Distributing Python Modules

Release 3.10.4

Guido van Rossum
and the Python development team

March 23, 2022

Python Software Foundation
Email: docs@python.org
# CONTENTS

1 Key terms .......................................................... 3
2 Open source licensing and collaboration ...................... 5
3 Installing the tools .................................................. 7
4 Reading the Python Packaging User Guide ................... 9
5 How do I…? .................................................................. 11
   5.1 … choose a name for my project? ........................... 11
   5.2 … create and distribute binary extensions? .............. 11
A Glossary .................................................................. 13
B About these documents ............................................. 27
   B.1 Contributors to the Python Documentation ............ 27
C History and License ................................................... 29
   C.1 History of the software ........................................ 29
   C.2 Terms and conditions for accessing or otherwise using Python .............................................. 30
   C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.10.4 .................................................. 30
   C.2.2 BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0 ........................................... 31
   C.2.3 CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1 .................................................... 32
   C.2.4 CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2 .............................. 33
   C.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON 3.10.4 DOCUMENTATION .............................................................................................................. 33
   C.3 Licenses and Acknowledgements for Incorporated Software .................................................. 34
   C.3.1 Mersenne Twister ................................................ 34
   C.3.2 Sockets .............................................................. 35
   C.3.3 Asynchronous socket services ............................ 35
   C.3.4 Cookie management .......................................... 36
   C.3.5 Execution tracing .............................................. 36
   C.3.6 UUencode and UUdecode functions ................... 37
   C.3.7 XML Remote Procedure Calls ............................ 37
   C.3.8 test_epoll .......................................................... 38
   C.3.9 Select kqueue ................................................... 38
   C.3.10 SipHash24 ....................................................... 39
   C.3.11 strtod and dtoa ............................................... 39
   C.3.12 OpenSSL ........................................................ 40
   C.3.13 expat .............................................................. 42
   C.3.14 libffi ............................................................. 42
   C.3.15 zlib ............................................................... 43
   C.3.16 cfuhash ........................................................ 43
   C.3.17 libmpdec ........................................................ 44
   C.3.18 W3C C14N test suite ...................................... 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 distribution part of the process. For a guide to installing other Python projects, refer to the installation guide.

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.
• 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 are 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 both GitHub and Bitbucket.

• 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).

• setuptools is a (largely) drop-in replacement for distutils first published in 2004. Its most notable addition over the unmodified distutils tools was the ability to declare dependencies on other packages. It is currently recommended as a more regularly updated alternative to distutils that offers consistent support for more recent packaging standards across a wide range of Python versions.

• wheel (in this context) is a project that adds the bdist_wheel command to distutils/setuptools. This produces a cross platform binary packaging format (called “wheels” or “wheel files” and defined in PEP 427) that allows Python libraries, even those including binary extensions, to be installed on a system without needing to be built locally.
In most parts of the world, software is automatically covered by copyright. This means that other developers require explicit permission to copy, use, modify and redistribute the software.

Open source licensing is a way of explicitly granting such permission in a relatively consistent way, allowing developers to share and collaborate efficiently by making common solutions to various problems freely available. This leaves many developers free to spend more time focusing on the problems that are relatively unique to their specific situation.

The distribution tools provided with Python are designed to make it reasonably straightforward for developers to make their own contributions back to that common pool of software if they choose to do so.

The same distribution tools can also be used to distribute software within an organisation, regardless of whether that software is published as open source software or not.
The standard library does not include build tools that support modern Python packaging standards, as the core development team has found that it is important to have standard tools that work consistently, even on older versions of Python.

The currently recommended build and distribution tools can be installed by invoking the `pip` module at the command line:

```
python -m pip install setuptools wheel twine
```

**Note:** For POSIX users (including macOS and Linux users), these instructions assume the use of a virtual environment.

For Windows users, these instructions assume that the option to adjust the system PATH environment variable was selected when installing Python.

The Python Packaging User Guide includes more details on the currently recommended tools.
READING THE PYTHON PACKAGING USER GUIDE

The Python Packaging User Guide covers the various key steps and elements involved in creating and publishing a project:

- Project structure
- Building and packaging the project
- Uploading the project to the Python Package Index
- The .pypirc file
These are quick answers or links for some common tasks.

5.1 ... choose a name for my project?

This isn’t an easy topic, but here are a few tips:

- check the Python Package Index to see if the name is already in use
- check popular hosting sites like GitHub, Bitbucket, etc to see if there is already a project with that name
- check what comes up in a web search for the name you’re considering
- avoid particularly common words, especially ones with multiple meanings, as they can make it difficult for users to find your software when searching for it

5.2 ... create and distribute binary extensions?

This is actually quite a complex topic, with a variety of alternatives available depending on exactly what you’re aiming to achieve. See the Python Packaging User Guide for more information and recommendations.

See also:

Python Packaging User Guide: Binary Extensions
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()*:

  ```python
  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:

  ```python
  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 referenced by name using dotted expressions. For example, if an object o has an attribute a it would be referenced as o.a.

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. It does not modify the object reference count. 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.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.

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).

coefficient  The implicit conversion of an instance of one type to another during an operation which involves two arguments of the same type. For example, int(3.15) converts the floating point number to the integer 3, but in 3+4.5, each argument is of a different type (one int, one float), and both must be converted to the same type before they can be added or it will raise a TypeError. Without coercion, all arguments of even compatible types would have to be normalized to the same value by the programmer, e.g., float(3)+4.5 rather than just 3+4.5.

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

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

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.

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

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

```python
def f(arg):
    ...
```
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 for n 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.

Since Python 3.3, 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 PEP 302, PEP 420 and PEP 451 for much more detail.

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 `yield` 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:

```python
>>> 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 `functools.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 `dict`) 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 Environment for Python. IDLE is a basic editor and interpreter environment which ships with the standard distribution of Python.

**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 typeit.py.

**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, the operator module provides three key function constructors: attrgetter(), itemgetter(), and methodcaller(). 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.

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).
locale.getpreferredencoding(False) can be used to get the locale encoding.

Python uses the filesystem encoding and error handler to convert between Unicode filenames and bytes filenames.

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 = [chr(i) for i in range(256)] generates a list of strings containing each hex number (0x..) in the range from 0 to 255. The if clause is optional. If omitted, all elements in range(256) are processed.

loader  An object that loads a module. It must define a method named load_module(). A loader is typically returned by a finder. See PEP 302 for details and importlib.abc.Loader for an abstract base class.

magic method  An informal synonym for special method.

mapping  A container object that supports arbitrary key lookups and implements the methods specified in the Mapping or 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 The Python 2.3 Method Resolution Order 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.

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`:

```python
>>> 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 with the factory function `collections.namedtuple()`. The latter technique also adds 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 an `__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:

  ```python
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:

  ```python
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:

  ```python
def func(kw_only1, kw_only2, *, keyword_or_positional): ...
```
```python
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:

```python
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_hook` 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”.

---

22 Appendix A. Glossary
**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:

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

As opposed to the cleaner, Pythonic method:

```python
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:

```python
>>> class C:
...
    ...
    class D:
    ...
        def meth(self):
        ...
        pass
...

>>> C.__qualname__
'C'
>>> C.D.__qualname__
'C.D'
>>> C.D.meth.__qualname__
'C.D.meth'
```

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`:

```python
>>> 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. Reference counting is generally not visible to Python code, but it is a key element of the CPython implementation. The `sys` module defines a `getrefcount()` function that programmers can call to return the reference count for a particular object.

**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 `__getitem__()`, `__len__()` 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__()`, `__len__()`, but is considered a mapping rather than a sequence because the lookups use arbitrary immutable keys rather than integers.

The `collections.abc.Sequence` abstract base class defines a much richer interface that goes beyond just `__getitem__()`, `__len__()`, adding `count()`, `index()` and `__contains__()` and `__reversed__()`. Types that implement this expanded interface can be registered explicitly using `register()`.

**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.

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`.

strong reference  In Python’s C API, a strong reference is a reference to an object which increments the object’s reference count when it is created and decrements the object’s reference count when it 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 codec which encodes Unicode strings to bytes.

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:

```python
def remove_gray_shades(
    colors: list[tuple[int, int, int]]) -> list[tuple[int, int, int]]:
    pass
```

could be made more readable like this:

```python
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 analysis tools, and 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 '\n', the Windows convention '\r\n', 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:

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

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

```python
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.
ABOUT THESE DOCUMENTS

These documents are generated from reStructuredText sources by Sphinx, a document processor specifically written for the Python documentation.

Development of the documentation and its toolchain 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 writer of much of the content;
- the Docutils project for creating reStructuredText and the Docutils suite;
- Fredrik Lundh for his Alternative Python Reference project from which Sphinx got many good ideas.

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.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.

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

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

Thanks to the many outside volunteers who have worked under Guido’s direction to make these releases possible.
C.2 Terms and conditions for accessing or otherwise using Python

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 3.10.4

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and
   the Individual or Organization ("Licensee") accessing and otherwise
   using Python
   3.10.4 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 3.10.4 alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's
   notice of
   copyright, i.e., "Copyright © 2001-2022 Python Software Foundation; All
   Rights
   Reserved" are retained in Python 3.10.4 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 3.10.4 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
   3.10.4.

4. PSF is making Python 3.10.4 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 3.10.4 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.10.4
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A
   RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.10.4, OR ANY
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 3.10.4, 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.

(continues on next 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 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 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 3.10.4 DOCUMENTATION

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 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, [http://www.wide.ad.jp/](http://www.wide.ad.jp/).

```
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 asyncchat and asyncore modules contain the following notice:

```
Copyright 1996 by Sam Rushing

All Rights Reserved

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

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```
C.3.4 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 MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.
C.3.8 test_epoll

The 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 http://www.netlib.org/fp/. The original file, as retrieved on March 16, 2009, contains the following copyright and licensing notice:

```
/*****************************************************************
*                                                                *
* The author of this software is David M. Gay.                  *
*                                                                *
* Copyright (c) 1991, 2000, 2001 by Lucent Technologies.        *
*                                                                *
* Permission to use, copy, modify, and distribute this software for any *
* purpose without fee is hereby granted, provided that this entire notice *
* is included in all copies of any software which is or includes a copy *
* or modification of this software and in all copies of the supporting *
* documentation for such software.                              *
*                                                                *
* THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED *
* WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY *
* REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY *
* OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.      *
*                                                                *
*********************************************************************/
```
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:

```
LICENSE ISSUES
===============

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License
---------------

/* ====================================================================
 * Copyright (c) 1998-2008 The OpenSSL 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. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT `AS IS' AND ANY
 * EXPRESSED 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 OpenSSL PROJECT OR
 * ITS 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)
 *
 */
```

(continues on next page)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
* ---------------------------------------------------------------------------
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com). This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
* /

Original SSLeay License
-----------------------
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 * 
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 * 
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are acheived. The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code. The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 * 
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 * 
 * 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 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. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *       "This product includes cryptographic software written by
 *       Eric Young (eay@cryptsoft.com)"
 *       The word 'cryptographic' can be left out if the rouines from the library
 *       being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *       "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 * 
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``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

(continues on next page)
Distributing Python Modules, Release 3.10.4

(continued from previous page)

* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed. i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/

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 extension 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

(continues on next page)
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
C.3.17 libmpdec

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:

(continues on next page)
* 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.
APPENDIX

COPYRIGHT

Python and this documentation is:
Copyright © 2001-2022 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.
Non-alphabetical
.... 13
2to3, 13
>>, 13
__future__, 17
__slots__, 23
A
abstract base class, 13
annotation, 13
argument, 13
asynchronous context manager, 14
asynchronous generator, 14
asynchronous generator iterator, 14
asynchronous iterable, 14
asynchronous iterator, 14
attribute, 14
available, 14
B
BDFL, 14
binary file, 14
borrowed reference, 14
bytecode, 15
bytes-like object, 14
C
callback, 15
C-contiguous, 15
class, 15
class variable, 15
coercion, 15
complex number, 15
context manager, 15
context variable, 15
contiguous, 15
coroutine, 15
coroutine function, 15
CPython, 15
D
decorator, 15
descriptor, 16
dictionary, 16
dictionary comprehension, 16
dictionary view, 16
docstring, 16
duck-typing, 16
E
EAFP, 16
expression, 16
extension module, 16
F
f-string, 16
file object, 16
file-like object, 17
filesystem encoding and error handler, 17
finder, 17
floor division, 17
Fortran contiguous, 15
function, 17
function annotation, 17
G
garbage collection, 17
generator, 17
generator expression, 18
generator iterator, 17
generic function, 18
generic type, 18
GIL, 18
global interpreter lock, 18
H
hash-based pyc, 18
hashable, 18
I
IDLE, 18
immutable, 18
import path, 18
importer, 19
importing, 18
interactive, 19
interpreted, 19
interpreter shutdown, 19
iterable, 19
iterator, 19
K
key function, 19
keyword argument, 19

L
lambda, 19
LBYL, 20
list, 20
list comprehension, 20
loader, 20
locale encoding, 20

M
magic
  method, 20
magic method, 20
mapping, 20
meta path finder, 20
metaclass, 20
method, 20
  magic, 20
  special, 24
method resolution order, 20
module, 20
module spec, 20
MRO, 20
mutable, 20

N
named tuple, 21
namespace, 21
namespace package, 21
nested scope, 21
new-style class, 21

O
object, 21

P
package, 21
parameter, 21
path based finder, 22
path entry, 22
path entry finder, 22
path entry hook, 22
path-like object, 22
PEP, 22
portion, 22
positional argument, 22
provisional API, 22
provisional package, 22
PyPI
  (see Python Package Index (PyPI)), 7
Python 3000, 22
Python Enhancement Proposals
  PEP 1, 22
  PEP 238, 17
  PEP 278, 25
  PEP 302, 17, 20
  PEP 343, 15
  PEP 362, 14, 22
  PEP 411, 22
  PEP 420, 17, 21, 22
  PEP 427, 3
  PEP 443, 18
  PEP 451, 17
  PEP 483, 18
  PEP 484, 13, 17, 18, 24, 25
  PEP 492, 14, 15
  PEP 498, 16
  PEP 519, 22
  PEP 525, 14
  PEP 526, 13, 25
  PEP 585, 18
  PEP 3116, 25
  PEP 3155, 23
Python Package Index (PyPI), 7
Pythonic, 23
qualified name, 23
reference count, 23
regular package, 23
sequence, 23
set comprehension, 23
single dispatch, 24
slice, 24
special
  method, 24
special method, 24
statement, 24
strong reference, 24

text encoding, 24
text file, 24
triple-quoted string, 24
type, 24
type alias, 24
type hint, 24
universal newlines, 25

variable annotation, 25
virtual environment, 25
virtual machine, 25

Zen of Python, 25