Installing Python Modules

Release 3.12.8

Guido van Rossum and the Python development team

January 17, 2025

Python Software Foundation Email: docs@python.org

CONTENTS

1	Key terms									
2	Basic usage									
3										
4	Common installation issues 4.1 Installing into the system Python on Linux	9 9 9								
A	Glossary	11								
В		27 27								
C	C.1 History of the software C.2 Terms and conditions for accessing or otherwise using Python C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.12.8 C.2.2 BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0 C.2.3 CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1 C.2.4 CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2 C.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.12.8 DOCUMEN-	29 30 30 31 31 32								
	C.3 Licenses and Acknowledgements for Incorporated Software C.3.1 Mersenne Twister C.3.2 Sockets C.3.3 Asynchronous socket services C.3.4 Cookie management C.3.5 Execution tracing C.3.6 UUencode and UUdecode functions C.3.7 XML Remote Procedure Calls C.3.8 test_epoll C.3.9 Select kqueue C.3.10 SipHash24 C.3.11 strtod and dtoa C.3.12 OpenSSL C.3.13 expat C.3.14 libffi	33 34 35 35 36 37 38 39 39 42 43 43								

Inc	dex																	49
D	Copyright																	47
	C.3.20	asyncio .			 													46
	C.3.19	Audioop .			 													46
	C.3.18	W3C C14	N test	suite														45
	C.3.17	libmpdec			 													44
	C.3.16	cfuhash .			 													44

Email

distutils-sig@python.org

As a popular open source development project, Python has an active supporting community of contributors and users that also make their software available for other Python developers to use under open source license terms.

This allows Python users to share and collaborate effectively, benefiting from the solutions others have already created to common (and sometimes even rare!) problems, as well as potentially contributing their own solutions to the common pool.

This guide covers the installation part of the process. For a guide to creating and sharing your own Python projects, refer to the Python packaging user guide.



1 Note

For corporate and other institutional users, be aware that many organisations have their own policies around using and contributing to open source software. Please take such policies into account when making use of the distribution and installation tools provided with Python.

1 **CONTENTS**

2 CONTENTS

CHAPTER

ONE

KEY TERMS

- pip is the preferred installer program. Starting with Python 3.4, it is included by default with the Python binary installers.
- A virtual environment is a semi-isolated Python environment that allows packages to be installed for use by a particular application, rather than being installed system wide.
- venv is the standard tool for creating virtual environments, and has been part of Python since Python 3.3. Starting with Python 3.4, it defaults to installing pip into all created virtual environments.
- virtualenv is a third party alternative (and predecessor) to venv. It allows virtual environments to be used on versions of Python prior to 3.4, which either don't provide venv at all, or aren't able to automatically install pip into created environments.
- The Python Package Index is a public repository of open source licensed packages made available for use by other Python users.
- the Python Packaging Authority is the group of developers and documentation authors responsible for the maintenance and evolution of the standard packaging tools and the associated metadata and file format standards. They maintain a variety of tools, documentation, and issue trackers on GitHub.
- distutils is the original build and distribution system first added to the Python standard library in 1998. While direct use of distutils is being phased out, it still laid the foundation for the current packaging and distribution infrastructure, and it not only remains part of the standard library, but its name lives on in other ways (such as the name of the mailing list used to coordinate Python packaging standards development).

Changed in version 3.5: The use of venv is now recommended for creating virtual environments.



See also

Python Packaging User Guide: Creating and using virtual environments

CHAPTER

TWO

BASIC USAGE

The standard packaging tools are all designed to be used from the command line.

The following command will install the latest version of a module and its dependencies from the Python Package Index:

```
python -m pip install SomePackage
```

1 Note

For POSIX users (including macOS and Linux users), the examples in this guide assume the use of a virtual environment.

For Windows users, the examples in this guide assume that the option to adjust the system PATH environment variable was selected when installing Python.

It's also possible to specify an exact or minimum version directly on the command line. When using comparator operators such as >, < or some other special character which get interpreted by shell, the package name and the version should be enclosed within double quotes:

```
python -m pip install SomePackage==1.0.4
                                            # specific version
python -m pip install "SomePackage>=1.0.4"
                                            # minimum version
```

Normally, if a suitable module is already installed, attempting to install it again will have no effect. Upgrading existing modules must be requested explicitly:

```
python -m pip install --upgrade SomePackage
```

More information and resources regarding pip and its capabilities can be found in the Python Packaging User Guide.

Creation of virtual environments is done through the venv module. Installing packages into an active virtual environment uses the commands shown above.

See also

Python Packaging User Guide: Installing Python Distribution Packages

CHAPTER

THREE

HOW DO I ...?

These are quick answers or links for some common tasks.

3.1 ... install pip in versions of Python prior to Python 3.4?

Python only started bundling pip with Python 3.4. For earlier versions, pip needs to be "bootstrapped" as described in the Python Packaging User Guide.

```
Python Packaging User Guide: Requirements for Installing Packages
```

3.2 ... install packages just for the current user?

Passing the --user option to python -m pip install will install a package just for the current user, rather than for all users of the system.

3.3 ... install scientific Python packages?

A number of scientific Python packages have complex binary dependencies, and aren't currently easy to install using pip directly. At this point in time, it will often be easier for users to install these packages by other means rather than attempting to install them with pip.

```
➢ See alsoPython Packaging User Guide: Installing Scientific Packages
```

3.4 ... work with multiple versions of Python installed in parallel?

On Linux, macOS, and other POSIX systems, use the versioned Python commands in combination with the -m switch to run the appropriate copy of pip:

```
python2 -m pip install SomePackage # default Python 2
python2.7 -m pip install SomePackage # specifically Python 2.7
python3 -m pip install SomePackage # default Python 3
python3.4 -m pip install SomePackage # specifically Python 3.4
```

Appropriately versioned pip commands may also be available.

On Windows, use the py Python launcher in combination with the -m switch:

```
py -2 -m pip install SomePackage # default Python 2
py -2.7 -m pip install SomePackage # specifically Python 2.7
py -3 -m pip install SomePackage # default Python 3
py -3.4 -m pip install SomePackage # specifically Python 3.4
```

COMMON INSTALLATION ISSUES

4.1 Installing into the system Python on Linux

On Linux systems, a Python installation will typically be included as part of the distribution. Installing into this Python installation requires root access to the system, and may interfere with the operation of the system package manager and other components of the system if a component is unexpectedly upgraded using pip.

On such systems, it is often better to use a virtual environment or a per-user installation when installing packages with pip.

4.2 Pip not installed

It is possible that pip does not get installed by default. One potential fix is:

```
python -m ensurepip --default-pip
```

There are also additional resources for installing pip.

4.3 Installing binary extensions

Python has typically relied heavily on source based distribution, with end users being expected to compile extension modules from source as part of the installation process.

With the introduction of support for the binary wheel format, and the ability to publish wheels for at least Windows and macOS through the Python Package Index, this problem is expected to diminish over time, as users are more regularly able to install pre-built extensions rather than needing to build them themselves.

Some of the solutions for installing scientific software that are not yet available as pre-built wheel files may also help with obtaining other binary extensions without needing to build them locally.

→ See also

Python Packaging User Guide: Binary Extensions

Α

GLOSSARY

>>>

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

. . .

Can refer to:

- The default Python prompt of the interactive shell when entering the code for an indented code block, when within a pair of matching left and right delimiters (parentheses, square brackets, curly braces or triple quotes), or after specifying a decorator.
- The Ellipsis built-in constant.

2to3

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

2to3 is available in the standard library as lib2to3; a standalone entry point is provided as Tools/scripts/2to3. See 2to3-reference.

abstract base class

Abstract base classes complement *duck-typing* by providing a way to define interfaces when other techniques like hasattr() would be clumsy or subtly wrong (for example with magic methods). ABCs introduce virtual subclasses, which are classes that don't inherit from a class but are still recognized by isinstance() and issubclass(); see the abc module documentation. Python comes with many built-in ABCs for data structures (in the collections.abc module), numbers (in the numbers module), streams (in the io module), import finders and loaders (in the importlib.abc module). You can create your own ABCs with the abc module.

annotation

A label associated with a variable, a class attribute or a function parameter or return value, used by convention as a *type hint*.

Annotations of local variables cannot be accessed at runtime, but annotations of global variables, class attributes, and functions are stored in the __annotations__ special attribute of modules, classes, and functions, respectively.

See *variable annotation*, *function annotation*, **PEP 484** and **PEP 526**, which describe this functionality. Also see annotations-howto for best practices on working with annotations.

argument

A value passed to a function (or method) when calling the function. There are two kinds of argument:

• *keyword argument*: an argument preceded by an identifier (e.g. name=) in a function call or passed as a value in a dictionary preceded by **. For example, 3 and 5 are both keyword arguments in the following calls to complex():

```
complex(real=3, imag=5)
complex(**{'real': 3, 'imag': 5})
```

• *positional argument*: an argument that is not a keyword argument. Positional arguments can appear at the beginning of an argument list and/or be passed as elements of an *iterable* preceded by *. For example, 3 and 5 are both positional arguments in the following calls:

```
complex(3, 5)
complex(*(3, 5))
```

Arguments are assigned to the named local variables in a function body. See the calls section for the rules governing this assignment. Syntactically, any expression can be used to represent an argument; the evaluated value is assigned to the local variable.

See also the *parameter* glossary entry, the FAQ question on the difference between arguments and parameters, and PEP 362.

asynchronous context manager

An object which controls the environment seen in an async with statement by defining __aenter__() and __aexit__() methods. Introduced by PEP 492.

asynchronous generator

A function which returns an *asynchronous generator iterator*. It looks like a coroutine function defined with async def except that it contains yield expressions for producing a series of values usable in an async for loop.

Usually refers to an asynchronous generator function, but may refer to an *asynchronous generator iterator* in some contexts. In cases where the intended meaning isn't clear, using the full terms avoids ambiguity.

An asynchronous generator function may contain await expressions as well as async for, and async with statements.

asynchronous generator iterator

An object created by a asynchronous generator function.

This is an *asynchronous iterator* which when called using the __anext__() method returns an awaitable object which will execute the body of the asynchronous generator function until the next yield expression.

Each yield temporarily suspends processing, remembering the location execution state (including local variables and pending try-statements). When the *asynchronous generator iterator* effectively resumes with another awaitable returned by __anext__(), it picks up where it left off. See **PEP 492** and **PEP 525**.

asynchronous iterable

An object, that can be used in an async for statement. Must return an *asynchronous iterator* from its __aiter__() method. Introduced by PEP 492.

asynchronous iterator

An object that implements the __aiter__() and __anext__() methods. __anext__() must return an awaitable object. async for resolves the awaitables returned by an asynchronous iterator's __anext__() method until it raises a StopAsyncIteration exception. Introduced by PEP 492.

attribute

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

It is possible to give an object an attribute whose name is not an identifier as defined by identifiers, for example using setattr(), if the object allows it. Such an attribute will not be accessible using a dotted expression, and would instead need to be retrieved with getattr().

awaitable

An object that can be used in an await expression. Can be a *coroutine* or an object with an __await__() method. See also **PEP 492**.

BDFL

Benevolent Dictator For Life, a.k.a. Guido van Rossum, Python's creator.

binary file

A file object able to read and write bytes-like objects. Examples of binary files are files opened in binary mode

('rb', 'wb' or 'rb+'), sys.stdin.buffer, sys.stdout.buffer, and instances of io.BytesIO and gzip.GzipFile.

See also *text file* for a file object able to read and write str objects.

borrowed reference

In Python's C API, a borrowed reference is a reference to an object, where the code using the object does not own the reference. It becomes a dangling pointer if the object is destroyed. For example, a garbage collection can remove the last *strong reference* to the object and so destroy it.

Calling Py_INCREF () on the *borrowed reference* is recommended to convert it to a *strong reference* in-place, except when the object cannot be destroyed before the last usage of the borrowed reference. The Py_NewRef () function can be used to create a new *strong reference*.

bytes-like object

An object that supports the bufferobjects and can export a C-contiguous buffer. This includes all bytes, bytearray, and array objects, as well as many common memoryview objects. Bytes-like objects can be used for various operations that work with binary data; these include compression, saving to a binary file, and sending over a socket.

Some operations need the binary data to be mutable. The documentation often refers to these as "read-write bytes-like objects". Example mutable buffer objects include bytearray and a memoryview of a bytearray. Other operations require the binary data to be stored in immutable objects ("read-only bytes-like objects"); examples of these include bytes and a memoryview of a bytes object.

bytecode

Python source code is compiled into bytecode, the internal representation of a Python program in the CPython interpreter. The bytecode is also cached in .pyc files so that executing the same file is faster the second time (recompilation from source to bytecode can be avoided). This "intermediate language" is said to run on a *virtual machine* that executes the machine code corresponding to each bytecode. Do note that bytecodes are not expected to work between different Python virtual machines, nor to be stable between Python releases.

A list of bytecode instructions can be found in the documentation for the dis module.

callable

A callable is an object that can be called, possibly with a set of arguments (see *argument*), with the following syntax:

```
callable(argument1, argument2, argumentN)
```

A *function*, and by extension a *method*, is a callable. An instance of a class that implements the __call__() method is also a callable.

callback

A subroutine function which is passed as an argument to be executed at some point in the future.

class

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

class variable

A variable defined in a class and intended to be modified only at class level (i.e., not in an instance of the class).

complex number

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

context manager

An object which controls the environment seen in a with statement by defining __enter__() and __exit__() methods. See PEP 343.

context variable

A variable which can have different values depending on its context. This is similar to Thread-Local Storage in which each execution thread may have a different value for a variable. However, with context variables, there may be several contexts in one execution thread and the main usage for context variables is to keep track of variables in concurrent asynchronous tasks. See contextvars.

contiguous

A buffer is considered contiguous exactly if it is either *C-contiguous* or *Fortran contiguous*. Zero-dimensional buffers are C and Fortran contiguous. In one-dimensional arrays, the items must be laid out in memory next to each other, in order of increasing indexes starting from zero. In multidimensional C-contiguous arrays, the last index varies the fastest when visiting items in order of memory address. However, in Fortran contiguous arrays, the first index varies the fastest.

coroutine

Coroutines are a more generalized form of subroutines. Subroutines are entered at one point and exited at another point. Coroutines can be entered, exited, and resumed at many different points. They can be implemented with the async def statement. See also PEP 492.

coroutine function

A function which returns a *coroutine* object. A coroutine function may be defined with the async def statement, and may contain await, async for, and async with keywords. These were introduced by **PEP** 492.

CPython

The canonical implementation of the Python programming language, as distributed on python.org. The term "CPython" is used when necessary to distinguish this implementation from others such as Jython or IronPython.

decorator

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

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

The same concept exists for classes, but is less commonly used there. See the documentation for function definitions and class definitions for more about decorators.

descriptor

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

For more information about descriptors' methods, see descriptors or the Descriptor How To Guide.

dictionary

An associative array, where arbitrary keys are mapped to values. The keys can be any object with __hash__() and __eq__() methods. Called a hash in Perl.

dictionary comprehension

A compact way to process all or part of the elements in an iterable and return a dictionary with the results. results = $\{n: n ** 2 \text{ for } n \text{ in range (10)} \}$ generates a dictionary containing key n mapped to value n ** 2. See comprehensions.

dictionary view

The objects returned from dict.keys(), dict.values(), and dict.items() are called dictionary views. They provide a dynamic view on the dictionary's entries, which means that when the dictionary changes, the view reflects these changes. To force the dictionary view to become a full list use list(dictview). See dict-views.

docstring

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

duck-typing

A programming style which does not look at an object's type to determine if it has the right interface; instead, the method or attribute is simply called or used ("If it looks like a duck and quacks like a duck, it must be a duck.") By emphasizing interfaces rather than specific types, well-designed code improves its flexibility by allowing polymorphic substitution. Duck-typing avoids tests using type() or isinstance(). (Note, however, that duck-typing can be complemented with abstract base classes.) Instead, it typically employs hasattr() tests or *EAFP* programming.

EAFP

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

expression

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

extension module

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

f-string

String literals prefixed with 'f' or 'F' are commonly called "f-strings" which is short for formatted string literals. See also **PEP 498**.

file object

An object exposing a file-oriented API (with methods such as read() or write()) to an underlying resource. Depending on the way it was created, a file object can mediate access to a real on-disk file or to another type of storage or communication device (for example standard input/output, in-memory buffers, sockets, pipes, etc.). File objects are also called *file-like objects* or *streams*.

There are actually three categories of file objects: raw *binary files*, buffered *binary files* and *text files*. Their interfaces are defined in the io module. The canonical way to create a file object is by using the open () function.

file-like object

A synonym for file object.

filesystem encoding and error handler

Encoding and error handler used by Python to decode bytes from the operating system and encode Unicode to the operating system.

The filesystem encoding must guarantee to successfully decode all bytes below 128. If the file system encoding fails to provide this guarantee, API functions can raise UnicodeError.

The sys.getfilesystemencoding() and sys.getfilesystemencodeerrors() functions can be used to get the filesystem encoding and error handler.

The filesystem encoding and error handler are configured at Python startup by the PyConfig_Read() function: see filesystem_encoding and filesystem_errors members of PyConfig.

See also the *locale encoding*.

finder

An object that tries to find the *loader* for a module that is being imported.

There are two types of finder: *meta path finders* for use with sys.meta_path, and *path entry finders* for use with sys.path_hooks.

See finders-and-loaders and importlib for much more detail.

floor division

Mathematical division that rounds down to nearest integer. The floor division operator is //. For example, the expression 11 // 4 evaluates to 2 in contrast to the 2.75 returned by float true division. Note that (-11) // 4 is -3 because that is -2.75 rounded *downward*. See **PEP 238**.

function

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

function annotation

An annotation of a function parameter or return value.

Function annotations are usually used for *type hints*: for example, this function is expected to take two int arguments and is also expected to have an int return value:

```
def sum_two_numbers(a: int, b: int) -> int:
    return a + b
```

Function annotation syntax is explained in section function.

See *variable annotation* and **PEP 484**, which describe this functionality. Also see annotations-howto for best practices on working with annotations.

__future_

A future statement, from __future__ import <feature>, directs the compiler to compile the current module using syntax or semantics that will become standard in a future release of Python. The __future__ module documents the possible values of *feature*. By importing this module and evaluating its variables, you can see when a new feature was first added to the language and when it will (or did) become the default:

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

garbage collection

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

generator

A function which returns a *generator iterator*. It looks like a normal function except that it contains yield expressions for producing a series of values usable in a for-loop or that can be retrieved one at a time with the next () function.

Usually refers to a generator function, but may refer to a *generator iterator* in some contexts. In cases where the intended meaning isn't clear, using the full terms avoids ambiguity.

generator iterator

An object created by a *generator* function.

Each <code>yield</code> temporarily suspends processing, remembering the location execution state (including local variables and pending try-statements). When the *generator iterator* resumes, it picks up where it left off (in contrast to functions which start fresh on every invocation).

generator expression

An expression that returns an iterator. It looks like a normal expression followed by a for clause defining a

loop variable, range, and an optional if clause. The combined expression generates values for an enclosing function:

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

generic function

A function composed of multiple functions implementing the same operation for different types. Which implementation should be used during a call is determined by the dispatch algorithm.

See also the single dispatch glossary entry, the functions.singledispatch() decorator, and PEP 443.

generic type

A type that can be parameterized; typically a container class such as list or dict. Used for type hints and annotations.

For more details, see generic alias types, PEP 483, PEP 484, PEP 585, and the typing module.

GIL

See global interpreter lock.

global interpreter lock

The mechanism used by the *CPython* interpreter to assure that only one thread executes Python *bytecode* at a time. This simplifies the CPython implementation by making the object model (including critical built-in types such as <code>dict</code>) implicitly safe against concurrent access. Locking the entire interpreter makes it easier for the interpreter to be multi-threaded, at the expense of much of the parallelism afforded by multi-processor machines.

However, some extension modules, either standard or third-party, are designed so as to release the GIL when doing computationally intensive tasks such as compression or hashing. Also, the GIL is always released when doing I/O.

Past efforts to create a "free-threaded" interpreter (one which locks shared data at a much finer granularity) have not been successful because performance suffered in the common single-processor case. It is believed that overcoming this performance issue would make the implementation much more complicated and therefore costlier to maintain.

hash-based pyc

A bytecode cache file that uses the hash rather than the last-modified time of the corresponding source file to determine its validity. See pyc-invalidation.

hashable

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

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

Most of Python's immutable built-in objects are hashable; mutable containers (such as lists or dictionaries) are not; immutable containers (such as tuples and frozensets) are only hashable if their elements are hashable. Objects which are instances of user-defined classes are hashable by default. They all compare unequal (except with themselves), and their hash value is derived from their id().

IDLE

An Integrated Development and Learning Environment for Python. idle is a basic editor and interpreter environment which ships with the standard distribution of Python.

immortal

Immortal objects are a CPython implementation detail introduced in PEP 683.

If an object is immortal, its *reference count* is never modified, and therefore it is never deallocated while the interpreter is running. For example, True and None are immortal in CPython.

immutable

An object with a fixed value. Immutable objects include numbers, strings and tuples. Such an object cannot

be altered. A new object has to be created if a different value has to be stored. They play an important role in places where a constant hash value is needed, for example as a key in a dictionary.

import path

A list of locations (or *path entries*) that are searched by the *path based finder* for modules to import. During import, this list of locations usually comes from sys.path, but for subpackages it may also come from the parent package's __path__ attribute.

importing

The process by which Python code in one module is made available to Python code in another module.

importer

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

interactive

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

interpreted

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

interpreter shutdown

When asked to shut down, the Python interpreter enters a special phase where it gradually releases all allocated resources, such as modules and various critical internal structures. It also makes several calls to the *garbage collector*. This can trigger the execution of code in user-defined destructors or weakref callbacks. Code executed during the shutdown phase can encounter various exceptions as the resources it relies on may not function anymore (common examples are library modules or the warnings machinery).

The main reason for interpreter shutdown is that the __main__ module or the script being run has finished executing.

iterable

An object capable of returning its members one at a time. Examples of iterables include all sequence types (such as list, str, and tuple) and some non-sequence types like dict, *file objects*, and objects of any classes you define with an __iter__() method or with a __getitem__() method that implements *sequence* semantics.

Iterables can be used in a for loop and in many other places where a sequence is needed (zip(), map(), ...). When an iterable object is passed as an argument to the built-in function iter(), it returns an iterator for the object. This iterator is good for one pass over the set of values. When using iterables, it is usually not necessary to call iter() or deal with iterator objects yourself. The for statement does that automatically for you, creating a temporary unnamed variable to hold the iterator for the duration of the loop. See also *iterator*, *sequence*, and *generator*.

iterator

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

More information can be found in typeiter.

CPython implementation detail: CPython does not consistently apply the requirement that an iterator define __iter__().

key function

A key function or collation function is a callable that returns a value used for sorting or ordering. For example, locale.strxfrm() is used to produce a sort key that is aware of locale specific sort conventions.

A number of tools in Python accept key functions to control how elements are ordered or grouped. They include min(), max(), sorted(), list.sort(), heapq.merge(), heapq.nsmallest(), heapq.nlargest(), and itertools.groupby().

There are several ways to create a key function. For example, the str.lower() method can serve as a key function for case insensitive sorts. Alternatively, a key function can be built from a lambda expression such as lambda r: (r[0], r[2]). Also, operator.attrgetter(), operator.itemgetter(), and operator.methodcaller() are three key function constructors. See the Sorting HOW TO for examples of how to create and use key functions.

keyword argument

See argument.

lambda

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

LBYL

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

In a multi-threaded environment, the LBYL approach can risk introducing a race condition between "the looking" and "the leaping". For example, the code, if key in mapping: return mapping[key] can fail if another thread removes *key* from *mapping* after the test, but before the lookup. This issue can be solved with locks or by using the EAFP approach.

list

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

list comprehension

A compact way to process all or part of the elements in a sequence and return a list with the results. result = $['\{:\#04x\}']$. format (x) for x in range (256) if x % 2 == 0] generates a list of strings containing even hex numbers (0x...) in the range from 0 to 255. The if clause is optional. If omitted, all elements in range (256) are processed.

loader

An object that loads a module. It must define a method named $load_module()$. A loader is typically returned by a *finder*. See also:

- finders-and-loaders
- importlib.abc.Loader
- PEP 302

locale encoding

On Unix, it is the encoding of the LC_CTYPE locale. It can be set with locale.setlocale(locale. LC_CTYPE , new_locale).

On Windows, it is the ANSI code page (ex: "cp1252").

On Android and VxWorks, Python uses "utf-8" as the locale encoding.

locale.getencoding() can be used to get the locale encoding.

See also the filesystem encoding and error handler.

magic method

An informal synonym for special method.

mapping

A container object that supports arbitrary key lookups and implements the methods specified in the

collections.abc.Mapping or collections.abc.MutableMapping abstract base classes. Examples include dict, collections.defaultdict, collections.OrderedDict and collections. Counter.

meta path finder

A *finder* returned by a search of sys.meta_path. Meta path finders are related to, but different from *path* entry finders.

See importlib.abc.MetaPathFinder for the methods that meta path finders implement.

metaclass

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

More information can be found in metaclasses.

method

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

method resolution order

Method Resolution Order is the order in which base classes are searched for a member during lookup. See python_2.3_mro for details of the algorithm used by the Python interpreter since the 2.3 release.

module

An object that serves as an organizational unit of Python code. Modules have a namespace containing arbitrary Python objects. Modules are loaded into Python by the process of *importing*.

See also package.

module spec

A namespace containing the import-related information used to load a module. An instance of importlib. machinery.ModuleSpec.

See also module-specs.

MRO

See method resolution order.

mutable

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

named tuple

The term "named tuple" applies to any type or class that inherits from tuple and whose indexable elements are also accessible using named attributes. The type or class may have other features as well.

Several built-in types are named tuples, including the values returned by time.localtime() and os. stat(). Another example is $sys.float_info$:

```
>>> 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
```

Some named tuples are built-in types (such as the above examples). Alternatively, a named tuple can be created from a regular class definition that inherits from tuple and that defines named fields. Such a class can be written by hand, or it can be created by inheriting typing. NamedTuple, or with the factory function collections.namedtuple(). The latter techniques also add some extra methods that may not be found in hand-written or built-in named tuples.

namespace

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

namespace package

A PEP 420 package which serves only as a container for subpackages. Namespace packages may have no physical representation, and specifically are not like a regular package because they have no __init__.py file.

See also module.

nested scope

The ability to refer to a variable in an enclosing definition. For instance, a function defined inside another function can refer to variables in the outer function. Note that nested scopes by default work only for reference and not for assignment. Local variables both read and write in the innermost scope. Likewise, global variables read and write to the global namespace. The nonlocal allows writing to outer scopes.

new-style class

Old name for the flavor of classes now used for all class objects. In earlier Python versions, only new-style classes could use Python's newer, versatile features like __slots__, descriptors, properties, __getattribute__(), class methods, and static methods.

object

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

package

A Python *module* which can contain submodules or recursively, subpackages. Technically, a package is a Python module with a __path__ attribute.

See also regular package and namespace package.

parameter

A named entity in a *function* (or method) definition that specifies an *argument* (or in some cases, arguments) that the function can accept. There are five kinds of parameter:

• positional-or-keyword: specifies an argument that can be passed either positionally or as a keyword argument. This is the default kind of parameter, for example foo and bar in the following:

```
def func(foo, bar=None): ...
```

• *positional-only*: specifies an argument that can be supplied only by position. Positional-only parameters can be defined by including a / character in the parameter list of the function definition after them, for example *posonly1* and *posonly2* in the following:

```
def func(posonly1, posonly2, /, positional_or_keyword): ...
```

• *keyword-only*: specifies an argument that can be supplied only by keyword. Keyword-only parameters can be defined by including a single var-positional parameter or bare * in the parameter list of the function definition before them, for example *kw only1* and *kw only2* in the following:

```
def func(arg, *, kw_only1, kw_only2): ...
```

• *var-positional*: specifies that an arbitrary sequence of positional arguments can be provided (in addition to any positional arguments already accepted by other parameters). Such a parameter can be defined by prepending the parameter name with *, for example *args* in the following:

```
def func(*args, **kwargs): ...
```

• *var-keyword*: specifies that arbitrarily many keyword arguments can be provided (in addition to any keyword arguments already accepted by other parameters). Such a parameter can be defined by prepending the parameter name with **, for example *kwargs* in the example above.

Parameters can specify both optional and required arguments, as well as default values for some optional arguments.

See also the *argument* glossary entry, the FAQ question on the difference between arguments and parameters, the inspect.Parameter class, the function section, and PEP 362.

path entry

A single location on the *import path* which the *path based finder* consults to find modules for importing.

path entry finder

A finder returned by a callable on sys.path_hooks (i.e. a path entry hook) which knows how to locate modules given a path entry.

See importlib.abc.PathEntryFinder for the methods that path entry finders implement.

path entry hook

A callable on the sys.path_hooks list which returns a *path entry finder* if it knows how to find modules on a specific *path entry*.

path based finder

One of the default meta path finders which searches an import path for modules.

path-like object

An object representing a file system path. A path-like object is either a str or bytes object representing a path, or an object implementing the os.PathLike protocol. An object that supports the os.PathLike protocol can be converted to a str or bytes file system path by calling the os.fspath() function; os.fsdecode() and os.fsencode() can be used to guarantee a str or bytes result instead, respectively. Introduced by PEP 519.

PEP

Python Enhancement Proposal. A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment. PEPs should provide a concise technical specification and a rationale for proposed features.

PEPs are intended to be the primary mechanisms for proposing major new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python. The PEP author is responsible for building consensus within the community and documenting dissenting opinions.

See PEP 1.

portion

A set of files in a single directory (possibly stored in a zip file) that contribute to a namespace package, as defined in **PEP 420**.

positional argument

See argument.

provisional API

A provisional API is one which has been deliberately excluded from the standard library's backwards compatibility guarantees. While major changes to such interfaces are not expected, as long as they are marked provisional, backwards incompatible changes (up to and including removal of the interface) may occur if deemed necessary by core developers. Such changes will not be made gratuitously – they will occur only if serious fundamental flaws are uncovered that were missed prior to the inclusion of the API.

Even for provisional APIs, backwards incompatible changes are seen as a "solution of last resort" - every attempt will still be made to find a backwards compatible resolution to any identified problems.

This process allows the standard library to continue to evolve over time, without locking in problematic design errors for extended periods of time. See PEP 411 for more details.

provisional package

See provisional API.

Python 3000

Nickname for the Python 3.x release line (coined long ago when the release of version 3 was something in the distant future.) This is also abbreviated "Py3k".

Pythonic

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

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

As opposed to the cleaner, Pythonic method:

```
for piece in food:
    print(piece)
```

qualified name

A dotted name showing the "path" from a module's global scope to a class, function or method defined in that module, as defined in **PEP 3155**. For top-level functions and classes, the qualified name is the same as the object's name:

When used to refer to modules, the *fully qualified name* means the entire dotted path to the module, including any parent packages, e.g. email.mime.text:

```
>>> import email.mime.text
>>> email.mime.text.__name__
'email.mime.text'
```

reference count

The number of references to an object. When the reference count of an object drops to zero, it is deallocated. Some objects are "immortal" and have reference counts that are never modified, and therefore the objects are never deallocated. Reference counting is generally not visible to Python code, but it is a key element of the *CPython* implementation. Programmers can call the sys.getrefcount() function to return the reference count for a particular object.

regular package

A traditional *package*, such as a directory containing an __init__.py file.

See also namespace package.

slots

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

sequence

An iterable which supports efficient element access using integer indices via the __qetitem__() special

method and defines a __len__() method that returns the length of the sequence. Some built-in sequence types are list, str, tuple, and bytes. Note that dict also supports __getitem__() and __len__(), but is considered a mapping rather than a sequence because the lookups use arbitrary <code>hashable</code> keys rather than integers.

The collections.abc.Sequence abstract base class defines a much richer interface that goes beyond just __getitem__() and __len__(), adding count(), index(), __contains__(), and __reversed__(). Types that implement this expanded interface can be registered explicitly using register(). For more documentation on sequence methods generally, see Common Sequence Operations.

set comprehension

A compact way to process all or part of the elements in an iterable and return a set with the results. results = {c for c in 'abracadabra' if c not in 'abc'} generates the set of strings {'r', 'd'}. See comprehensions.

single dispatch

A form of *generic function* dispatch where the implementation is chosen based on the type of a single argument.

slice

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

soft deprecated

A soft deprecated API should not be used in new code, but it is safe for already existing code to use it. The API remains documented and tested, but will not be enhanced further.

Soft deprecation, unlike normal deprecation, does not plan on removing the API and will not emit warnings.

See PEP 387: Soft Deprecation.

special method

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

statement

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

static type checker

An external tool that reads Python code and analyzes it, looking for issues such as incorrect types. See also *type hints* and the typing module.

strong reference

In Python's C API, a strong reference is a reference to an object which is owned by the code holding the reference. The strong reference is taken by calling Py_INCREF() when the reference is created and released with Py_DECREF() when the reference is deleted.

The $Py_NewRef()$ function can be used to create a strong reference to an object. Usually, the $Py_DECREF()$ function must be called on the strong reference before exiting the scope of the strong reference, to avoid leaking one reference.

See also borrowed reference.

text encoding

A string in Python is a sequence of Unicode code points (in range U+0000-U+10FFFF). To store or transfer a string, it needs to be serialized as a sequence of bytes.

Serializing a string into a sequence of bytes is known as "encoding", and recreating the string from the sequence of bytes is known as "decoding".

There are a variety of different text serialization codecs, which are collectively referred to as "text encodings".

text file

A file object able to read and write str objects. Often, a text file actually accesses a byte-oriented datastream

and handles the *text encoding* automatically. Examples of text files are files opened in text mode ('r' or 'w'), sys.stdin, sys.stdout, and instances of io.StringIO.

See also binary file for a file object able to read and write bytes-like objects.

triple-quoted string

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

type

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

type alias

A synonym for a type, created by assigning the type to an identifier.

Type aliases are useful for simplifying type hints. For example:

could be made more readable like this:

```
Color = tuple[int, int, int]

def remove_gray_shades(colors: list[Color]) -> list[Color]:
    pass
```

See typing and PEP 484, which describe this functionality.

type hint

An *annotation* that specifies the expected type for a variable, a class attribute, or a function parameter or return value.

Type hints are optional and are not enforced by Python but they are useful to *static type checkers*. They can also aid IDEs with code completion and refactoring.

Type hints of global variables, class attributes, and functions, but not local variables, can be accessed using typing.get_type_hints().

See typing and PEP 484, which describe this functionality.

universal newlines

A manner of interpreting text streams in which all of the following are recognized as ending a line: the Unix end-of-line convention ' \r ', the Windows convention ' \r ', and the old Macintosh convention ' \r '. See PEP 278 and PEP 3116, as well as bytes.splitlines() for an additional use.

variable annotation

An annotation of a variable or a class attribute.

When annotating a variable or a class attribute, assignment is optional:

```
class C:
field: 'annotation'
```

Variable annotations are usually used for *type hints*: for example this variable is expected to take int values:

```
count: int = 0
```

Variable annotation syntax is explained in section annassign.

See *function annotation*, **PEP 484** and **PEP 526**, which describe this functionality. Also see annotations-howto for best practices on working with annotations.

virtual environment

A cooperatively isolated runtime environment that allows Python users and applications to install and upgrade Python distribution packages without interfering with the behaviour of other Python applications running on the same system.

See also venv.

virtual machine

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

Zen of Python

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

В

ABOUT THIS DOCUMENTATION

Python's documentation is generated from reStructuredText sources using Sphinx, a documentation generator originally created for Python and now maintained as an independent project.

Development of the documentation and its toolchain is an entirely volunteer effort, just like Python itself. If you want to contribute, please take a look at the reporting-bugs page for information on how to do so. New volunteers are always welcome!

Many thanks go to:

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

B.1 Contributors to the Python documentation

Many people have contributed to the Python language, the Python standard library, and the Python documentation. See Misc/ACKS in the Python source distribution for a partial list of contributors.

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

C

HISTORY AND LICENSE

C.1 History of the software

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

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

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

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

Release	Derived from	Year	Owner	GPL compatible?
0.9.0 thru 1.2	n/a	1991-1995	CWI	yes
1.3 thru 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	no
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2 and above	2.1.1	2001-now	PSF	yes



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

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

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

Python software and documentation are licensed under the PSF License Agreement.

Starting with Python 3.8.6, examples, recipes, and other code in the documentation are dual licensed under the PSF License Agreement and the *Zero-Clause BSD license*.

Some software incorporated into Python is under different licenses. The licenses are listed with code falling under that license. See *Licenses and Acknowledgements for Incorporated Software* for an incomplete list of these licenses.

C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.12.8

- 1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 3.12.8 software in source or binary form and its associated documentation.
- 2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 3.12.8 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2023 Python Software Foundation; All Rights Reserved" are retained in Python 3.12.8 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 3.12.8 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 3.12.8.
- 4. PSF is making Python 3.12.8 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 3.12.8 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
- 5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.12.8 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.12.8, 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 3.12.8, Licensee agrees to be bound by the terms and conditions of this License Agreement.

C.2.2 BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

- 1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").
- 2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.
- 3. BeOpen is making the Software available to Licensee on an "AS IS" basis.
 BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF
 EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR
 WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE
 USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
- 4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
- 5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
- 6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.
- 7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

C.2.3 CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

- 1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.
- 2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All

(continues on next page)

(continued from previous page)

Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the internet using the following URL: http://hdl.handle.net/1895.22/1013."

- 3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.
- 4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
- 5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
- 6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
- 7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
- 8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

C.2.4 CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright

(continues on next page)

notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

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

C.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.12.8 DOCU-MENTATION

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.

C.3 Licenses and Acknowledgements for Incorporated Software

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

C.3.1 Mersenne Twister

The $_$ random C extension underlying the random module includes code based on a download from http://www.math. sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html. The following are the verbatim comments from the original code:

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

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

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

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

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

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

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

Any feedback is very welcome. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)

C.3.2 Sockets

The socket module uses the functions, getaddrinfo(), and getnameinfo(), which are coded in separate source files from the WIDE Project, 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.

C.3.3 Asynchronous socket services

The test.support.asynchat and test.support.asyncore modules contain the following notice:

Copyright 1996 by Sam Rushing

All Rights Reserved

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

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

C.3.4 Cookie management

The http.cookies module contains the following notice:

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

All Rights Reserved

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

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

C.3.5 Execution tracing

The trace module contains the following notice:

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

Author: Zooko O'Whielacronx

http://zooko.com/
mailto:zooko@zooko.com

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

Author: Skip Montanaro

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

Author: Andrew Dalke

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

Author: Skip Montanaro

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

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

C.3.6 UUencode and UUdecode functions

The uu module contains the following notice:

Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.

All Rights Reserved

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

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

Modified by Jack Jansen, CWI, July 1995:

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

C.3.7 XML Remote Procedure Calls

The xmlrpc.client module contains the following notice:

The XML-RPC client interface is

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

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

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

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

C.3.8 test_epoll

The test.test_epoll module contains the following notice:

Copyright (c) 2001-2006 Twisted Matrix Laboratories.

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

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

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

C.3.9 Select kqueue

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

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

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

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

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

C.3.10 SipHash24

The file Python/pyhash.c contains Marek Majkowski' implementation of Dan Bernstein's SipHash24 algorithm. It contains the following note:

```
<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/siphash24/little)
  djb (supercop/crypto_auth/siphash24/little2)
  Jean-Philippe Aumasson (https://131002.net/siphash/siphash24.c)
```

C.3.11 strtod and dtoa

The file Python/dtoa.c, which supplies C functions dtoa and strtod for conversion of C doubles to and from strings, is derived from the file of the same name by David M. Gay, currently available from https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c. The original file, as retrieved on March 16, 2009, contains the following copyright and licensing notice:

C.3.12 OpenSSL

The modules hashlib, posix, ssl, crypt use the OpenSSL library for added performance if made available by the operating system. Additionally, the Windows and macOS installers for Python may include a copy of the OpenSSL libraries, so we include a copy of the OpenSSL license here. For the OpenSSL 3.0 release, and later releases derived from that, the Apache License v2 applies:

```
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.
  "Licensor" 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
  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 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 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

C.3.13 expat

The pyexpat extension is built using an included copy of the expat sources unless the build is configured --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 the following conditions:

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

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

C.3.14 libffi

The _ctypes C extension underlying the ctypes module is built using an included copy of the libffi sources unless the build is configured --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.

C.3.15 zlib

The zlib extension is built using an included copy of the zlib sources if the zlib version found on the system is too old to be used for the build:

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:

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

C.3.16 cfuhash

The implementation of the hash table used by the tracemalloc is based on the cfuhash project:

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.

C.3.17 libmpdec

The $_decimal\ C$ extension underlying the $decimal\ module$ is built using an included copy of the libmpdec library unless the build is configured --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.

C.3.18 W3C C14N test suite

The C14N 2.0 test suite in the test package (Lib/test/xmltestdata/c14n-20/) was retrieved from the W3C website at https://www.w3.org/TR/xml-c14n2-testcases/ and is distributed under the 3-clause BSD license:

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, 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.

C.3.19 Audioop

The audioop module uses the code base in g771.c file of the SoX project. https://sourceforge.net/projects/sox/files/sox/12.17.7/sox-12.17.7.tar.gz

This source code is a product of Sun Microsystems, Inc. and is provided for unrestricted use. Users may copy or modify this source code without charge.

SUN SOURCE CODE IS PROVIDED AS IS WITH NO WARRANTIES OF ANY KIND INCLUDING THE WARRANTIES OF DESIGN, MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE, OR ARISING FROM A COURSE OF DEALING, USAGE OR TRADE PRACTICE.

Sun source code is provided with no support and without any obligation on the part of Sun Microsystems, Inc. to assist in its use, correction, modification or enhancement.

SUN MICROSYSTEMS, INC. SHALL HAVE NO LIABILITY WITH RESPECT TO THE INFRINGEMENT OF COPYRIGHTS, TRADE SECRETS OR ANY PATENTS BY THIS SOFTWARE OR ANY PART THEREOF.

In no event will Sun Microsystems, Inc. be liable for any lost revenue or profits or other special, indirect and consequential damages, even if Sun has been advised of the possibility of such damages.

Sun Microsystems, Inc. 2550 Garcia Avenue Mountain View, California 94043

C.3.20 asyncio

Parts of the asyncio module are incorporated from uvloop 0.16, which is distributed under the MIT license:

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 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.

APPENDIX

D

COPYRIGHT

Python and this documentation is:

Copyright © 2001-2023 Python Software Foundation. All rights reserved.

Copyright © 2000 BeOpen.com. All rights reserved.

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

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

See *History and License* for complete license and permissions information.

INDEX

Non-alphabetical	docstring, 15 duck-typing, 15
, 11 2to3, 11	
>>>, 11	E
future, 16	EAFP, 15
slots, 23	expression, 15
_	extension module, 15
A	F
abstract base class, 11	F
annotation, 11	f-string, 15
argument, 11	file object, 15
asynchronous context manager, 12	file-like object, 15
asynchronous generator, 12	filesystem encoding and error handler, 15
asynchronous generator iterator, 12	finder, 16
asynchronous iterable, 12	floor division, 16
asynchronous iterator, 12	Fortran contiguous, 14
attribute, 12	function, 16
awaitable, 12	function annotation, 16
В	G
BDFL, 12	garbage collection, 16
binary file, 12	generator, 16
borrowed reference, 13	generator expression, 16
bytecode, 13	generator iterator, 16
bytes-like object, 13	generic function, 17
	generic type, 17
C	GIL, 17
callable, 13	global interpreter lock, 17
callback, 13	11
C-contiguous, 14	Н
class, 13	hash-based pyc, 17
class variable, 13	hashable, 17
complex number, 13	1
context manager, 13	I
context variable, 14	IDLE, 17
contiguous, 14	immortal, 17
coroutine, 14	immutable, 17
coroutine function, 14	import path, 18
CPython, 14	importer, 18
D	importing, 18
	interactive, 18
decorator, 14	interpreted, 18
descriptor, 14	interpreter shutdown, 18
dictionary, 14	iterable, 18
dictionary comprehension, 14	iterator, 18
dictionary view 15	

K	PEP 343,13
key function, 19	PEP 362, 12, 22
keyword argument, 19	PEP 411, 22
1	PEP 420, 21, 22
L	PEP 443, 17
lambda, 19	PEP 483, 17 PEP 484, 11, 16, 17, 25
LBYL, 19	PEP 494, 11, 10, 17, 25
list, 19	PEP 498, 15
list comprehension, 19	PEP 519, 22
loader, 19	PEP 525, 12
locale encoding, 19	PEP 526, 11, 25
M	PEP 585, 17
	PEP 683, 17
magic	PEP 3116, 25
method, 19	PEP 3155, 23
magic method, 19 mapping, 19	Pythonic, 23
meta path finder, 20	
metaclass, 20	Q
method, 20	qualified name, 23
magic, 19	D
special, 24	R
method resolution order, 20	reference count, 23
module, 20	regular package, 23
module spec, 20	S
MRO, 20	_
mutable, 20	sequence, 23
N	set comprehension, 24
• •	single dispatch, 24
named tuple, 20	slice, 24
namespace, 21	soft deprecated, 24
namespace package, 21	special method, 24
nested scope, 21	special method, 24
new-style class, 21	statement, 24
0	static type checker, 24
object, 21	strong reference, 24
object, 21	- -
P	1
package, 21	text encoding, 24
parameter, 21	text file, 24
path based finder, 22	triple-quoted string, 25
path entry, 22	type, 25
path entry finder, 22	type alias, 25
path entry hook, 22	type hint, 25
path-like object, 22	U
PEP, 22	25
portion, 22	universal newlines, 25
positional argument, 22	V
provisional API, 22	variable annotation, 25
provisional package, 22	virtual environment, 26
Python 3000, 23 Python Enhancement Proposals	virtual machine, 26
PEP 1, 22	_
PEP 238, 16	Z
PEP 278, 25	Zen of Python, 26
DED 202 10	- /

50 Index