειτουργικής μονάδας και την αξιολόγηση των μεταβλητών της, μπορείτε να δείτε τότε μια νέα δυνατότητα προστέθηκε για πρώτη φορά στην γλώσσα και τότε θα γίνει (ή έγινε) η προεπιλογή:

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

συλλογή απορριμάτων

Η διαδικασία απελευθέρωσης της μνήμης όταν δεν χρησιμοποιείται άλλο. Η Python εκτελεί συλλογή απορριμάτων μέσω καταμέτρησης αναφορών και ενός κυκλικού συλλέκτη σκουπιδιών που είναι σε

θέση να ανιχνεύει και να σπάει τους κύκλους αναφοράς. Ο συλλέκτης απορριμμάτων μπορεί να ελεγχθεί χρησιμοποιώντας το module `gc`.

generator

Μια συνάρτηση που επιστρέφει ένα *generator iterator*. Μοιάζει με μια κανονική συνάρτηση εκτός από το ότι περιέχει εκφράσεις `yield` για την παραγωγή μιας σειράς τιμών που μπορούν να χρησιμοποιηθούν σε έναν βρόχο `for` ή που μπορούν να ανακτηθούν μία τη φορά με την συνάρτηση `next()` function.

Συνήθως αναφέρεται σε μια συνάρτηση *generator*, αλλά μπορεί να αναφέρεται σε έναν *generator iterator* σε μερικά contexts. Σε περιπτώσεις όπου το επιδιωκόμενο νόημα δεν είναι σαφές, η χρήση των πλήρων όρων αποφεύγει την ασάφεια.

generator iterator

Ένα αντικείμενο που δημιουργείται από μια συνάρτηση *generator*.

Κάθε `yield` αναστέλλει προσωρινά την επεξεργασία, θυμάται την κατάσταση εκτέλεσης (συμπεριλαμβανομένων των τοπικών μεταβλητών και των δηλώσεων δοκιμής σε εκκρεμότητα). Όταν ο *generator iterator* συνεχίσει, συνεχίζει από εκεί που σταμάτησε (σε αντίθεση με τις συναρτήσεις που ξεκινούν από την αρχή σε κάθε επίκληση).

generator έκφραση

Μια *expression* που επιστρέφει έναν *iterator*. Μοιάζει με κανονική έκφραση που ακολουθείται από μια πρόταση `for` που ορίζει μια μεταβλητή βρόχου, ένα εύρος και μια προαιρετική πρόταση `if`. Η συνδυασμένη έκφραση δημιουργεί τιμές για μια συνάρτηση εγκλεισμού:

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

γενική συνάρτηση

Μια συνάρτηση που αποτελείται από πολλαπλές συναρτήσεις που υλοποιούν την ίδια λειτουργία για διαφορετικούς τύπους. Ποια υλοποίηση πρέπει να χρησιμοποιηθεί κατά τη διάρκεια μια κλήσης καθορίζεται από τον αλγόριθμο αποστολής.

Βλ. επίσης την καταχώρηση του *single dispatch*, τον decorator `functools.singledispatch()` και **PEP 443**.

γενικός τύπος

Ένας *type* που μπορεί να παραμετροποιηθεί" συνήθως μια container class, όπως `list` ή `dict`. Χρησιμοποιείται για *type hints* και *annotations*.

Για περισσότερες λεπτομέρειες, βλ. generic alias types **PEP 483**, **PEP 484**, **PEP 585**, και το module `typing`.

GIL

Βλ. *global interpreter lock*.

global interpreter lock

Ο μηχανισμός που χρησιμοποιείται από τον διερμηνέα *CPython* για να διασφαλίσει ότι μόνο ένα νήμα εκτελεί Python *bytecode* κάθε φορά. Αυτό απλοποιεί την υλοποίηση *CPython* δημιουργώντας το μοντέλο αντικειμένου (συμπεριλαμβανομένων κρίσιμων ενσωματωμένων τύπων όπως π.χ. `dict`) έμμεσα ασφαλές έναντι ταυτόχρονης πρόσβασης. Το κλείδωμα ολόκληρου του διερμηνέα διευκολύνει τον διερμηνέα να είναι πολλαπλών νημάτων, εις βάρος του μεγάλου μέρους του παραλληλισμού που παρέχουν οι μηχανές πολλαπλών επεξεργαστών.

Ωστόσο, ορισμένες λειτουργικές μονάδες επέκτασης, είτε τυπικές είτε τρίτων, έχουν σχεδιαστεί έτσι ώστε να απελευθερώνουν το GIL όταν εκτελούν εργασίες εντατικών υπολογισμών όπως συμπίεση ή κατακερματισμός. Επίσης, το GIL απελευθερώνεται πάντα όταν εκτελείτε I/O.

Από την έκδοση Python 3.13, ο GIL μπορεί να απενεργοποιηθεί χρησιμοποιώντας τη ρύθμιση `--disable-gil` κατά τη διαμόρφωση της κατασκευής. Μετά την κατασκευή της Python με αυτήν με αυτήν την επιλογή, ο κώδικας πρέπει να εκτελείται με την επιλογή `-X gil=0` ή αφού ρυθμιστεί η μεταβλητή περιβάλλοντος `PYTHON_GIL=0`. Αυτή η δυνατότητα επιτρέπει βελτιωμένη απόδοση για

εφαρμογές πολλαπλών νημάτων και διευκολύνει τη χρήση των επεξεργαστών πολλαπλών πυρήνων με αποδοτικό τρόπο. Για περισσότερες λεπτομέρειες, δείτε το [PEP 703](#).

Σε προηγούμενες εκδόσεις του C API της Python, μια συνάρτηση μπορεί να δηλώνει ότι απαιτεί την τήρηση του GIL για να χρησιμοποιηθεί. Αυτό αναφέρεται στην ύπαρξη μιας κατάστασης *attached thread state*.

hash-based pyc

Ένα αρχείο κρυφής μνήμης *bytecode* που χρησιμοποιεί τον κατακερματισμό και όχι τον χρόνο τροποποίησης του αντίστοιχου αρχείου προέλευσης για να προσδιορίσει την εγκυρότητα του. Βλ. *pyc-invalidation*.

hashable

Ένα αντικείμενο είναι *hashable* εάν έχει μια τιμή κατακερματισμού που δεν αλλάζει ποτέ κατά τη διάρκεια της ζωής του (χρειάζεται μια μέθοδο `__hash__()`), και μπορεί να συγκριθεί με άλλα αντικείμενα (χρειάζεται μια μέθοδο `__eq__()`). Τα *hashable* αντικείμενα που συγκρίνονται ως προς την ισότητα τους πρέπει να έχουν την ίδια τιμή κατακερματισμού.

Η ύπαρξη *hashable* κάνει ένα αντικείμενο να μπορεί να χρησιμοποιηθεί ως κλειδί λεξικού και ως μέλος ενός συνόλου, επειδή αυτές οι δομές δεδομένων χρησιμοποιούν τιμές κατακερματισμού.

Τα περισσότερα από τα αμετάβλητα ενσωματωμένα αντικείμενα της Python μπορούν να κατακερματιστούν τα μεταβλητά κοντέινερ (όπως οι λίστες ή τα λεξικά) δεν είναι τα αμετάβλητα κοντέινερ (όπως πλειάδες και τα *frozensets*) μπορούν να κατακερματιστούν μόνο εάν τα στοιχεία τους είναι κατακερματισμένα. Τα αντικείμενα που είναι στιγμιότυπα κλάσεων που ορίζονται από το χρήστη μπορούν να κατακερματιστούν από προεπιλογή. Όλα συγκρίνονται άνισα εκτός από τον εαυτό τους) και η τιμή κατακερματισμού τους προέρχεται από το `id()`.

IDLE

Ένα ολοκληρωμένο περιβάλλον ανάπτυξης και μάθησης για την Python. *idle* είναι ένα βασικό περιβάλλον επεξεργασίας και διερμηνείας που συνοδεύεται από την τυπική διανομή της Python.

Αθάνατο

Αθάνατα αντικείμενα είναι μια λεπτομέρεια υλοποίησης της CPython που εισήχθη στην [PEP 683](#).

Εάν ένα αντικείμενο είναι αθάνατο, ο *πλήθος αναφορές* του δεν τροποποιείται, και επομένως δεν εκχωρείται ποτέ ενώ εκτελείται ο διερμηνέας. Για παράδειγμα, `True` και `None` είναι αθάνατα στην CPython.

Τα αθάνατα αντικείμενα μπορούν να αναγνωριστούν μέσω της `sys._is_immortal()`, ή μέσω της `PyUnstable_IsImmortal()` στο C API.

immutable

Ένα αντικείμενο με σταθερή τιμή. Τα αμετάβλητα αντικείμενα περιλαμβάνουν αριθμούς, συμβολοσειρές και πλειάδες. Ένα τέτοιο αντικείμενο δεν μπορεί να αλλάξει. Ένα νέο αντικείμενο πρέπει να δημιουργηθεί εάν πρέπει να αποθηκευτεί μια διαφορετική τιμή. Παίζουν σημαντικό ρόλο σε μέρη όπου μια σταθερά απαιτείται, για παράδειγμα ως κλειδί σε ένα λεξικό.

εισαγόμενο path

Μια λίστα από τοποθεσίες (ή *καταχωρίσεις διαδρομής*) που μπορούν να αναζητηθούν *path based finder* για να εισαχθούν *modules*. Κατά την διαδικασία εισαγωγής, αυτή η λίστα με τοποθεσίες συνήθως έρχεται από `sys.path`, αλλά για τα υποπακέτα μπορεί επίσης να έρθει από το χαρακτηριστικό του πακέτου γονέα `__path__`.

εισαγωγή

Η διαδικασία κατά την οποία ο κώδικας της Python σε ένα *module* είναι διαθέσιμη στον κώδικα Python ενός άλλου *module*.

εισαγωγέας

Ένα αντικείμενο μπορεί και να αναζητεί και να φορτώνει ένα *module* και ένα *finder* και *loader* αντικείμενο.

διαδραστικός

Η Python έχει έναν διαδραστικό διερμηνέα όπου σημαίνει ότι μπορείς να εισάγεις δηλώσεις και εκφράσεις στην εισαγωγή εντολών του διερμηνέα, εκτελώντας τες άμεσα και εμφανίζοντας τα αποτελέσματα.

Απλώς εκκινήστε την python χωρίς ορίσματα (πιθανώς επιλέγοντας το από το κύριο μενού του υπολογιστή σας). Αποτελεί έναν αποδοτικό τρόπο για να δοκιμάσετε νέες ιδέες ή να εξετάσετε modules και πακέτα (θυμηθείτε `help(x)`). Για περισσότερα σχετικά με τη διαδραστική λειτουργία, δείτε `tut-interac`.

interpreted

Η Python είναι μια interpreted γλώσσα, σε αντίθεση με μια μεταγλωττισμένη, αν και η διάκριση μπορεί να είναι και θολή λόγω της παρουσία του bytecode μεταγλωττιστή. Αυτό σημαίνει ότι τα αρχεία προέλευσης μπορούν να εκτελεστούν απευθείας χωρίς να δημιουργηθεί ρητά ένα εκτελέσιμο αρχείο που στην συνέχεια εκτελείται. Οι interpreted γλώσσες συνήθως έχουν μικρότερο κύκλο ανάπτυξης/ εντοπισμού σφαλμάτων από τις μεταγλωττισμένες, αν και τα προγράμματά τους γενικά εκτελούνται πιο αργά. Βλ. επίσης *interactive*.

τερματισμός λειτουργίας διερμηνέα

Όταν ζητείται τερματισμός λειτουργίας, ο διερμηνέας της Python εισέρχεται σε μια ειδική φάση όπου απελευθερώνει σταδιακά όλους τους διατιθέμενους πόρους, όπως λειτουργικές μονάδες και πολλαπλές κρίσιμες εσωτερικές δομές. Επίσης πραγματοποιεί αρκετές κλήσεις στο *συλλέκτης σκουπιδιών*. Αυτό μπορεί να ενεργοποιήσει την εκτέλεση κώδικα σε καταστροφείς που ορίζονται από το χρήστη ή σε callbacks ασθενούς ανταποκρίσεις. Ο κώδικας που εκτελείται κατά τη φάση τερματισμού λειτουργίας μπορεί να συναντήσει διάφορες εξαιρέσεις, καθώς οι πόροι στους οποίους βασίζεται ενδέχεται να μην λειτουργούν πλέον (συνήθη παραδείγματα είναι οι λειτουργικές μονάδες βιβλιοθήκης ή ο μηχανισμός ειδοποιήσεων).

Ο βασικός λόγος τερματισμού λειτουργίας του διερμηνέα είναι ότι το `__main__` module ή ολοκληρώθηκε η εκτέλεση του κώδικα που έτρεχε.

iterable

Ένα αντικείμενο ικανό να επιστρέψει τα μέλη του ένα κάθε φορά. Παραδείγματα iterables περιλαμβάνουν όλους του τύπους ακολουθιών (όπως `list`, `str`, και `tuple`) και μερικούς τύπους μη ακολουθίας όπως `dict`, *αντικείμενο αρχείου*, και αντικείμενα οποιονδήποτε κλάσεων που μπορούν να οριστούν με μια μέθοδο `__iter__()` ή με μία μέθοδο `__getitem__()` που υλοποιεί τη σημασιολογία *sequence*.

Τα iterables μπορούν να χρησιμοποιηθούν σε ένα `for` βρόχο και σε πολλά άλλα σημεία όπου χρειάζεται μια ακολουθία (`zip()`, `map()`, ...). Όταν ένα iterable αντικείμενο μεταβιβάζεται ως όρισμα στην ενσωματωμένη συνάρτηση `iter()`, επιστρέφει έναν iterator για αντικείμενο. Αυτός ο iterator είναι καλός για ένα πέρασμα από ένα σύνολο τιμών. Όταν χρησιμοποιείται επαναληπτικά, συνήθως δεν είναι απαραίτητο να καλέσετε το `iter()` ή να ασχοληθείτε μόνοι σας με αντικείμενα iterator. Η δήλωση `for` το κάνει αυτόματα για εσάς, δημιουργώντας μια προσωρινή μεταβλητή χωρίς όνομα για να κρατά τον iterator για την διάρκεια του βρόχου. Βλ. επίσης *iterator*, *sequence*, και *generator*.

iterator

Ένα αντικείμενο που αντιπροσωπεύει μια ροή δεδομένων. Επαναλαμβανόμενες κλήσεις προς τη μέθοδο `__next__()` του iterator (ή μεταβίβαση του στην ενσωματωμένη συνάρτηση `next()`) επιστρέφουν διαδοχικά στοιχεία στην ροή. Όταν όχι περισσότερα δεδομένα είναι διαθέσιμα εγείρεται μια εξαίρεση `StopIteration`. Σε αυτό το σημείο, το αντικείμενο iterator εξαντλείται και τυχόν περαιτέρω κλήσεις στη μέθοδο `__next__()` απλώς απλά εγείρουν ξανά το `StopIteration`. Οι iterators πρέπει να έχουν μια μέθοδο `__iter__()` που επιστρέφει το ίδιο το αντικείμενο iterator, έτσι ώστε κάθε iterator να είναι επίσης iterable και μπορεί να χρησιμοποιηθεί στα περισσότερα μέρη όπου γίνονται αποδεκτοί και άλλοι iterators. Μια αξιοσημείωτη εξαίρεση είναι ο κώδικας που επιχειρεί πολλαπλά περάσματα iteration. Ένα αντικείμενο κοντέινερ (όπως ένα `list`) παράγει έναν καθαρά νέο iterator κάθε φορά που κάθε φορά που μεταβιβάζεται στην συνάρτηση `iter()` ή τον χρησιμοποιείται σε έναν `for` βρόχο. Εάν επιχειρήσετε αυτό με έναν iterator απλώς θα επιστρέψετε το ίδιο εξαντλημένο αντικείμενο iterator που χρησιμοποιήθηκε στο προηγούμενο πέρασμα iteration, κάνοντας το να φαίνεται σαν ένα άδειο κοντέινερ.

Περισσότερες πληροφορίες μπορούν να βρεθούν στο `typeiter`.

Το CPython δεν εφαρμόζει με συνέπεια την απαίτηση να ορίζει ένας iterator `__iter__()`. Επίσης σημειώστε ότι η έκδοση CPython με ελεύθερη υποστήριξη νημάτων δεν εγγυάται την ασφάλεια νημάτων για διαδικασίες με iterators.

συνάρτηση key

Μια συνάρτηση κλειδί ή μια συνάρτηση ταξινόμησης είναι μια δυνατότητα κλήσης που επιστρέφει μια

τιμή που χρησιμοποιείται για ταξινόμηση ή διάταξη. Για παράδειγμα, `locale.strxfrm()` χρησιμοποιείται για την παραγωγή ενός κλειδιού ταξινόμησης που γνωρίζει τις συμβάσεις ταξινόμησης για συγκεκριμένες τοπικές ρυθμίσεις.

Ένα αριθμός εργαλείων στην Python δέχεται βασικές συναρτήσεις για τον έλεγχο του τρόπου με τον οποίο τα στοιχεία ταξινομούνται ή ομαδοποιούνται. Αυτά περιέχουν `min()`, `max()`, `sorted()`, `list.sort()`, `heapq.merge()`, `heapq.nsmallest()`, `heapq.nlargest()`, και `itertools.groupby()`.

Υπάρχουν διάφοροι τρόποι για να δημιουργήσετε μια συνάρτηση κλειδιού. Για παράδειγμα, η μέθοδος `str.lower()` μπορεί να χρησιμεύσει ως συνάρτηση κλειδί για την περίπτωση μη διάκρισης πεζών-κεφαλαίων. Εναλλακτικά, μια συνάρτηση κλειδιού μπορεί να δημιουργηθεί από μια `lambda` έκφραση όπως `lambda r: (r[0], r[2])`. Επίσης, `operator.attrgetter()`, `operator.itemgetter()` και `operator.methodcaller()` είναι τρεις κατασκευαστές βασικών συναρτήσεων. Βλ. το Ταξινόμηση HOW TO για παραδείγματα δημιουργίας και χρήσης βασικών συναρτήσεων.

όρισμα keyword

Βλ. [argument](#).

lambda

Μια ανώνυμη ενσωματωμένη συνάρτηση που αποτελείται από μια μοναδική *expression* η οποία αξιολογείται όταν καλείται η συνάρτηση. Η σύνταξη για τη δημιουργία μιας συνάρτησης `lambda` είναι `lambda [parameters]: expression`

LBYL

Look before you leap. Αυτό το στυλ κωδικοποίησης ελέγχει ρητά τις προϋποθέσεις πριν πραγματοποιήσει κλήσεις ή αναζητήσεις. Αυτό το στυλ έρχεται σε αντίθεση με την προσέγγιση *EAFP* και χαρακτηρίζεται από την παρουσία πολλών δηλώσεων `if`.

Σε ένα περιβάλλον πολλαπλών νημάτων, η προσέγγιση LBYL μπορεί να διακινδυνεύσει να εισάγει μια συνθήκη αγώνα μεταξύ «the Looking» και «the leaping». Για παράδειγμα ο κώδικας, `if key in mapping: return mapping[key]` μπορεί να αποτύχει εάν ένα άλλο νήμα αφαιρέσει το `key` από το `mapping` μετά τη δοκιμή, αλλά πριν από την αναζήτηση. Αυτό το πρόβλημα μπορεί να λυθεί με κλειδώματα ή χρησιμοποιώντας την προσέγγιση EAFP.

λεξικός αναλυτής

Επίσημη ονομασία για τον *tokenizer* · βλ. *token*.

λίστα

Ένα ενσωματωμένο Python *sequence*. Παρά το όνομα του, μοιάζει περισσότερο με έναν πίνακα σε άλλες γλώσσες παρά με μια συνδεδεμένη λίστα, καθώς η πρόσβαση στα στοιχεία είναι $O(1)$.

list comprehension

Ένα συμπαγής τρόπος για να επεξεργαστείτε όλα ή μέρος των στοιχείων σε μια ακολουθία και να επιστρέψετε μια λίστα με τα αποτελέσματα. `result = ['{:04x}'.format(x) for x in range(256) if x % 2 == 0]` δημιουργεί μια λίστα συμβολοσειρών που περιέχουν ζυγούς δεκαεξαδικούς αριθμούς (0x..) στο εύρος από 0 έως 255. Η πρόταση `if` είναι προαιρετική. Εάν παραλειφθεί, όλα τα στοιχεία στο `range(256)` υποβάλλονται σε επεξεργασία.

loader

Ένα αντικείμενο που φορτώνει ένα module. Πρέπει να ορίζει τις μεθόδους `exec_module()` και `create_module()` για την υλοποίηση της διεπαφής *Loader*. Ένας loader συνήθως επιστρέφεται με ένα *finder*. Δείτε επίσης:

- [finders-and-loaders](#)
- `importlib.abc.Loader`
- **PEP 302**

τοπική κωδικοποίησηση

Στο Unix, είναι η κωδικοποίηση της τοπική ρύθμισης `LC_CTYPE`. Μπορεί να ρυθμιστεί με `locale.setlocale(locale.LC_CTYPE, new_locale)`.

Στα Windows, είναι η code page ANSI (π.χ. "cp1252").

Στο Android και το VxWorks, η Python χρησιμοποιεί το "utf-8" ως τοπική κωδικοποίηση.

`locale.getencoding()` μπορεί να χρησιμοποιηθεί για την ανάκτηση της τοπικής κωδικοποίησης.

Βλ. επίσης το *filesystem encoding and error handler*.

μαγική μέθοδος

Ένα άτυπο συνώνυμο για *special method*.

mapping

Ένα αντικείμενο κοντέινερ που υποστηρίζει αυθαίρετες αναζητήσεις κλειδιών και υλοποιεί τις μεθόδους που καθορίζονται στο `collections.abc.Mapping` ή `collections.abc.MutableMapping` abstract base classes. Τα παραδείγματα περιλαμβάνουν `dict`, `collections.defaultdict`, `collections.OrderedDict` και `collections.Counter`.

meta path finder

Ένας *finder* που επιστράφηκε με αναζήτηση στο `sys.meta_path`. Οι *finders* μετα-διαδρομής σχετίζονται, αλλά διαφέρουν από τα *finders entry διαδρομής*.

Βλ. `importlib.abc.MetaPathFinder` για τις μεθόδους που υλοποιούν οι meta path finders.

μετα-κλάση

Η κλάση μιας κλάσης. Οι ορισμοί κλάσης δημιουργούν ένα όνομα κλάσης, ένα λεξικό κλάσης και μια λίστα βασικών κλάσεων. Η μετα-κλάση είναι υπεύθυνη για την απόκτηση αυτών των τριών ορισμάτων και την δημιουργία της κλάσης. Οι περισσότερες αντικειμενοστρεφείς γλώσσες προγραμματισμού παρέχουν μια προεπιλεγμένη υλοποίηση. Αυτό που κάνει την Python ξεχωριστή είναι ότι είναι δυνατή η δημιουργία προσαρμοσμένων μετακλάσεων. Οι περισσότεροι χρήστες δεν χρειάζονται ποτέ αυτό το εργαλείο, αλλά όταν παραστεί ανάγκη, αυτό το εργαλείο, οι μετα-κλάσεις μπορούν να παρέχουν ισχυρές, κομψές λύσεις. Έχουν χρησιμοποιηθεί για την καταγραφή πρόσβασης χαρακτηριστικών, την προσθήκη ασφάλειας νημάτων, την παρακολούθηση δημιουργίας αντικειμένων, την υλοποίηση *singletons*, και πολλές άλλες εργασίες.

Περισσότερες πληροφορίες μπορούν να βρεθούν στο *metaclasses*.

μέθοδος

Μια συνάρτηση που ορίζεται μέσα στο σώμα μιας κλάσης. Εάν καλείται ως χαρακτηριστικό μιας περίπτωσης αυτής της κλάσης, η μέθοδος θα λάβει αντικείμενο περίπτωσης ως πρώτο της *argument* (το οποίο συνήθως ονομάζεται `self`). Βλ. *function* και *nested scope*.

σειρά ανάλυσης μεθόδων

Η Σειρά Ανάλυσης Μεθόδων είναι η σειρά με την οποία οι βασικές κλάσεις αναζητούνται για ένα μέλος κατά την αναζήτηση. Βλ. `python_2.3_mro` για λεπτομέρειες του αλγορίθμου που χρησιμοποιείται από τον διερμηνέα της Python από την έκδοση 2.3.

module

Ένα αντικείμενο που χρησιμεύει ως οργανωτική μονάδα του κώδικα της Python. Τα modules έχουν έναν χώρο ονομάτων που περιέχει αυθαίρετα αντικείμενα Python. Τα modules φορτώνονται στην Python με την διαδικασία *importing*.

Βλ. επίσης *package*.

τεχνικές προδιαγραφές module

Ένα namespace που περιέχει τις πληροφορίες που σχετίζονται με την εισαγωγή που χρησιμοποιούνται για την φόρτωση ενός module. Μια περίπτωση του `importlib.machinery.ModuleSpec`.

Βλ. επίσης *module-specs*.

MRO

Βλ. *method resolution order*.

mutable

Τα ευμετάβλητα αντικείμενα μπορούν να αλλάξουν τις τιμές αλλά να κρατήσουν τα `id()`. Βλ. επίσης *immutable*.

named tuple

Ο όρος «named tuple» εφαρμόζεται για οποιονδήποτε τύπο ή κλάση που κληρονομείται από την

πλειάδα και των οποίων τα στοιχεία μπορούν να ευρετηριοποιηθούν είναι προσβάσιμα χρησιμοποιώντας επώνυμα χαρακτηριστικά. Ο τύπος ή η κλάση μπορεί να έχει και άλλα χαρακτηριστικά.

Πολλοί ενσωματωμένοι τύποι είναι `named tuples`, συμπεριλαμβανομένων των τιμών που επιστρέφονται από `time.localtime()` και `os.stat()`. Ένα άλλο παράδειγμα είναι το `sys.float_info`:

```
>>> sys.float_info[1]                # indexed access
1024
>>> sys.float_info.max_exp           # named field access
1024
>>> isinstance(sys.float_info, tuple) # kind of tuple
True
```

Ορισμένες αναγνωρισμένες πλειάδες είναι ενσωματωμένοι τύποι (όπως τα παραπάνω παραδείγματα). Εναλλακτικά, μια αναγνωρισμένη πλειάδα μπορεί να δημιουργηθεί από έναν ορισμό κανονικής κλάσης που κληρονομεί από `tuple` και που ορίζει έγκυρα πεδία. Μια τέτοια κλάση μπορεί να είναι γραμμένη με το `xrange` ή μπορεί να δημιουργηθεί κληρονομώντας το `typing.NamedTuple`, ή με την `factory` συνάρτηση `collections.namedtuple()`. Οι τελευταίες τεχνικές προσθέτουν επίσης μερικές επιπλέον μεθόδους που μπορεί να μην βρεθούν σε χειρόγραφες ή ενσωματωμένες πλειάδες με όνομα.

namespace

Το μέρος όπου αποθηκεύεται μια μεταβλητή. Τα namespaces υλοποιούνται ως λεξικά. Υπάρχουν οι τοπικοί, οι καθολικοί και οι ενσωματωμένοι namespaces καθώς και οι ένθετοι namespaces σε αντικείμενα (σε μεθόδους). Για παράδειγμα οι συναρτήσεις `builtins.open` και `os.open()` διακρίνονται από τους χώρους ονομάτων τους. Οι χώροι ονομάτων βοηθούν επίσης την αναγνωσιμότητα και τη συντηρησιμότητα καθιστώντας σαφές ποιο module υλοποιεί μια λειτουργία. Για παράδειγμα, γράφοντας `random.seed()` ή `itertools.islice()` καθιστά σαφές ότι αυτές οι συναρτήσεις υλοποιούνται από τα module `random` και `itertools`, αντίστοιχα.

πακέτο namespace

Ένα *package* που χρησιμεύει μόνο ως κοντέινερ για υποπακέτα. Τα πακέτα χώρου ονομάτων μπορεί να μην έχουν φυσική αναπαράσταση και συγκεκριμένα να μην είναι σαν ένα *regular package* επειδή δεν έχουν το `__init__.py` αρχείο.

Τα πακέτα χώρου ονομάτων επιτρέπουν σε πολλά πακέτα με δυνατότητα εγκατάστασης μεμονωμένα να έχουν ένα κοινό γονικό πακέτο. Διαφορετικά, συνίσταται η χρήση ενός *regular package*.

Για περισσότερες πληροφορίες, δείτε το **PEP 420** και το `reference-namespaces-package`.

Βλ. επίσης *module*.

nested scope

Η δυνατότητα αναφοράς σε μια μεταβλητή σε έναν περικλειόμενο ορισμό. Για παράδειγμα μια συνάρτηση που ορίζεται μέσα σε μια άλλη συνάρτηση μπορεί να αναφέρεται σε μεταβλητές στην εξωτερική συνάρτηση. Σημειώστε ότι τα ένθετα πεδία από προεπιλογή λειτουργούν μόνο για αναφορά και όχι για εκχώρηση. Οι τοπικές μεταβλητές διαβάζονται και γράφονται στο εσωτερικό πεδίο εφαρμογής. Ομοίως, οι καθολικές μεταβλητές διαβάζουν και γράφουν στον καθολικό χώρο ονομάτων. Το `nonlocal` επιτρέπει την εγγραφή σε εξωτερικά πεδία.

κλάση νέου στυλ

Το παλιό όνομα για το είδος των κλάσεων χρησιμοποιείται πλέον για όλα τα αντικείμενα. Σε παλιότερες εκδόσεις της Python, μόνο οι κλάσεις νέου στυλ μπορούσαν να χρησιμοποιήσουν τις νεότερες, ευέλικτες δυνατότητες της Python όπως `__slots__`, `descriptors`, ιδιότητες `__getattr__()`, μέθοδοι κλάσης, και στατικές μέθοδοι.

αντικείμενο

Οποιαδήποτε δεδομένα με κατάσταση (χαρακτηριστικά ή τιμή) και καθορισμένη συμπεριφορά (μέθοδοι). Επίσης, η τελική βασική κλάση οποιασδήποτε *new-style class*.

βελτιστοποιημένο πεδίο ορατότητας (scope)

Ένα πεδίο ορατότητας (scope) όπου τα ονόματα των τοπικών μεταβλητών είναι γνωστό με βεβαιότητα στον μεταγλωττιστή κατά τη μεταγλώττιση του κώδικα, επιτρέποντας τη βελτιστοποίηση της πρόσβασης για ανάγνωση και εγγραφή σε αυτά τα ονόματα. Οι τοπικοί χώροι ονομάτων για συναρτήσεις,

γεννήτριες, συναρτήσεις *coroutine*, συμπτύξεις (*comprehensions*) και εκφράσεις γεννητριών βελτιστοποιούνται με αυτόν τον τρόπο. Σημείωση: οι περισσότερες βελτιστοποιήσεις του διερμηνέα εφαρμόζονται σε όλα τα πεδία ορατότητας· μόνο εκείνες που βασίζονται σε γνωστό σύνολο τοπικών και μη τοπικών μεταβλητών περιορίζονται σε βελτιστοποιημένα πεδία ορατότητας.

πακέτο

Ένα Python *module* που μπορεί να περιέχει *submodules* ή αναδρομικά, υποπακέτα. Τεχνικά, ένα πακέτο είναι μια λειτουργική μονάδα Python με ένα `__path__` χαρακτηριστικό.

Βλ. επίσης *regular package* και *namespace package*.

παράμετρος

Μια έγκυρη οντότητα σε έναν ορισμό *function* (ή μέθοδος) που καθορίζει ένα *argument* (ή σε ορισμένες περιπτώσεις, ορίσματα) που μπορεί να δεχθεί η συνάρτηση. Υπάρχουν πέντε είδη παραμέτρων:

- *λέξη-κλειδί ή θέση*: καθορίζει ένα όρισμα που μπορεί να μεταβιβαστεί είτε *θέσεως* ή ως *όρισμα λέξης-κλειδιού*. Αυτό είναι το προεπιλεγμένο είδος παραμέτρου, για παράδειγμα *foo* και *bar* στα ακόλουθα:

```
def func(foo, bar=None): ...
```

- *θέσεως μόνο*: καθορίζει ένα όρισμα που μπορεί να παρέχεται μόνο από τη θέση. Οι παράμετροι μόνο θέσης μπορούν να οριστούν συμπεριλαμβάνοντας έναν χαρακτήρα / στη λίστα παραμέτρων του ορισμού συνάρτησης μετά από αυτές, για παράδειγμα *posonly1* και *posonly2* στα εξής:

```
def func(posonly1, posonly2, /, positional_or_keyword): ...
```

- *λέξης-κλειδί μόνο*: καθορίζει ένα όρισμα που μπορεί να παρέχεται μόνο με λέξη κλειδί. Οι παράμετροι μόνο για λέξη-κλειδί μπορούν να οριστούν συμπεριλαμβάνοντας μια παράμετρο θέσης ή σκέτο * στη λίστα παραμέτρων του ορισμού συνάρτησης πριν από αυτές, για παράδειγμα *kw_only1* και *kw_only2* στα ακόλουθα:

```
def func(arg, *, kw_only1, kw_only2): ...
```

- *μεταβλητή θέσης*: καθορίζει ότι μπορεί να παρασχεθεί μια αυθαίρετη ακολουθία ορισμάτων θέσης (επιπλέον των ορισμάτων θέσης που είναι ήδη αποδεκτά από άλλες παραμέτρους). Μια τέτοια παράμετρος μπορεί να οριστεί προσαρτώντας το όνομα της παραμέτρου με *, για παράδειγμα *args* στα ακόλουθα:

```
def func(*args, **kwargs): ...
```

- *μεταβλητή λέξη-κλειδί*: καθορίζει ότι μπορούν να παρέχονται αυθαίρετα πολλά ορίσματα λέξης-κλειδιού (επιπλέον των ορισμάτων λέξης κλειδιού που είναι αποδεκτά από άλλες παραμέτρους). Μια τέτοια παράμετρος μπορεί να οριστεί προσαρτώντας το όνομα της παραμέτρου με **, για παράδειγμα *kwargs* όπως παραπάνω.

Οι παράμετροι μπορούν να καθορίσουν τόσο τα προαιρετικά όσο και τα απαιτούμενα ορίσματα, καθώς και προεπιλεγμένες τιμές για ορισμένα προαιρετικά ορίσματα.

Βλ. επίσης την *argument* καταχώριση ευρετηρίου, την ερώτηση FAQ σχετικά με τη διαφορά μεταξύ ορισμάτων και παραμέτρων, την κλάση `inspect.Parameter`, την ενότητα *function* και **PEP 362**.

path entry

Μια μεμονωμένη τοποθεσία στο *import path* την οποία συμβουλεύεται ο *path based finder* για να βρει *modules* για εισαγωγή.

path entry finder

Ένας *finder* που επιστρέφεται από έναν καλούμενο στο `sys.path_hooks` (δηλαδή ένα *path entry hook*) που ξέρει πως να εντοπίζει *modules* με *path entry*.

Βλ. `importlib.abc.PathEntryFinder` για τις μεθόδους που ο *entry finder* διαδρομής υλοποιεί.

path entry hook

Ένα καλούμενο στη λίστα `sys.path_hooks`, το οποίο επιστρέφει ένα *path entry finder* εάν ξέρει πως να βρίσκει module σε μια συγκεκριμένη *path entry*.

path based finder

Ένα από τα προεπιλεγμένα *meta path finders* που αναζητά ένα *import path* για modules.

path-like αντικείμενο

Ένα αντικείμενο που αντιπροσωπεύει ένα path συστήματος αρχείων. Ένα αντικείμενο path είναι είτε ένα αντικείμενο `str` ή `bytes` που αντιπροσωπεύει ένα path ή ένα αντικείμενο που υλοποιεί το πρωτόκολλο `os.PathLike`. Ένα αντικείμενο που υποστηρίζει το πρωτόκολλο `os.PathLike` μπορεί να μετατραπεί σε path συστήματος αρχείων `str` ή `bytes` καλώντας την συνάρτηση `os.fspath()` τα `os.fsdecode()` και `os.fsencode()` μπορούν να χρησιμοποιηθούν για την εγγύηση ενός αποτελέσματος `str` ή `bytes`, αντίστοιχα. Εισήχθη από τον **PEP 519**.

PEP

Πρόταση Βελτίωσης Python. Ένα PEP είναι ένα έγγραφο σχεδιασμού που παρέχει πληροφορίες στην κοινότητα Python ή περιγράφει μια νέα δυνατότητα για την Python ή τις διαδικασίες ή το περιβάλλον της. Τα PEP θα πρέπει να παρέχουν μια συνοπτική τεχνική προδιαγραφή και μια λογική για τα προτεινόμενα χαρακτηριστικά.

Τα PEP προορίζονται να είναι οι κύριοι μηχανισμοί για την πρόταση σημαντικών νέων χαρακτηριστικών, για τη συλλογή πληροφοριών της κοινότητας για ένα ζήτημα και για την τεκμηρίωση των αποφάσεων σχεδιασμού που έχουν εισαχθεί στην Python. Ο συγγραφέας του PEP είναι υπεύθυνος για την οικοδόμηση συναίνεσης εντός της κοινότητας και την τεκμηρίωση αντίθετων απόψεων.

Βλ. **PEP 1**.

τιμήμα

Ένα σύνολο από αρχεία σε έναν μόνο κατάλογο (ενδεχομένως αποθηκευμένο σε αρχείο *zip*) που συμβάλλουν σε ένα namespace πακέτο, όπως ορίζεται στο **PEP 420**.

όρισμα θέσης

Βλ. *argument*.

provisional API

Ένα provisional API είναι αυτό που έχει εσκεμμένα εξαιρεθεί από τις backwards εγγυήσεις συμβατότητας της τυπικής βιβλιοθήκης. Αν και δεν αναμένονται σημαντικές αλλαγές σε τέτοιες διεπαφές, εφόσον επισημαίνονται ως προσωρινές, αλλαγές μη backwards συμβατότητας (μέχρι και κατάργηση της διεπαφής) μπορεί να προκύψουν εάν κριθεί απαραίτητο από τους βασικούς προγραμματιστές. Τέτοιες αλλαγές δεν θα γίνουν άσκοπα – θα συμβούν μόνο εάν αποκαλυφθούν σοβαρά θεμελιώδη ελαττώματα που παραλείφθηκαν πριν από τη συμπερίληψη του API.

Ακόμη και για provisional API, οι μη backwards συμβατές αλλαγές θεωρούνται «λύση έσχατης ανάγκης»- θα εξακολουθεί να γίνεται κάθε προσπάθεια για να βρεθεί μια λύση backwards συμβατή σε τυχόν εντοπισμένα προβλήματα.

Αυτή η διαδικασία επιτρέπει στην τυπική βιβλιοθήκη να συνεχίσει να εξελίσσεται με την πάροδο του χρόνου, χωρίς να κλειδώνει προβληματικά σφάλματα σχεδιασμού για εκτεταμένες χρονικές περιόδους. Βλ. **PEP 411** για περισσότερες λεπτομέρειες.

provisional πακέτο

Βλ. *provisional API*.

Python 3000

Ψευδώνυμο για το σύνολο εκδόσεων Python 3.x (επινοήθηκε πριν από πολύ καιρό όταν η κυκλοφορία της έκδοσης 3 ήταν κάτι στο μακρινό μέλλον.) Αυτό ονομάζεται επίσης ως συντομογραφία «Py3k».

Pythonic

Μια ιδέα ή ένα κομμάτι κώδικα που ακολουθεί πιστά τα πιο κοινά ιδιώματα της γλώσσας Python, αντί να υλοποιεί κώδικα χρησιμοποιώντας έννοιες κοινές σε άλλες γλώσσες. Για παράδειγμα, ένα κοινό ιδίωμα στην Python είναι να κάνει μια επανάληψη πάνω από όλα τα στοιχεία ενός iterable χρησιμοποιώντας μια δήλωση `for`. Πολλές άλλες γλώσσες που δεν έχουν αυτόν τον τύπο κατασκευής, έτσι οι άνθρωποι που δεν είναι εξοικειωμένοι με την Python χρησιμοποιούν μερικές φορές έναν αριθμητικό μετρητή:

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

Αντίθετα, μια πιο καθαρή μέθοδος Pythonic:

```
for piece in food:
    print(piece)
```

αναγνωρισμένο όνομα

Ένα όνομα με κουκκίδες που δείχνει τη «διαδρομή» από το καθολικό εύρος ενός module σε μια κλάση, συνάρτηση ή μέθοδο που ορίζεται σε αυτήν την ενότητα, όπως ορίζεται στο [PEP 3155](#). Για συναρτήσεις και κλάσεις ανώτατου επιπέδου, το αναγνωρισμένο όνομα είναι ίδιο με το όνομα του αντικειμένου:

```
>>> class C:
...     class D:
...         def meth(self):
...             pass
...
>>> C.__qualname__
'C'
>>> C.D.__qualname__
'C.D'
>>> C.D.meth.__qualname__
'C.D.meth'
```

Όταν χρησιμοποιείται για αναφορά σε modules, το πλήρως αναγνωρισμένο όνομα σημαίνει ολόκληρο το διακεκομμένο path προς το module, συμπεριλαμβανομένων τυχόν γονικών πακέτων π.χ. `email.mime.text`:

```
>>> import email.mime.text
>>> email.mime.text.__name__
'email.mime.text'
```

πλήθος αναφορές

Ο αριθμός των αναφορών σε ένα αντικείμενο. Όταν το πλήθος αναφορών ενός αντικειμένου πέσει στο μηδέν, κατανέμεται. Μερικά αντικείμενα είναι *immortal* και έχουν πλήθος αναφορών που δεν τροποποιούνται ποτέ και επομένως τα αντικείμενα δεν κατανέμονται ποτέ. Η καταμέτρηση αναφορών γενικά δεν είναι ορατή στον κώδικα της Python, αλλά είναι βασικό στοιχείο της υλοποίησης *CPython*. Οι προγραμματιστές μπορούν να καλέσουν τη συνάρτηση `sys.getrefcount()` για να επιστρέψουν το πλήθος αναφορές για ένα συγκεκριμένο αντικείμενο.

In *CPython*, reference counts are not considered to be stable or well-defined values; the number of references to an object, and how that number is affected by Python code, may be different between versions.

κανονικό πακέτο

Ένα παραδοσιακό *package*, όπως ένας κατάλογος που περιέχει ένα `__init__.py` αρχείο.

Βλ. επίσης *namespace package*.

REPL

Ακρωνύμιο του «read-eval-print loop», άλλη ονομασία για το *interactive* περιβάλλον του διερμηνέα.

__slots__

Μια δήλωση μέσα σε μια κλάση που εξοικονομεί μνήμη δηλώνοντας εκ των προτέρων χώρο για παράδειγμα χαρακτηριστικά και εξαλείφοντας λεξικά στιγμιότυπων. Αν και δημοφιλής, η τεχνική είναι κάπως δύσκολο να γίνει σωστή και προορίζεται καλύτερα για σπάνιες περιπτώσεις όπου υπάρχει μεγάλος αριθμός στιγμιότυπων σε μια εφαρμογή κρίσιμης-μνήμης.

ακολουθία

Ένας *iterable* που υποστηρίζει την αποτελεσματική πρόσβαση στο στοιχείο χρησιμοποιώντας άκερτους δείκτες μέσω της ειδικής μεθόδου `__getitem__()` και ορίζει μια μέθοδο `__len__()` που

επιστρέφει το μήκος της ακολουθίας. Ορισμένοι ενσωματωμένοι τύποι ακολουθιών είναι `list`, `str`, `tuple`, και `bytes`. Σημειώστε ότι το `dict` υποστηρίζει επίσης `__getitem__()` και `__len__()`, αλλά θεωρείται αντιστοίχιση και όχι ακολουθία επειδή οι αναζητήσεις χρησιμοποιούν αυθαίρετα *hashable* κλειδιά παρά ακέραιοι.

Η αφηρημένη βασική κλάση `collections.abc.Sequence` ορίζει μια πολύ πιο πλούσια διεπαφή που ξεπερνά τα απλά `__getitem__()` και `__len__()`, adding `count()`, `index()`, `__contains__()`, και `__reversed__()`. Οι τύποι που υλοποιούν αυτήν την διευρυμένη διεπαφή μπορούν να καταχωρηθούν ρητά χρησιμοποιώντας `register()`. Για περισσότερη τεκμηρίωση σχετικά με τις μεθόδους ακολουθίας γενικά, ανατρέξτε στο *Common Sequence Operations*.

set comprehension

Ένας συμπαγής τρόπος για να επεξεργαστείτε όλα ή μέρος των στοιχείων σε ένα iterable και να επιστραφεί ένα σύνολο με τα αποτελέσματα. `results = {c for c in 'abracadabra' if c not in 'abc'}` δημιουργεί το σύνολο συμβολοσειρών `{'r', 'd'}`. Βλ. *comprehensions*.

μοναδικό dispatch

Μια μορφή dispatch *generic function* όπου η υλοποίηση επιλέγεται με βάση τον τύπο ενός μεμονωμένου ορίσματος.

slice

Ένα αντικείμενο που συνήθως περιέχει ένα τμήμα μιας ακολουθίας *sequence*. Δημιουργείται ένα slice χρησιμοποιώντας τη σημείωση subscript, `[]` με άνω και κάτω τελείες μεταξύ αριθμών όταν δίνονται πολλοί, όπως στο `variable_name[1:3:5]`. Η σημείωση αγκύλης (subscript) χρησιμοποιεί εσωτερικά αντικείμενα slice.

απαρχαιωμένη με ήπιο τρόπο

Ένα απαρχαιωμένο με ήπιο τρόπο API δεν θα πρέπει να χρησιμοποιείται σε νέο κώδικα, αλλά είναι ασφαλές σε ήδη υπάρχοντα κώδικα να το χρησιμοποιεί. Το API παραμένει τεκμηριωμένο και δοκιμασμένο, αλλά δεν θα ενισχυθεί περαιτέρω.

Η κατάργηση με ήπιο τρόπο, σε αντίθεση με την κανονική κατάργηση, δεν σχεδιάζει την κατάργηση του API και δεν θα εκπέμπει ειδοποιήσεις

Δείτε *PEP 387: Soft Deprecation*.

ειδική μέθοδος

Μια μέθοδος που καλείται σιωπηρά από την Python για να εκτελέσει μια συγκεκριμένη λειτουργία σε έναν τύπο, όπως η προσθήκη. Τέτοιες μέθοδοι έχουν ονόματα που ξεκινούν και τελειώνουν με διπλές κάτω παύλες. Οι ειδικές μέθοδοι τεκμηριώνονται στο *specialnames*.

standard library

The collection of *packages*, *modules* and *extension modules* distributed as a part of the official Python interpreter package. The exact membership of the collection may vary based on platform, available system libraries, or other criteria. Documentation can be found at *library-index*.

See also `sys.stdlib_module_names` for a list of all possible standard library module names.

δήλωση

Μια πρόταση είναι μέρος μιας σουίτας (ένα «μπλοκ» κώδικα). Μια πρόταση είναι είτε ένας *expression* είτε μια από πολλές δομές με μια λέξη-κλειδί όπως `if`, `while` ή `for`.

ελεγκτής στατικού τύπου

Ένα εξωτερικό εργαλείο όπου διαβάζει τον Python κώδικα και τον αναλύει, αναζητώντας προβλήματα όπως λανθασμένοι τύποι. Βλ. επίσης *type hints* και το *module typing*.

stdlib

An abbreviation of *standard library*.

strong reference

Στο C API της Python, μια ισχυρή αναφορά είναι μια αναφορά σε ένα αντικείμενο που ανήκει στον κώδικα που περιέχει την αναφορά. Η ισχυρή αναφορά λαμβάνεται καλώντας το `Py_INCREF()` όταν η αναφορά δημιουργείται και απελευθερώνεται με `Py_DECREF()` όταν διαγραφεί η αναφορά.

Η συνάρτηση `Py_NewRef()` μπορεί να χρησιμοποιηθεί για τη δημιουργία ισχυρής αναφοράς σε ένα αντικείμενο. Συνήθως, η συνάρτηση `Py_DECREF()` πρέπει να καλείται στην ισχυρή αναφορά πριν βγει από το εύρος της ισχυρής αναφοράς, για να αποφευχθεί η διαρροή μιας αναφοράς.

Βλ. επίσης *borrowed reference*.

t-string

t-strings

String literals prefixed with `t` or `T` are commonly called «t-strings» which is short for template string literals.

κωδικοποίηση κειμένου

Μια συμβολοσειρά στην Python είναι μια ακολουθία σημείων κώδικα Unicode (στο εύρος U+0000–U+10FFFF). Για να αποθηκεύσετε ή να μεταφέρετε μια συμβολοσειρά, πρέπει να σειριοποιηθεί ως δυαδική ακολουθία.

Η σειριοποίηση μιας συμβολοσειράς σε μια δυαδική ακολουθία είναι γνωστή ως «κωδικοποίηση», και η αναδημιουργία της συμβολοσειράς από την δυαδική ακολουθία είναι γνωστή ως «αποκωδικοποίηση».

Υπάρχει μια ποικιλία διαφορετικής σειριοποίησης κειμένου codecs, οι οποίοι συλλογικά αναφέρονται ως «κωδικοποιήσεις κειμένου».

αρχείο κειμένου

Ένα *file object* ικανό να διαβάζει και να γράφει αντικείμενα `str`. Συχνά, ένα αρχείο κειμένου αποκτά πραγματικά πρόσβαση σε μια ροή δυαδική ροή δεδομένων και χειρίζεται αυτόματα την *text encoding*. Παραδείγματα αρχείων κειμένου είναι αρχεία που ανοίγουν σε λειτουργία κειμένου (`'r'` ή `'w'`), `sys.stdin`, `sys.stdout`, και στιγμιότυπα του `io.StringIO`.

Βλ. επίσης *binary file* για ένα αντικείμενο αρχείου με δυνατότητα ανάγνωσης και εγγραφής *δυαδικά αντικείμενα*.

κατάσταση νήματος

Οι πληροφορίες που χρησιμοποιούνται από τη ροή εκτέλεσης της *CPython* για την εκτέλεση σε ένα νήμα λειτουργικού συστήματος. Για παράδειγμα, αυτό περιλαμβάνει την τρέχουσα εξαίρεση, εάν υπάρχει, και την κατάσταση του διερμηνέα bytecode.

Κάθε κατάσταση νήματος είναι συνδεδεμένη με ένα μόνο νήμα λειτουργικού συστήματος, αλλά τα νήματα μπορεί να έχουν πολλές διαθέσιμες καταστάσεις νήματος. Το πολύ, μία από αυτές μπορεί να είναι *attached* ταυτόχρονα.

Απαιτείται ένα *attached thread state* για την κλήση του μεγαλύτερου μέρους του C API της Python, εκτός εάν μια συνάρτηση τεκμηριώνεται ρητά από το αντίθετο. Ο διερμηνέας bytecode εκτελείται μόνο υπό κατάσταση συνημμένου νήματος.

Κάθε κατάσταση νήματος ανήκει σε έναν μόνο διερμηνέα, αλλά κάθε διερμηνέα μπορεί να έχει πολλές καταστάσεις νήματος, συμπεριλαμβανομένων πολλαπλών για το ίδιο νήμα λειτουργικού συστήματος. Οι καταστάσεις νήματος από πολλαπλούς διερμηνείς μπορεί να είναι συνδεδεμένες με το ίδιο νήμα, αλλά μόνο μία μπορεί να είναι *attached* σε αυτό το νήμα σε οποιαδήποτε δεδομένη στιγμή.

Δείτε το Thread State and the Global Interpreter Lock για περισσότερες πληροφορίες.

λεκτικό σύμβολο (token)

Μια μικρή μονάδα πηγαιού κώδικα, που παράγεται από τον lexical analyzer (γνωστό και ως *αναλυτή (tokenizer)*). Ονόματα, αριθμοί, συμβολοσειρές, τελεστές αλλαγές γραμμής και παρόμοια στοιχεία αναπαρίστανται ως λεκτικά σύμβολα (tokens).

Το module `tokenize` εκθέτει τον λεξικό αναλυτή της Python. Το module `token` περιέχει πληροφορίες για τους διάφορους τύπους λεκτικών συμβόλων (tokens).

συμβολοσειρά τριπλών εισαγωγικών

Μια συμβολοσειρά που δεσμεύεται από τρεις περιπτώσεις είτε ενός εισαγωγικού (`»`) ή μιας αποστρόφου (`“`). Αν και δεν παρέχουν καμία λειτουργικότητα που δεν είναι διαθέσιμη με συμβολοσειρές με μονά εισαγωγικά, είναι χρήσιμες για διάφορους λόγους. Σας επιτρέπουν να συμπεριλάβετε μονά και διπλά εισαγωγικά χωρίς διαφυγή σε μια συμβολοσειρά και μπορούν να εκτείνονται σε πολλές γραμμές χωρίς τη χρήση του χαρακτήρα συνέχεια, καθιστώντας τα ιδιαίτερα χρήσιμα κατά τη σύνταξη εγγράφων με συμβολοσειρές.

τύπος

Ο τύπος ενός Python αντικειμένου καθορίζει τι είδους αντικείμενο είναι. Κάθε αντικείμενο έχει έναν τύπο. Ο τύπος ενός αντικειμένου είναι προσβάσιμος ως το χαρακτηριστικό `__class__` ή μπορεί να ανακτηθεί με `type(obj)`.

type alias

Ένα συνώνυμο για έναν τύπο, που δημιουργείται με την ανάθεση τύπου σε ένα αναγνωριστικό.

Τα type aliases είναι χρήσιμα για την απλοποίηση *type alias*. Για παράδειγμα:

```
def remove_gray_shades(
    colors: list[tuple[int, int, int]]) -> list[tuple[int, int,
    int]]:
    pass
```

μπορεί να γίνει πιο ευανάγνωστο όπως:

```
Color = tuple[int, int, int]

def remove_gray_shades(colors: list[Color]) -> list[Color]:
    pass
```

Βλ. `typing` και **PEP 484**, που περιγράφει αυτήν την λειτουργικότητα.

type hint

Ένας *annotation* που καθορίζει τον αναμενόμενο τύπο για μια μεταβλητή, ένα χαρακτηριστικό κλάσης ή μια παράμετρο συνάρτησης ή τιμή επιστροφής.

Οι υποδείξεις τύπων (type hints) είναι προαιρετικές και δεν επιβάλλονται από την Python, αλλά είναι χρήσιμες για *static type checkers*. Μπορούν επίσης να βοηθήσουν τους IDEs με τη συμπλήρωση και την αναδιαμόρφωση κώδικα.

Υποδείξεις τύπου (type hints) για καθολικές μεταβλητές, χαρακτηριστικά κλάσης και συναρτήσεις, αλλά όχι τοπικές μεταβλητές, μπορούν να προσπελαστούν χρησιμοποιώντας το `typing.get_type_hints()`.

Βλ. `typing` και **PEP 484**, που περιγράφει αυτήν την λειτουργικότητα.

καθολικές νέες γραμμές

Ένα τρόπος ερμηνείας ροών κειμένου στον οποίο όλα τα ακόλουθα αναγνωρίζονται ως λήξεις μιας γραμμής: η σύμβαση τέλους γραμμής του Unix `'\n'`, η σύμβαση των Windows `'\r\n'`, και την παλιά σύμβαση Macintosh `'\r'`. Βλ. **PEP 278** και **PEP 3116**, καθώς και `bytes.splitlines()` για πρόσθετη χρήση.

annotation μεταβλητής

Ένας *annotation* μια μεταβλητής ή ενός χαρακτηριστικού κλάσης.

Όταν annotating μια μεταβλητή ή ένα χαρακτηριστικό κλάσης, η ανάθεση είναι προαιρετική:

```
class C:
    field: 'annotation'
```

Τα annotations μεταβλητών χρησιμοποιούνται συνήθως για *type hints*: για παράδειγμα αυτή η μεταβλητή αναμένεται να λάβει τιμές `int`:

```
count: int = 0
```

Η σύνταξη annotation μεταβλητής περιγράφεται στην ενότητα `annassign`.

Βλ. *function annotation*, **PEP 484** και **PEP 526**, που περιγράφουν αυτή τη λειτουργία. Δείτε επίσης `annotations-howto` για βέλτιστες πρακτικές σχετικά με την εργασία με σχολιασμούς.

virtual environment

Ένα συνεργατικά απομονωμένο περιβάλλον χρόνου εκτέλεσης που επιτρέπει στους χρήστες και τις

εφαρμογές της Python να εγκαταστήσουν και να αναβαθμίσουν πακέτα διανομής Python χωρίς να παρεμβαίνουν στη συμπεριφορά άλλων εφαρμογών Python που εκτελούνται στο ίδιο σύστημα.

Βλ. επίσης `venv`.

virtual machine

Ένας υπολογιστής ορίζεται εξ ολοκλήρου από το λογισμικό. Η εικονική μηχανή της Python εκτελεί το *bytecode* που εκπέμπεται από τον μεταγλωττιστή `bytecode`.

walrus operator

A light-hearted way to refer to the assignment expression operator `:` because it looks a bit like a walrus if you turn your head.

Zen της Python

Κατάλογος σχεδιαστικών αρχών και φιλοσοφιών που είναι χρήσιμες για την κατανόηση και τη χρήση της γλώσσας. Ο κατάλογος μπορεί να βρεθεί πληκτρολογώντας «`import this`» στην διαδραστική κονσόλα.

Σχετικά με την τεκμηρίωση

Η τεκμηρίωση της Python έχει δημιουργηθεί από τα [reStructuredText](#) sources του [Sphinx](#), έναν επεξεργαστή εγγράφων που έχει δημιουργηθεί ειδικά για τα έγγραφα της Python.

Η ανάπτυξη των εγγράφων και των εργαλείων τους είναι εξ΄ ολοκλήρου εθελοντική προσπάθεια, όπως και η ίδια η Python. Εάν θέλετε να συνεισφέρετε, ρίξτε μια ματιά στη σελίδα [reporting-bugs](#) για πληροφορίες σχετικές με το πως να το κάνετε. Καινούριοι εθελοντές είναι πάντα ευπρόσδεκτοι!

Πολλές ευχαριστίες πηγαίνουν στους:

- Fred L. Drake, Jr., τον δημιουργό των αρχικών εργαλείων της τεκμηρίωσης της Python και συντάκτη αρκετού περιεχομένου·
- το [Docutils](#) πρότζεκτ για την δημιουργία των εφαρμογών reStructuredText και Docutils·
- Fredrik Lundh για το δικό του Alternative Python Reference πρότζεκτ από το οποίο το Sphinx πήρε πολύ καλές ιδέες.

Β΄.1 Συντελεστές στη τεκμηρίωση της Python

Πολλοί άνθρωποι έχουν συνεισφέρει στη γλώσσα Python, την βιβλιοθήκη της Python, και τα έγγραφα της Python. Δείτε [Misc/ACKS](#) στις πηγές διανομής της Python για μια λίστα των συντελεστών.

Μόνο με τη συμβολή και τις συνεισφορές της κοινότητας της Python, η Python έχει τέτοια υπέροχα έγγραφα – Σας ευχαριστούμε!

Γ'.1 Η ιστορία του λογισμικού

Η Python δημιουργήθηκε στις αρχές του 1990 από τον Guido van Rossum στο Stichting Mathematisch Centrum (CWI, βλ. <https://www.cwi.nl>) στην Ολλανδία ως διάδοχος μια γλώσσας που ονομάζεται ABC. Ο Guido παραμένει ο κύριος συγγραφέας της Python, παρόλα αυτά περιλαμβάνει συνεισφορές και από άλλα άτομα.

Το 1995, ο Guido συνέχισε το έργο του για την Python στο Corporation for National Research Initiatives (CNRI, βλ. <https://www.cnri.reston.va.us>) στο Reston της Virginia, όπου κυκλοφόρησε πολλές εκδόσεις του λογισμικού.

Τον Μάιο του 2000, ο Guido και η βασική ομάδα ανάπτυξης της Python μετακόμισαν στο BeOpen.com για να σχηματίσουν την ομάδα BeOpen PythonLabs. Τον Οκτώβριο του ίδιου έτους, η ομάδα του PythonLabs μετακόμισε στην Digital Creations, η οποία μετατράπηκε σε Zope Corporation. Το 2001, το Python Software Foundation (PSF, βλ. <https://www.python.org/psf>) δημιουργήθηκε ως ένας μη κερδοσκοπικός οργανισμός με στόχο να κατέχει την πνευματική ιδιοκτησία που σχετίζεται με την Python. Η Zope Corporation ήταν μέλος-χορηγός του PSF.

Όλες οι εκδόσεις της Python είναι Ανοιχτού Κώδικα (βλ. <https://opensource.org> για τον ορισμό του Ανοιχτού Κώδικα). Ιστορικά οι περισσότερες, αλλά όχι όλες, εκδόσεις της Python ήταν επίσης συμβατές με την άδεια GPL: ο παρακάτω πίνακας συνοψίζει τις διάφορες εκδόσεις.

Έκδοση	Προερχόμενη από	Έτος	Ιδιοκτησία	Συμβατότητα GPL; (1)
0.9.0 έως 1.2	δ/υ	1991-1995	CWI	ναι
1.3 έως 1.5.2	1.2	1995-1999	CNRI	ναι
1.6	1.5.2	2000	CNRI	όχι
2.0	1.6	2000	BeOpen.com	όχι
1.6.1	1.6	2001	CNRI	ναι (2)
2.1	2.0+1.6.1	2001	PSF	όχι
2.0.1	2.0+1.6.1	2001	PSF	ναι
2.1.1	2.1+2.0.1	2001	PSF	ναι
2.1.2	2.1.1	2002	PSF	ναι
2.1.3	2.1.2	2002	PSF	ναι
2.2 και πάνω	2.1.1	2001-σήμερα	PSF	ναι

i Σημείωση

- (1) Η συμβατότητα με GPL δεν σημαίνει ότι διανέμεται η Python κάτω από την άδεια GPL. Όλες οι άδειες της Python, σε αντίθεση με την GPL, σας επιτρέπουν να διανείμετε μια τροποποιημένη έκδοση χωρίς να κάνετε τις αλλαγές σας, ανοιχτού κώδικα. Οι άδειες με συμβατότητα GPL καθιστούν δυνατό τον συνδυασμό της Python με άλλο λογισμικό που κυκλοφορεί με βάση της GPL, ενώ οι άλλες όχι.
- (2) Σύμφωνα με τον Richard Stallman, 1.6.1 δεν είναι συμβατή με την GPL, επειδή η άδεια της έχει νομική ρήτρα επιλογής, Σύμφωνα με το CNRI, ωστόσο, ο δικηγόρος του Stallman είπε στον δικηγόρο της CNRI ότι η 1.6.1 «δεν είναι συμβατή» με την GPL.

Χάρη, στους πολλούς εξωτερικούς εθελοντές που εργάστηκαν κάτω από τις οδηγίες του Guido, αυτές οι εκδόσεις έγιναν εφικτές.

Γ'.2 Όροι και προϋποθέσεις για την πρόσβαση ή την χρήση της Python με άλλους τρόπους

Το λογισμικό της Python και η τεκμηρίωση αδειοδοτούνται σύμφωνα με την άδεια χρήσης Python Software Foundation Έκδοση 2.

Ξεκινώντας από την Python 3.8.6, παραδείγματα, συνταγές και ο άλλος κώδικας στην τεκμηρίωση έχουν διπλή άδεια χρήσης, σύμφωνα με την Άδεια PSF Έκδοση 2 και της *Zero-Clause BSD άδεια*.

Κάποιο λογισμικό που είναι ενσωματωμένο στην Python είναι υπό διαφορετικές άδειες χρήσης. Οι άδειες παρατίθενται με κώδικα που εμπίπτει σε αυτήν την άδεια. Δείτε *Άδειες και Ευχαριστίες για Ενσωματωμένο Λογισμικό* για μια ελλιπή λίστα αυτών των αδειών.

Γ'.2.1 PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") 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 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001 Python Software Foundation; All Rights Reserved" are retained in Python 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 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.

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

→to Python.

4. PSF is making Python 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 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON
FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A
→RESULT OF
MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, 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, Licensee agrees
to be bound by the terms and conditions of this License Agreement.

Γ'.2.2 ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ BEOPEN.COM ΓΙΑ PYTHON 2.0

ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ ΑΝΟΙΧΤΟΥ ΚΩΔΙΚΑ BEOPEN PYTHON ΕΚΔΟΣΗ 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

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

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.

Γ'.2.3 ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ CNRI ΓΙΑ 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

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

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

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

→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.

Γ'.2.4 ΣΥΜΦΩΝΙΑ ΑΔΕΙΑΣ CWI ΓΙΑ PYTHON 0.9.0 ΕΩΣ 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

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```

→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.

```

Γ'.2.5 ZERO-CLAUSE BSD ΑΔΕΙΑ ΓΙΑ ΤΟΝ ΚΩΔΙΚΑ ΣΤΗΝ ΤΕΚΜΗΡΙΩΣΗ ΤΗΣ PYTHON

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted.

```

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
→WITH
REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL,
→DIRECT,
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.

```

Γ'.3 Άδειες και Ευχαριστίες για Ενσωματωμένο Λογισμικό

Αυτή η ενότητα είναι μια ημιτελής, αλλά αυξανόμενη λίστα αδειών και ευχαριστιών για λογισμικό τρίτων, που ενσωματώνεται στην διανομή της Python.

Γ'.3.1 Mersenne Twister

Η επέκταση `_random` C που βρίσκεται κάτω από την ενότητα `random` περιλαμβάνει έναν κώδικα με βάση μια λήψη από <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html>. Ακολουθούν κατά λέξη τα σχόλια από το αρχικό κώδικα:

```

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.

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

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.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>

email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)

Γ'.3.2 Sockets

Η ενότητα socket χρησιμοποιεί τις συναρτήσεις, `getaddrinfo()`, και `getnameinfo()`, τα οποία έχουν υλοποιηθεί σε διαφορετικά αρχεία από το WIDE Έργο, <https://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 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 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.

Γ'.3.3 Ασύγχρονες socket υπηρεσίες

Οι ενότητες `test.support.asyncio` και `test.support.asyncore` περιέχουν την παρακάτω ειδοποίηση:

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.

Γ'.3.4 Διαχείριση Cookie

Η ενότητα `http.cookies` περιέχει την παρακάτω ειδοποίηση:

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.

Γ'.3.5 Ανίχνευση εκτέλεσης

Η ενότητα `trace` περιέχει την παρακάτω ειδοποίηση:

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

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```
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.
```

Γ'.3.6 Συναρτήσεις UUencode και UUdecode

Ο `uu` κωδικοποιητής περιέχει την παρακάτω ειδοποίηση:

```
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
```

Γ'.3.7 Κλήσεις Απομακρυσμένης Διαδικασίας XML

Η ενότητα `xmlrpc.client` περιέχει την παρακάτω ειδοποίηση:

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.

Γ'.3.8 `test_epoll`

Η ενότητα `test.test_epoll` περιέχει την παρακάτω ειδοποίηση:

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.

Γ'.3.9 Επιλογή kqueue

Η ενότητα select περιέχει την παρακάτω ειδοποίηση για την kqueue διεπαφή:

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.

Γ'.3.10 SipHash24

Το αρχείο Python/pyhash.c περιέχει την υλοποίηση του Marek Majkowski του αλγορίθμου του Dan Bernstein, SipHash24. Αυτό περιέχει την παρακάτω σημείωση:

<MIT License>
Copyright (c) 2013 Marek Majkowski <marek@popcount.org>

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.
</MIT License>

Original location:
<https://github.com/majek/csiphash/>

Solution inspired by code from:
Samuel Neves (supercop/crypto_auth/siphhash24/little)
djb (supercop/crypto_auth/siphhash24/little2)
Jean-Philippe Aumasson (<https://131002.net/siphhash/siphhash24.c>)

Γ'.3.11 strtod και dtoa

Το αρχείο `Python/dtoa.c`, που παρέχει τις συναρτήσεις `dtoa` και `strtod` της C για μετατροπή των C doubles προς και από strings, προέρχεται από το ομώνυμο αρχείο του David M. Gay, προς το παρόν διαθέσιμο από <https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c>. Το αρχικό αρχείο, όπως ανακτήθηκε στις 16 Μαρτίου, 2009, περιέχει τα ακόλουθα πνευματικά δικαιώματα και την ειδοποίηση αδειοδότησης:

```

/*****
 *
 * The author of this software is David M. Gay.
 *
 * Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose without fee is hereby granted, provided that this entire notice
 * is included in all copies of any software which is or includes a copy
 * or modification of this software and in all copies of the supporting
 * documentation for such software.
 *
 * THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
 * REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY
 * OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
 *
 *****/

```

Γ'.3.12 OpenSSL

Οι μονάδες `hashlib`, `posix` και `ssl` χρησιμοποιούν την βιβλιοθήκη OpenSSL για επιπλέον απόδοση, εάν διατίθενται από το λειτουργικό σύστημα. Επιπλέον, τα προγράμματα εγκατάστασης για την Python για Windows και macOS, ενδέχεται να περιλαμβάνουν ένα αντίγραφο των βιβλιοθηκών OpenSSL, επομένως συμπεριλαμβάνουμε ένα αντίγραφο της άδειας OpenSSL εδώ. Για την έκδοση OpenSSL 3.0 και για νεότερες εκδόσεις που προέρχονται από αυτή, ισχύει η άδεια Apache v2:

```

                                Apache License
                                Version 2.0, January 2004
                                https://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction,
and distribution as defined by Sections 1 through 9 of this document.

"Licenser" shall mean the copyright owner or entity authorized by
the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all
other entities that control, are controlled by, or are under common
control with that entity. For the purposes of this definition,
"control" means (i) the power, direct or indirect, to cause the
direction or management of such entity, whether by contract or
otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity

```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Γ'.3.13 expat

Η επέκταση pyexpat δημιουργείται χρησιμοποιώντας ένα συμπεριλαμβανόμενο αντίγραφο των πηγών expat, εκτός εάν η έκδοση έχει την ρύθμιση --with-system-expat:

Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd
and Clark Cooper

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

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

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.

Γ'.3.14 libffi

Η επέκταση της C `_ctypes` που βρίσκεται κάτω από την ενότητα `ctypes` δημιουργείται χρησιμοποιώντας ένα συμπεριλαμβανόμενο αντίγραφο των πηγών `libffi`, εκτός εάν η έκδοση έχει την ρύθμιση `--with-system-libffi`:

Copyright (c) 1996–2008 Red Hat, Inc and others.

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.

Γ'.3.15 zlib

Η επέκταση `zlib` δημιουργείται χρησιμοποιώντας ένα συμπεριλαμβανόμενου αντίγραφο των πηγών `zlib`, εάν η έκδοση του `zlib` που βρίσκεται στο σύστημα είναι πολύ παλιά για να χρησιμοποιηθεί για την κατασκευή:

Copyright (C) 1995–2011 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
 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.
- Jean-loup Gailly Mark Adler
jloup@gzip.org madler@alumni.caltech.edu

Γ'.3.16 cfuhash

Η υλοποίηση του πίνακα κατακερματισμού που χρησιμοποιείται από το `tracemalloc` βασίζεται στο έργο `cfuhash`:

Copyright (c) 2005 Don Owens
All rights reserved.

This code is released under the BSD license:

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

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * 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.
- * Neither the name of the author 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 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.

Γ'.3.17 libmpdec

Η επέκταση `_decimal` που βρίσκεται κάτω από την ενότητα `decimal` είναι φτιαγμένη χρησιμοποιώντας ένα συμπεριλαμβανόμενο αντίγραφο της βιβλιοθήκης `libmpdec`, εκτός αν η έκδοση έχει ρύθμιση `--with-system-libmpdec`:

Copyright (c) 2008–2020 Stefan Krah. 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.

Γ'.3.18 W3C C14N σουίτα δοκιμής

Η σουίτα δοκιμής C14N 2.0 στο πακέτο `test` (`Lib/test/xmltestdata/c14n-20/`) ανακτήθηκε από τον ιστότοπο του W3C <https://www.w3.org/TR/xml-c14n2-testcases/> και διανέμεται με την άδεια 3 ρήτρων BSD:

Copyright (c) 2013 W3C(R) (MIT, ERCIM, Keio, Beihang), All Rights Reserved.

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

- * Redistributions of works must retain the original copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the original copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the W3C nor the names of its contributors may be used to endorse or promote products derived from this work 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,

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

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.

Γ'.3.19 mimalloc

MIT Άδεια:

Copyright (c) 2018–2021 Microsoft Corporation, Daan Leijen

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.

Γ'.3.20 asyncio

Μέρη της ενότητας asyncio ενσωματώνονται από το [uvloop 0.16](#), η οποία διανέμεται με άδεια MIT:

Copyright (c) 2015–2021 MagicStack Inc. <http://magic.io>

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

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

```

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.

```

Γ'.3.21 Καθολικές Απεριόριστες Ακολουθίες (KAA)

Το αρχείο `Python/qsbr.c` είναι προσαρμοσμένο από το σύστημα ασφαλούς ανάκτησης μνήμης «Global Unbounded Sequences» του FreeBSD, που υλοποιείται στο `subr_smr.c`. Το αρχείο διανέμεται υπό την Άδεια 2-Clause BSD:

```

Copyright (c) 2019,2020 Jeffrey Roberson <jeff@FreeBSD.org>

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 unmodified, 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 "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 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.

```

Γ'.3.22 Δεσμεύσεις Zstandard

Οι δεσμεύσεις Zstandard στα `Modules/_zstd` και `Lib/compression/zstd` βασίζονται σε κώδικα από τη βιβλιοθήκη `pyzstd library`, πνευματικής ιδιοκτησίας του Ma Lin και των συνεργατών του. Ο κώδικας της `pyzstd` διανέμεται υπό την άδεια 3-Clause BSD.

```

Copyright (c) 2020-present, Ma Lin and contributors.
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.

```

(συνέχεια στην επόμενη σελίδα)

(συνεχίζεται από την προηγούμενη σελίδα)

3. Neither the name of the copyright holder 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 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 HOLDER 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

Η Python και αυτή η τεκμηρίωση είναι:

Copyright © 2001 Python Software Foundation. All rights reserved.

Copyright © 2000 BeOpen.com. Όλα τα δικαιώματα διατηρούνται.

Copyright © 1995-2000 Corporation for National Research Initiatives. Όλα τα δικαιώματα διατηρούνται.

Copyright © 1991-1995 Stichting Mathematisch Centrum. Όλα τα δικαιώματα διατηρούνται.

Ανατρέξτε στο *[Ιστορία και Άδεια](#)* για πλήρης πληροφόρηση σχετικά με την άδεια χρήσης και τις εξουσιοδοτήσεις.

μη-αλφαβητικά

..., [93](#)

-?

command line option, [5](#)

>>>, [93](#)

-B

command line option, [6](#)

BDFL, [95](#)

BOLT_APPLY_FLAGS

command line option, [31](#)

BOLT_INSTRUMENT_FLAGS

command line option, [31](#)

BZIP2_CFLAGS

command line option, [28](#)

BZIP2_LIBS

command line option, [28](#)

CC

command line option, [27](#)

CFLAGS, [30](#), [4042](#)

command line option, [27](#)

CFLAGS_NODIST, [4042](#)

CONFIG_SITE

command line option, [37](#)

CPP

command line option, [27](#)

CPPFLAGS, [40](#), [42](#)

command line option, [27](#)

CPython, [97](#)

CURSES_CFLAGS

command line option, [28](#)

CURSES_LIBS

command line option, [28](#)

C-contiguous, [97](#)

-E

command line option, [6](#)

EAFP, [99](#)

Fortran contiguous, [97](#)

GDBM_CFLAGS

command line option, [28](#)

GDBM_LIBS

command line option, [28](#)

GIL, [101](#)

HOSTRUNNER

command line option, [37](#)

-I

command line option, [6](#)

IDLE, [102](#)

LBYL, [104](#)

LDFLAGS, [40](#), [42](#)

command line option, [28](#)

LDFLAGS_NODIST, [42](#)

LIBB2_CFLAGS

command line option, [28](#)

LIBB2_LIBS

command line option, [28](#)

LIBEDIT_CFLAGS

command line option, [28](#)

LIBEDIT_LIBS

command line option, [28](#)

LIBFFI_CFLAGS

command line option, [28](#)

LIBFFI_LIBS

command line option, [28](#)

LIBLZMA_CFLAGS

command line option, [28](#)

LIBLZMA_LIBS

command line option, [28](#)

LIBMPDEC_CFLAGS

command line option, [28](#)

LIBMPDEC_LIBS

command line option, [28](#)

LIBREADLINE_CFLAGS

command line option, [28](#)

LIBREADLINE_LIBS

command line option, [28](#)

LIBS

command line option, [28](#)

LIBSQLITE3_CFLAGS

command line option, [29](#)

LIBSQLITE3_LIBS

command line option, [29](#)

LIBUUID_CFLAGS

command line option, [29](#)

LIBUUID_LIBS

command line option, [29](#)

LIBZSTD_CFLAGS

command line option, [29](#)

LIBZSTD_LIBS
 command line option, 29
 MACHDEP
 command line option, 28
 MRO, **105**
 -O
 command line option, 7
 -OO
 command line option, 7
 OPT, **33**
 -P
 command line option, 7
 PANEL_CFLAGS
 command line option, 29
 PANEL_LIBS
 command line option, 29
 PATH, **11, 21, 44, 48, 49, 51, 52, 5860, 63, 65**
 PATHEXT, **60**
 PEP, **108**
 PKG_CONFIG
 command line option, 27
 PKG_CONFIG_LIBDIR
 command line option, 27
 PKG_CONFIG_PATH
 command line option, 27
 PROFILE_TASK, **30**
 PYLAUNCHER_ALLOW_INSTALL, **67**
 PYLAUNCHER_ALWAYS_INSTALL, **67**
 PYLAUNCHER_DEBUG, **67**
 PYLAUNCHER_DRYRUN, **67**
 PYLAUNCHER_NO_SEARCH_PATH, **65**
 PYTHONCOERCECLOCALE, **25**
 PYTHONDEBUG, **6, 32**
 PYTHONDEVMODE, **9**
 PYTHONDONTWRITEBYTECODE, **6**
 PYTHONDUMPPREFS, **33**
 PYTHONFAULTHANDLER, **9**
 PYTHONHASHSEED, **7, 13**
 PYTHONHOME, **6, 11, 56, 89**
 PYTHONINSPECT, **6**
 PYTHONINTMAXSTRDIGITS, **9**
 PYTHONIOENCODING, **15**
 PYTHONLEGACYWINDOWSSTDIO, **13**
 PYTHONMALLOC, **15, 31**
 PYTHONNODEBUGRANGES, **10**
 PYTHONNOUSERSITE, **7**
 PYTHONOPTIMIZE, **7**
 PYTHONPATH, **6, 11, 56, 89, 90**
 PYTHONPERFSUPPORT, **10**
 PYTHONPROFILEIMPORTTIME, **9**
 PYTHONPYCACHEPREFIX, **9**
 PYTHONSAFEPATH, **7**
 PYTHONSTARTUP, **6, 12**
 PYTHONTRACEMALLOC, **9**
 PYTHONUNBUFFERED, **8**
 PYTHONUTF8, **9, 16, 55**
 PYTHONVERBOSE, **8**
 PYTHONWARNDEFAULTENCODING, **9**
 PYTHONWARNINGS, **9**
 PYTHON_COLORS, **11**
 PYTHON_CONTEXT_AWARE_WARNINGS, **11**
 PYTHON_CPU_COUNT, **10**
 PYTHON_DISABLE_REMOTE_DEBUG, **10**
 PYTHON_FROZEN_MODULES, **10**
 PYTHON_GIL, **10, 101**
 PYTHON_JIT, **27**
 PYTHON_MANAGER_DEFAULT, **44**
 PYTHON_PERF_JIT_SUPPORT, **10**
 PYTHON_PRESITE, **10**
 PYTHON_THREAD_INHERIT_CONTEXT, **10**
 PYTHON_TLBC, **11**
 PY_PYTHON, **66**
 Py_REMOTE_DEBUG (*C macro*), **32**
 Python 3000, **108**
 Python Enhancement Proposals
 PEP 1, **108**
 PEP 7, **23**
 PEP 8, **91**
 PEP 11, **23, 55**
 PEP 238, **100**
 PEP 278, **112**
 PEP 302, **104**
 PEP 338, **4**
 PEP 343, **97**
 PEP 362, **94, 107**
 PEP 370, **8, 13**
 PEP 397, **63**
 PEP 411, **108**
 PEP 420, **106, 108**
 PEP 443, **101**
 PEP 483, **101**
 PEP 484, **93, 100, 101, 112**
 PEP 488, **7**
 PEP 492, **94, 95, 97**
 PEP 498, **99**
 PEP 514, **63**
 PEP 519, **108**
 PEP 525, **94**
 PEP 526, **93, 112**
 PEP 528, **56**
 PEP 529, **15, 56**
 PEP 538, **16, 25**
 PEP 585, **101**
 PEP 649, **93**
 PEP 683, **102**
 PEP 703, **62, 78, 100, 102**
 PEP 768, **10, 16, 32**
 PEP 3116, **112**
 PEP 3155, **109**
 Pythonic, **108**
 -R
 command line option, 7
 REPL, **109**
 -S
 command line option, 8
 TCLTK_CFLAGS

command line option, 29
 TCLTK_LIBS
 command line option, 29
 -V
 command line option, 6
 -W
 command line option, 8
 -X
 command line option, 9
 ZLIB_CFLAGS
 command line option, 29
 ZLIB_LIBS
 command line option, 29
 Zen της Python, 113
 __future__, 100
 __slots__, 109
 annotation, 93
 annotation μεταβλητής, 112
 awaitable, 95
 -b
 command line option, 6
 --build
 command line option, 37
 bytecode, 96
 bytes-like αντικείμενα, 95
 -c
 command line option, 4
 callable, 96
 callback, 96
 --check-hash-based-pycs
 command line option, 6
 command line option
 -?, 5
 -B, 6
 BOLT_APPLY_FLAGS, 31
 BOLT_INSTRUMENT_FLAGS, 31
 BZIP2_CFLAGS, 28
 BZIP2_LIBS, 28
 CC, 27
 CFLAGS, 27
 CONFIG_SITE, 37
 CPP, 27
 CPPFLAGS, 27
 CURSES_CFLAGS, 28
 CURSES_LIBS, 28
 -E, 6
 GDBM_CFLAGS, 28
 GDBM_LIBS, 28
 HOSTRUNNER, 37
 -I, 6
 LDFLAGS, 28
 LIBB2_CFLAGS, 28
 LIBB2_LIBS, 28
 LIBEDIT_CFLAGS, 28
 LIBEDIT_LIBS, 28
 LIBFFI_CFLAGS, 28
 LIBFFI_LIBS, 28
 LIBLZMA_CFLAGS, 28
 LIBLZMA_LIBS, 28
 LIBMPDEC_CFLAGS, 28
 LIBMPDEC_LIBS, 28
 LIBREADLINE_CFLAGS, 28
 LIBREADLINE_LIBS, 28
 LIBS, 28
 LIBSQLITE3_CFLAGS, 29
 LIBSQLITE3_LIBS, 29
 LIBUUID_CFLAGS, 29
 LIBUUID_LIBS, 29
 LIBZSTD_CFLAGS, 29
 LIBZSTD_LIBS, 29
 MACHDEP, 28
 -O, 7
 -OO, 7
 -P, 7
 PANEL_CFLAGS, 29
 PANEL_LIBS, 29
 PKG_CONFIG, 27
 PKG_CONFIG_LIBDIR, 27
 PKG_CONFIG_PATH, 27
 -R, 7
 -S, 8
 TCLTK_CFLAGS, 29
 TCLTK_LIBS, 29
 -V, 6
 -W, 8
 -X, 9
 ZLIB_CFLAGS, 29
 ZLIB_LIBS, 29
 -b, 6
 --build, 37
 -c, 4
 --check-hash-based-pycs, 6
 -d, 6
 --disable-gil, 27
 --disable-ipv6, 24
 --disable-safety, 35
 --disable-test-modules, 30
 --enable-big-digits, 24
 --enable-bolt, 31
 --enable-experimental-jit, 27
 --enable-framework, 36
 --enable-loadable-sqlite-extensions, 24
 --enable-optimizations, 30
 --enable-profiling, 31
 --enable-pystats, 25
 --enable-shared, 33
 --enable-slower-safety, 35
 --enable-universalsdk, 36
 --enable-wasm-dynamic-linking, 29
 --enable-wasm-pthreads, 29
 --exec-prefix, 29
 -h, 5
 --help, 5
 --help-all, 5
 --help-env, 5

--help-xoptions, 5
--host, 37
-i, 6
-m, 4
--prefix, 29
-q, 7
-s, 7
-u, 8
-v, 8
--version, 6
--with-address-sanitizer, 33
--with-app-store-compliance, 36
--with-assertions, 33
--with-build-python, 37
--with-builtin-hashlib-hashes, 35
--with-computed-gotos, 31
--with-dbmliborder, 25
--with-dtrace, 33
--with-ensurepip, 30
--with-framework-name, 36
--with-hash-algorithm, 35
--with-libc, 34
--with-libm, 34
--with-libs, 34
--with-lto, 30
--with-memory-sanitizer, 33
--with-openssl, 34
--with-openssl-rpath, 34
--without-c-locale-coercion, 25
--without-decimal-contextvar, 25
--without-doc-strings, 31
--without-mimalloc, 31
--without-pymalloc, 31
--without-readline, 34
--without-remote-debug, 31
--without-static-libpython, 33
--with-pkg-config, 25
--with-platlibdir, 25
--with-pydebug, 32
--with-readline, 34
--with-ssl-default-suites, 35
--with-strict-overflow, 31
--with-suffix, 24
--with-system-expat, 34
--with-system-libmpdec, 34
--with-tail-call-interp, 31
--with-thread-sanitizer, 33
--with-trace-refs, 32
--with-tzpath, 25
--with-undefined-behavior-sanitizer, 33
--with-universal-archs, 36
--with-valgrind, 33
--with-wheel-pkg-dir, 25
-x, 9
context, 97
context μεταβλητή, 97
contiguous, 97
coroutine, 97
coroutine συνάρτηση, 97
-d
 command line option, 6
decorator, 98
descriptor, 98
--disable-gil
 command line option, 27
--disable-ipv6
 command line option, 24
--disable-safety
 command line option, 35
--disable-test-modules
 command line option, 30
docstring, 98
duck-typing, 98
dunder, 99
--enable-big-digits
 command line option, 24
--enable-bolt
 command line option, 31
--enable-experimental-jit
 command line option, 27
--enable-framework
 command line option, 36
--enable-loadable-sqlite-extensions
 command line option, 24
--enable-optimizations
 command line option, 30
--enable-profiling
 command line option, 31
--enable-pystats
 command line option, 25
--enable-shared
 command line option, 33
--enable-slower-safety
 command line option, 35
--enable-universalsdk
 command line option, 36
--enable-wasm-dynamic-linking
 command line option, 29
--enable-wasm-pthreads
 command line option, 29
--exec-prefix
 command line option, 29
f-string, 99
f-strings, 99
finder, 100
generator, 101
generator iterator, 101
generator έκφραση, 101
global interpreter lock, 101
-h
 command line option, 5
hash-based pyc, 102
hashable, 102
--help
 command line option, 5

- help-all
 - command line option, 5
- help-env
 - command line option, 5
- help-xoptions
 - command line option, 5
- host
 - command line option, 37
- i
 - command line option, 6
- immutable, 102
- interpreted, 103
- iterable, 103
- iterator, 103
- lambda, 104
- list comprehension, 104
- loader, 104
- m
 - command line option, 4
- magic
 - μέθοδος, 105
- mapping, 105
- meta path finder, 105
- module, 105
- module επέκτασης, 99
- mutable, 105
- named tuple, 105
- namespace, 106
- nested scope, 106
- path based finder, 108
- path entry, 107
- path entry finder, 107
- path entry hook, 108
- path-like αντικείμενο, 108
- prefix
 - command line option, 29
- provisional API, 108
- provisional πακέτο, 108
- q
 - command line option, 7
- s
 - command line option, 7
- set comprehension, 110
- slice, 110
- special
 - μέθοδος, 110
- standard library, 110
- stdlib, 110
- strong reference, 110
- t-string, 111
- t-strings, 111
- type alias, 112
- type hint, 112
- u
 - command line option, 8
- v
 - command line option, 8
- version
 - command line option, 6
- virtual environment, 112
- virtual machine, 113
- walrus operator, 113
- with-address-sanitizer
 - command line option, 33
- with-app-store-compliance
 - command line option, 36
- with-assertions
 - command line option, 33
- with-build-python
 - command line option, 37
- with-builtin-hashlib-hashes
 - command line option, 35
- with-computed-gotos
 - command line option, 31
- with-dbmliborder
 - command line option, 25
- with-dtrace
 - command line option, 33
- with-ensurepip
 - command line option, 30
- with-framework-name
 - command line option, 36
- with-hash-algorithm
 - command line option, 35
- with-libc
 - command line option, 34
- with-libm
 - command line option, 34
- with-libs
 - command line option, 34
- with-lto
 - command line option, 30
- with-memory-sanitizer
 - command line option, 33
- with-openssl
 - command line option, 34
- with-openssl-rpath
 - command line option, 34
- without-c-locale-coercion
 - command line option, 25
- without-decimal-contextvar
 - command line option, 25
- without-doc-strings
 - command line option, 31
- without-mimalloc
 - command line option, 31
- without-pymalloc
 - command line option, 31
- without-readline
 - command line option, 34
- without-remote-debug
 - command line option, 31
- without-static-libpython
 - command line option, 33
- with-pkg-config
 - command line option, 25

--with-platlibdir
 command line option, 25

--with-pydebug
 command line option, 32

--with-readline
 command line option, 34

--with-ssl-default-suites
 command line option, 35

--with-strict-overflow
 command line option, 31

--with-suffix
 command line option, 24

--with-system-expat
 command line option, 34

--with-system-libmpdec
 command line option, 34

--with-tail-call-interp
 command line option, 31

--with-thread-sanitizer
 command line option, 33

--with-trace-refs
 command line option, 32

--with-tzpath
 command line option, 25

--with-undefined-behavior-sanitizer
 command line option, 33

--with-universal-archs
 command line option, 36

--with-valgrind
 command line option, 33

--with-wheel-pkg-dir
 command line option, 25

-x
 command line option, 9

A

Αθάνατο, 102

ακέραια διαίρεση, 100

ακολουθία, 109

αναγνωρισμένο όνομα, 109

αντικείμενο, 106

αντικείμενο αρχείου, 99

αντικείμενο που μοιάζει με αρχείο, 99

αξιολόγηση συνάρτησης, 99

απαρχαιωμένη με ήπιο τρόπο, 110

αρχείο κειμένου, 111

ασύγχρονος generator, 94

ασύγχρονος generator iterator, 94

ασύγχρονος iterable, 94

ασύγχρονος iterator, 94

ασύγχρονος διαχειριστής context, 94

αφηρημένη βασική κλάση, 93

B

βελτιστοποιημένο πεδίο ορατότητας
 (*scope*), 106

Γ

γενική συνάρτηση, 101

γενικός τύπος, 101

Δ

δανεική αναφορά, 95

δήλωση, 110

διαδραστικός, 102

διαχειριστής context, 97

δυαδικό αρχείο, 95

δωρεάν μεταβλητή, 100

δωρεάν νήμα, 100

Ε

ειδική μέθοδος, 110

εισαγόμενο path, 102

εισαγωγέας, 102

εισαγωγή, 102

έκφραση, 99

ελεγκτής στατικού τύπου, 110

Κ

καθολικές νέες γραμμές, 112

κανονικό πακέτο, 109

κατανόηση λεξικού, 98

κατάσταση νήματος, 111

κατάσταση συνδεδεμένου νήματος, 95

κλάση, 96

κλάση νέου στυλ, 106

κυκλική απομόνωση, 97

κωδικοποίηση κειμένου, 111

κωδικοποίηση συστήματος αρχείων και
 χειριστής σφαλμάτων, 99

Λ

λεκτικό σύμβολο (*token*), 111

λεξικό, 98

λεξικός αναλυτής, 104

λίστα, 104

Μ

μαγική μέθοδος, 105

μέθοδος, 105

magic, 105

special, 110

μετα-κλάση, 105

μεταβλητή κλάσης, 96

μεταβλητή κλεισίματος, 96

μεταβλητή περιβάλλοντος

 BASECFLAGS, 41

 BASECPPFLAGS, 40

 BLDSHARED, 42

 CC, 40

 CCSHARED, 41

 CFLAGS, 30, 4042

 CFLAGSFORSHARED, 41

 CFLAGS_ALIASING, 41

CFLAGS_NODIST, 4042
 COMPILEALL_OPTS, 40
 CONFIGURE_CFLAGS, 40
 CONFIGURE_CFLAGS_NODIST, 41
 CONFIGURE_CPPFLAGS, 40
 CONFIGURE_LDFLAGS, 42
 CONFIGURE_LDFLAGS_NODIST, 42
 CPPFLAGS, 40, 42
 CXX, 40
 EXTRA_CFLAGS, 40
 LDFLAGS, 40, 42
 LDFLAGS_NODIST, 42
 LD_SHARED, 42
 LIBS, 42
 LINKCC, 42
 OPT, 33, 41
 PATH, 11, 21, 44, 48, 49, 51, 52, 5860, 63, 65
 PATHEXT, 60
 PROFILE_TASK, 30
 PURIFY, 41
 PYLAUNCHER_ALLOW_INSTALL, 67
 PYLAUNCHER_ALWAYS_INSTALL, 67
 PYLAUNCHER_DEBUG, 67
 PYLAUNCHER_DRYRUN, 67
 PYLAUNCHER_NO_SEARCH_PATH, 65
 PYTHONASYNCODEBUG, 14
 PYTHONBREAKPOINT, 12
 PYTHONCASEOK, 12
 PYTHONCOERCECLOCALE, 15, 25
 PYTHONDEBUG, 6, 12, 32
 PYTHONDEVMODE, 9, 16
 PYTHONDONTWRITEBYTECODE, 6, 12
 PYTHONDUMPREFS, 18, 33
 PYTHONDUMPREFSFILE, 18
 PYTHONEXECUTABLE, 13
 PYTHONFAULTHANDLER, 9, 14
 PYTHONHASHSEED, 7, 13
 PYTHONHOME, 6, 11, 56, 89
 PYTHONINSPECT, 6, 12
 PYTHONINTMAXSTRDIGITS, 9, 13
 PYTHONIOENCODING, 13, 15
 PYTHONLEGACYWINDOWSFSENCODING, 15
 PYTHONLEGACYWINDOWSSTDIO, 13, 15
 PYTHONMALLOC, 14, 15, 31
 PYTHONMALLOCSTATS, 15
 PYTHONNODEBUGRANGES, 10, 16
 PYTHONNOUSERSITE, 7, 13
 PYTHONOPTIMIZE, 7, 12
 PYTHONPATH, 6, 11, 56, 89, 90
 PYTHONPERFSUPPORT, 10, 16
 PYTHONPLATLIBDIR, 12
 PYTHONPROFILEIMPORTTIME, 9, 14
 PYTHONPYCACHEPREFIX, 9, 13
 PYTHONSAFEPATH, 7, 11
 PYTHONSTARTUP, 6, 12
 PYTHONTRACEMALLOC, 9, 14
 PYTHONUNBUFFERED, 8, 12
 PYTHONUSERBASE, 13

PYTHONUTF8, 9, 16, 55
 PYTHONVERBOSE, 8, 12
 PYTHONWARNDEFAULTENCODING, 9, 16
 PYTHONWARNINGS, 9, 13
 PYTHON_BASIC_REPL, 17
 PYTHON_COLORS, 11, 17
 PYTHON_CONTEXT_AWARE_WARNINGS, 11, 17
 PYTHON_CPU_COUNT, 10, 17
 PYTHON_DISABLE_REMOTE_DEBUG, 10, 16
 PYTHON_FROZEN_MODULES, 10, 17
 PYTHON_GIL, 10, 17, 101
 PYTHON_HISTORY, 17
 PYTHON_JIT, 17, 27
 PYTHON_MANAGER_DEFAULT, 44
 PYTHON_PERF_JIT_SUPPORT, 10, 16
 PYTHON_PRESITE, 10, 18
 PYTHON_THREAD_INHERIT_CONTEXT, 10, 17
 PYTHON_TLBC, 11, 18
 PY_BUILTIN_MODULE_CFLAGS, 41
 PY_CFLAGS, 41
 PY_CFLAGS_NODIST, 41
 PY_CORE_CFLAGS, 41
 PY_CORE_LDFLAGS, 42
 PY_CPPFLAGS, 40
 PY_LDFLAGS, 42
 PY_LDFLAGS_NODIST, 42
 PY_PYTHON, 66
 PY_STDMODULE_CFLAGS, 41
 μιγαδικός αριθμός, 96
 μοναδικό dispatch, 110

O

όρισμα, 94
 όρισμα keyword, 104
 όρισμα θέσης, 108
 όψη λεξικού, 98

Π

πακέτο, 107
 πακέτο namespace, 106
 παράμετρος, 107
 πλήθος αναφοράς, 109
 πρωτόκολλο διαχείρισης περιβάλλοντος, 97

Σ

σειρά ανάλυσης μεθόδων, 105
 συλλογή απορριμάτων, 100
 συμβολοσειρά τριπλών εισαγωγικών, 111
 συνάρτηση, 100
 συνάρτηση annotate, 93
 συνάρτηση annotation, 100
 συνάρτηση key, 103

T

τερματισμός λειτουργίας διερμηνέα, 103

τεχνικές προδιαγραφές module, **105**

τμήμα, **108**

τοπική κωδικοποίηση, **104**

τρέχον πλαίσιο, **97**

τύπος, **112**

X

χαρακτηριστικό, **95**