:mod:`sys` --- System-specific parameters and functions

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 4)

Unknown directive type "module".

```
.. module:: sys
   :synopsis: Access system-specific parameters and functions.
```

This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 14)

Unknown directive type "data".

```
.. data:: abiflags

   On POSIX systems where Python was built with the standard ``configure``
   script, this contains the ABI flags as specified by :pep:`3149`.

   .. versionchanged:: 3.8
      Default flags became an empty string (``m`` flag for pymalloc has been
      removed).

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 26)

Unknown directive type "function".

```
.. function:: addaudithook(hook)

   Append the callable *hook* to the list of active auditing hooks for the
   current (sub)interpreter.

   When an auditing event is raised through the :func:`sys.audit` function, each
   hook will be called in the order it was added with the event name and the
   tuple of arguments. Native hooks added by :c:func:`PySys_AddAuditHook` are
   called first, followed by hooks added in the current (sub)interpreter.  Hooks
   can then log the event, raise an exception to abort the operation,
   or terminate the process entirely.

   .. audit-event:: sys.addaudithook "" sys.addaudithook

      Calling :func:`sys.addaudithook` will itself raise an auditing event
      named ``sys.addaudithook`` with no arguments. If any
      existing hooks raise an exception derived from :class:`RuntimeError`, the
      new hook will not be added and the exception suppressed. As a result,
      callers cannot assume that their hook has been added unless they control
      all existing hooks.

   See the :ref:`audit events table <audit-events>` for all events raised by
   CPython, and :pep:`578` for the original design discussion.

   .. versionadded:: 3.8

   .. versionchanged:: 3.8.1

      Exceptions derived from :class:`Exception` but not :class:`RuntimeError`
      are no longer suppressed.
```

```
   .. impl-detail::

      When tracing is enabled (see :func:`settrace`), Python hooks are only
      traced if the callable has a ``__cantrace__`` member that is set to a
      true value. Otherwise, trace functions will skip the hook.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 64)

Unknown directive type "data".

```
.. data:: argv

   The list of command line arguments passed to a Python script. ``argv[0]`` is the
   script name (it is operating system dependent whether this is a full pathname or
   not).  If the command was executed using the :option:`-c` command line option to
   the interpreter, ``argv[0]`` is set to the string ``'-c'``.  If no script name
   was passed to the Python interpreter, ``argv[0]`` is the empty string.

   To loop over the standard input, or the list of files given on the
   command line, see the :mod:`fileinput` module.

   See also :data:`sys.orig_argv`.

   .. note::
      On Unix, command line arguments are passed by bytes from OS.  Python decodes
      them with filesystem encoding and "surrogateescape" error handler.
      When you need original bytes, you can get it by
      ``[os.fsencode(arg) for arg in sys.argv]``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 86)

Unknown directive type "function".

```
.. function:: audit(event, *args)

   .. index:: single: auditing

   Raise an auditing event and trigger any active auditing hooks.
   *event* is a string identifying the event, and *args* may contain
   optional arguments with more information about the event.  The
   number and types of arguments for a given event are considered a
   public and stable API and should not be modified between releases.

   For example, one auditing event is named ``os.chdir``. This event has
   one argument called *path* that will contain the requested new
   working directory.

   :func:`sys.audit` will call the existing auditing hooks, passing
   the event name and arguments, and will re-raise the first exception
   from any hook. In general, if an exception is raised, it should not
   be handled and the process should be terminated as quickly as
   possible. This allows hook implementations to decide how to respond
   to particular events: they can merely log the event or abort the
   operation by raising an exception.

   Hooks are added using the :func:`sys.addaudithook` or
   :c:func:`PySys_AddAuditHook` functions.

   The native equivalent of this function is :c:func:`PySys_Audit`. Using the
   native function is preferred when possible.

   See the :ref:`audit events table <audit-events>` for all events raised by
   CPython.

   .. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 120)

Unknown directive type "data".

```
.. data:: base_exec_prefix

   Set during Python startup, before ``site.py`` is run, to the same value as
   :data:`exec_prefix`. If not running in a
   :ref:`virtual environment <venv-def>`, the values will stay the same; if
   ``site.py`` finds that a virtual environment is in use, the values of
   :data:`prefix` and :data:`exec_prefix` will be changed to point to the
   virtual environment, whereas :data:`base_prefix` and
   :data:`base_exec_prefix` will remain pointing to the base Python
   installation (the one which the virtual environment was created from).

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 134)

Unknown directive type "data".

```
.. data:: base_prefix

   Set during Python startup, before ``site.py`` is run, to the same value as
   :data:`prefix`. If not running in a :ref:`virtual environment <venv-def>`, the values
   will stay the same; if ``site.py`` finds that a virtual environment is in
   use, the values of :data:`prefix` and :data:`exec_prefix` will be changed to
   point to the virtual environment, whereas :data:`base_prefix` and
   :data:`base_exec_prefix` will remain pointing to the base Python
   installation (the one which the virtual environment was created from).

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 147)

Unknown directive type "data".

```
.. data:: byteorder

   An indicator of the native byte order.  This will have the value ``'big'`` on
   big-endian (most-significant byte first) platforms, and ``'little'`` on
   little-endian (least-significant byte first) platforms.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 154)

Unknown directive type "data".

```
.. data:: builtin_module_names

   A tuple of strings containing the names of all modules that are compiled into this
   Python interpreter.  (This information is not available in any other way ---
   ``modules.keys()`` only lists the imported modules.)

   See also the :attr:`sys.stdlib_module_names` list.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 163)

Unknown directive type "function".

```
.. function:: call_tracing(func, args)

   Call ``func(*args)``, while tracing is enabled.  The tracing state is saved,
   and restored afterwards.  This is intended to be called from a debugger from
   a checkpoint, to recursively debug some other code.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

main\Doc\library\[cpython-main][Doc][library]sys.rst, line 170)

Unknown directive type "data".

```
.. data:: copyright

   A string containing the copyright pertaining to the Python interpreter.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 175)

Unknown directive type "function".

```
.. function:: _clear_type_cache()

   Clear the internal type cache. The type cache is used to speed up attribute
   and method lookups. Use the function *only* to drop unnecessary references
   during reference leak debugging.

   This function should be used for internal and specialized purposes only.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 184)

Unknown directive type "function".

```
.. function:: _current_frames()

   Return a dictionary mapping each thread's identifier to the topmost stack frame
   currently active in that thread at the time the function is called. Note that
   functions in the :mod:`traceback` module can build the call stack given such a
   frame.

   This is most useful for debugging deadlock:  this function does not require the
   deadlocked threads' cooperation, and such threads' call stacks are frozen for as
   long as they remain deadlocked.  The frame returned for a non-deadlocked thread
   may bear no relationship to that thread's current activity by the time calling
   code examines the frame.

   This function should be used for internal and specialized purposes only.

   .. audit-event:: sys._current_frames "" sys._current_frames
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 201)

Unknown directive type "function".

```
.. function:: _current_exceptions()

   Return a dictionary mapping each thread's identifier to the topmost exception
   currently active in that thread at the time the function is called.
   If a thread is not currently handling an exception, it is not included in
   the result dictionary.

   This is most useful for statistical profiling.

   This function should be used for internal and specialized purposes only.

   .. audit-event:: sys._current_exceptions "" sys._current_exceptions
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 214)

Unknown directive type "function".

```
.. function:: breakpointhook()

   This hook function is called by built-in :func:`breakpoint`.  By default,
   it drops you into the :mod:`pdb` debugger, but it can be set to any other
   function so that you can choose which debugger gets used.
```

```
   The signature of this function is dependent on what it calls.  For example,
   the default binding (e.g. ``pdb.set_trace()``) expects no arguments, but
   you might bind it to a function that expects additional arguments
   (positional and/or keyword).  The built-in ``breakpoint()`` function passes
   its ``*args`` and ``**kws`` straight through.  Whatever
   ``breakpointhooks()`` returns is returned from ``breakpoint()``.

   The default implementation first consults the environment variable
   :envvar:`PYTHONBREAKPOINT`.  If that is set to ``"0"`` then this function
   returns immediately; i.e. it is a no-op.  If the environment variable is
   not set, or is set to the empty string, ``pdb.set_trace()`` is called.
   Otherwise this variable should name a function to run, using Python's
   dotted-import nomenclature, e.g. ``package.subpackage.module.function``.
   In this case, ``package.subpackage.module`` would be imported and the
   resulting module must have a callable named ``function()``.  This is run,
   passing in ``*args`` and ``**kws``, and whatever ``function()`` returns,
   ``sys.breakpointhook()`` returns to the built-in :func:`breakpoint`
   function.

   Note that if anything goes wrong while importing the callable named by
   :envvar:`PYTHONBREAKPOINT`, a :exc:`RuntimeWarning` is reported and the
   breakpoint is ignored.

   Also note that if ``sys.breakpointhook()`` is overridden programmatically,
   :envvar:`PYTHONBREAKPOINT` is *not* consulted.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 248)

Unknown directive type "function".

```
.. function:: _debugmallocstats()

   Print low-level information to stderr about the state of CPython's memory
   allocator.

   If Python is `built in debug mode <debug-build>` (:option:`configure
   --with-pydebug option <--with-pydebug>`), it also performs some expensive
   internal consistency checks.

   .. versionadded:: 3.3

   .. impl-detail::

      This function is specific to CPython.  The exact output format is not
      defined here, and may change.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 265)

Unknown directive type "data".

```
.. data:: dllhandle

   Integer specifying the handle of the Python DLL.

   .. availability:: Windows.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 272)

Unknown directive type "function".

```
.. function:: displayhook(value)

   If *value* is not ``None``, this function prints ``repr(value)`` to
   ``sys.stdout``, and saves *value* in ``builtins._``. If ``repr(value)`` is
   not encodable to ``sys.stdout.encoding`` with ``sys.stdout.errors`` error
   handler (which is probably ``'strict'``), encode it to
   ``sys.stdout.encoding`` with ``'backslashreplace'`` error handler.

   ``sys.displayhook`` is called on the result of evaluating an :term:`expression`
```

```
entered in an interactive Python session.  The display of these values can be
customized by assigning another one-argument function to ``sys.displayhook``.

Pseudo-code::

    def displayhook(value):
        if value is None:
            return
        # Set '_' to None to avoid recursion
        builtins._ = None
        text = repr(value)
        try:
            sys.stdout.write(text)
        except UnicodeEncodeError:
            bytes = text.encode(sys.stdout.encoding, 'backslashreplace')
            if hasattr(sys.stdout, 'buffer'):
                sys.stdout.buffer.write(bytes)
            else:
                text = bytes.decode(sys.stdout.encoding, 'strict')
                sys.stdout.write(text)
        sys.stdout.write("\n")
        builtins._ = value

.. versionchanged:: 3.2
   Use ``'backslashreplace'`` error handler on :exc:`UnicodeEncodeError`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 308)

Unknown directive type "data".

```
.. data:: dont_write_bytecode

   If this is true, Python won't try to write ``.pyc`` files on the
   import of source modules.  This value is initially set to ``True`` or
   ``False`` depending on the :option:`-B` command line option and the
   :envvar:`PYTHONDONTWRITEBYTECODE` environment variable, but you can set it
   yourself to control bytecode file generation.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 317)

Unknown directive type "data".

```
.. data:: pycache_prefix

   If this is set (not ``None``), Python will write bytecode-cache ``.pyc``
   files to (and read them from) a parallel directory tree rooted at this
   directory, rather than from ``__pycache__`` directories in the source code
   tree. Any ``__pycache__`` directories in the source code tree will be ignored
   and new `.pyc` files written within the pycache prefix. Thus if you use
   :mod:`compileall` as a pre-build step, you must ensure you run it with the
   same pycache prefix (if any) that you will use at runtime.

   A relative path is interpreted relative to the current working directory.

   This value is initially set based on the value of the :option:`-X`
   ``pycache_prefix=PATH`` command-line option or the
   :envvar:`PYTHONPYCACHEPREFIX` environment variable (command-line takes
   precedence). If neither are set, it is ``None``.

   .. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 337)

Unknown directive type "function".

```
.. function:: excepthook(type, value, traceback)

   This function prints out a given traceback and exception to ``sys.stderr``.

   When an exception is raised and uncaught, the interpreter calls
```

```
   ``sys.excepthook`` with three arguments, the exception class, exception
   instance, and a traceback object.  In an interactive session this happens just
   before control is returned to the prompt; in a Python program this happens just
   before the program exits.  The handling of such top-level exceptions can be
   customized by assigning another three-argument function to ``sys.excepthook``.

   .. audit-event:: sys.excepthook hook,type,value,traceback sys.excepthook

      Raise an auditing event ``sys.excepthook`` with arguments ``hook``,
      ``type``, ``value``, ``traceback`` when an uncaught exception occurs.
      If no hook has been set, ``hook`` may be ``None``. If any hook raises
      an exception derived from :class:`RuntimeError` the call to the hook will
      be suppressed. Otherwise, the audit hook exception will be reported as
      unraisable and ``sys.excepthook`` will be called.

   .. seealso::

      The :func:`sys.unraisablehook` function handles unraisable exceptions
      and the :func:`threading.excepthook` function handles exception raised
      by :func:`threading.Thread.run`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 364)

Unknown directive type "data".

```
.. data:: __breakpointhook__
          __displayhook__
          __excepthook__
          __unraisablehook__

   These objects contain the original values of ``breakpointhook``,
   ``displayhook``, ``excepthook``, and ``unraisablehook`` at the start of the
   program.  They are saved so that ``breakpointhook``, ``displayhook`` and
   ``excepthook``, ``unraisablehook`` can be restored in case they happen to
   get replaced with broken or alternative objects.

   .. versionadded:: 3.7
      __breakpointhook__

   .. versionadded:: 3.8
      __unraisablehook__
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 382)

Unknown directive type "function".

```
.. function:: exception()

   This function returns the exception instance that is currently being
   handled.  This exception is specific both to the current thread and
   to the current stack frame.  If the current stack frame is not handling
   an exception, the exception is taken from the calling stack frame, or its
   caller, and so on until a stack frame is found that is handling an
   exception.  Here, "handling an exception" is defined as "executing an
   except clause." For any stack frame, only the exception being currently
   handled is accessible.

   .. index:: object: traceback

   If no exception is being handled anywhere on the stack, ``None`` is
   returned.

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 401)

Unknown directive type "function".

```
.. function:: exc_info()
```

```
   This function returns the old-style representation of the handled
   exception. If an exception ``e`` is currently handled (so
   :func:`exception` would return ``e``), :func:`exc_info` returns the
   tuple ``(type(e), e, e.__traceback__)``.
   That is, a tuple containing the type of the exception (a subclass of
   :exc:`BaseException`), the exception itself, and a :ref:`traceback
   object <traceback-objects>` which typically encapsulates the call
   stack at the point where the exception last occurred.

   .. index:: object: traceback

   If no exception is being handled anywhere on the stack, this function
   return a tuple containing three ``None`` values.

   .. versionchanged:: 3.11
      The ``type`` and ``traceback`` fields are now derived from the ``value``
      (the exception instance), so when an exception is modified while it is
      being handled, the changes are reflected in the results of subsequent
      calls to :func:`exc_info`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 423)

Unknown directive type "data".

```
.. data:: exec_prefix

   A string giving the site-specific directory prefix where the platform-dependent
   Python files are installed; by default, this is also ``'/usr/local'``.  This can
   be set at build time with the ``--exec-prefix`` argument to the
   :program:`configure` script.  Specifically, all configuration files (e.g. the
   :file:`pyconfig.h` header file) are installed in the directory
   :file:`{exec_prefix}/lib/python{X.Y}/config`, and shared library modules are
   installed in :file:`{exec_prefix}/lib/python{X.Y}/lib-dynload`, where *X.Y*
   is the version number of Python, for example ``3.2``.

   .. note::

      If a :ref:`virtual environment <venv-def>` is in effect, this
      value will be changed in ``site.py`` to point to the virtual environment.
      The value for the Python installation will still be available, via
      :data:`base_exec_prefix`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 442)

Unknown directive type "data".

```
.. data:: executable

   A string giving the absolute path of the executable binary for the Python
   interpreter, on systems where this makes sense. If Python is unable to retrieve
   the real path to its executable, :data:`sys.executable` will be an empty string
   or ``None``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 450)

Unknown directive type "function".

```
.. function:: exit([arg])

   Raise a :exc:`SystemExit` exception, signaling an intention to exit the interpreter.

   The optional argument *arg* can be an integer giving the exit status
   (defaulting to zero), or another type of object.  If it is an integer, zero
   is considered "successful termination" and any nonzero value is considered
   "abnormal termination" by shells and the like.  Most systems require it to be
   in the range 0--127, and produce undefined results otherwise.  Some systems
   have a convention for assigning specific meanings to specific exit codes, but
   these are generally underdeveloped; Unix programs generally use 2 for command
   line syntax errors and 1 for all other kind of errors.  If another type of
   object is passed, ``None`` is equivalent to passing zero, and any other
   object is printed to :data:`stderr` and results in an exit code of 1.  In
```

```
particular, ``sys.exit("some error message")`` is a quick way to exit a
program when an error occurs.

Since :func:`exit` ultimately "only" raises an exception, it will only exit
the process when called from the main thread, and the exception is not
intercepted. Cleanup actions specified by finally clauses of :keyword:`try` statements
are honored, and it is possible to intercept the exit attempt at an outer level.

.. versionchanged:: 3.6
   If an error occurs in the cleanup after the Python interpreter
   has caught :exc:`SystemExit` (such as an error flushing buffered data
   in the standard streams), the exit status is changed to 120.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 478)

Unknown directive type "data".

```
.. data:: flags

   The :term:`named tuple` *flags* exposes the status of command line
   flags. The attributes are read only.

   ============================== ==========================================================================
   attribute                      flag
   ============================== ==========================================================================
   :const:`debug`                 :option:`-d`
   :const:`inspect`               :option:`-i`
   :const:`interactive`           :option:`-i`
   :const:`isolated`              :option:`-I`
   :const:`optimize`              :option:`-O` or :option:`-OO`
   :const:`dont_write_bytecode`   :option:`-B`
   :const:`no_user_site`          :option:`-s`
   :const:`no_site`               :option:`-S`
   :const:`ignore_environment`    :option:`-E`
   :const:`verbose`               :option:`-v`
   :const:`bytes_warning`         :option:`-b`
   :const:`quiet`                 :option:`-q`
   :const:`hash_randomization`    :option:`-R`
   :const:`dev_mode`              :option:`-X dev <-X>` (:ref:`Python Development Mode <devmode>`)
   :const:`utf8_mode`             :option:`-X utf8 <-X>`
   ============================== ==========================================================================

   .. versionchanged:: 3.2
      Added ``quiet`` attribute for the new :option:`-q` flag.

   .. versionadded:: 3.2.3
      The ``hash_randomization`` attribute.

   .. versionchanged:: 3.3
      Removed obsolete ``division_warning`` attribute.

   .. versionchanged:: 3.4
      Added ``isolated`` attribute for :option:`-I` ``isolated`` flag.

   .. versionchanged:: 3.7
      Added the ``dev_mode`` attribute for the new :ref:`Python Development
      Mode <devmode>` and the ``utf8_mode`` attribute for the new  :option:`-X`
      ``utf8`` flag.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 521)

Unknown directive type "data".

```
.. data:: float_info

   A :term:`named tuple` holding information about the float type. It
   contains low level information about the precision and internal
   representation.  The values correspond to the various floating-point
   constants defined in the standard header file :file:`float.h` for the 'C'
   programming language; see section 5.2.4.2.2 of the 1999 ISO/IEC C standard
   [C99]_, 'Characteristics of floating types', for details.

   .. tabularcolumns:: |l|l|L|
```

```
+---------------------+----------------+--------------------------------------------------+
| attribute           | float.h macro  | explanation                                      |
+=====================+================+==================================================+
| :const:`epsilon`    | DBL_EPSILON    | difference between 1.0 and the least value       |
|                     |                | greater than 1.0 that is representable as a float|
|                     |                |                                                  |
|                     |                | See also :func:`math.ulp`.                       |
+---------------------+----------------+--------------------------------------------------+
| :const:`dig`        | DBL_DIG        | maximum number of decimal digits that can be     |
|                     |                | faithfully represented in a float;  see below    |
+---------------------+----------------+--------------------------------------------------+
| :const:`mant_dig`   | DBL_MANT_DIG   | float precision: the number of base-``radix``    |
|                     |                | digits in the significand of a float             |
+---------------------+----------------+--------------------------------------------------+
| :const:`max`        | DBL_MAX        | maximum representable positive finite float      |
+---------------------+----------------+--------------------------------------------------+
| :const:`max_exp`    | DBL_MAX_EXP    | maximum integer *e* such that ``radix**(e-1)`` is|
|                     |                | a representable finite float                     |
+---------------------+----------------+--------------------------------------------------+
| :const:`max_10_exp` | DBL_MAX_10_EXP | maximum integer *e* such that ``10**e`` is in the|
|                     |                | range of representable finite floats             |
+---------------------+----------------+--------------------------------------------------+
| :const:`min`        | DBL_MIN        | minimum representable positive *normalized* float|
|                     |                |                                                  |
|                     |                | Use :func:`math.ulp(0.0) <math.ulp>` to get the  |
|                     |                | smallest positive *denormalized* representable   |
|                     |                | float.                                           |
+---------------------+----------------+--------------------------------------------------+
| :const:`min_exp`    | DBL_MIN_EXP    | minimum integer *e* such that ``radix**(e-1)`` is|
|                     |                | a normalized float                               |
+---------------------+----------------+--------------------------------------------------+
| :const:`min_10_exp` | DBL_MIN_10_EXP | minimum integer *e* such that ``10**e`` is a     |
|                     |                | normalized float                                 |
+---------------------+----------------+--------------------------------------------------+
| :const:`radix`      | FLT_RADIX      | radix of exponent representation                 |
+---------------------+----------------+--------------------------------------------------+
| :const:`rounds`     | FLT_ROUNDS     | integer constant representing the rounding mode  |
|                     |                | used for arithmetic operations.  This reflects   |
|                     |                | the value of the system FLT_ROUNDS macro at      |
|                     |                | interpreter startup time.  See section 5.2.4.2.2 |
|                     |                | of the C99 standard for an explanation of the    |
|                     |                | possible values and their meanings.              |
+---------------------+----------------+--------------------------------------------------+

The attribute :attr:`sys.float_info.dig` needs further explanation.  If
``s`` is any string representing a decimal number with at most
:attr:`sys.float_info.dig` significant digits, then converting ``s`` to a
float and back again will recover a string representing the same decimal
value::

   >>> import sys
   >>> sys.float_info.dig
   15
   >>> s = '3.14159265358979'    # decimal string with 15 significant digits
   >>> format(float(s), '.15g')  # convert to float and back -> same value
   '3.14159265358979'

But for strings with more than :attr:`sys.float_info.dig` significant digits,
this isn't always true::

   >>> s = '9876543211234567'    # 16 significant digits is too many!
   >>> format(float(s), '.16g')  # conversion changes value
   '9876543211234568'
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 596)

Unknown directive type "data".

```
.. data:: float_repr_style

   A string indicating how the :func:`repr` function behaves for
   floats.  If the string has value ``'short'`` then for a finite
   float ``x``, ``repr(x)`` aims to produce a short string with the
   property that ``float(repr(x)) == x``.  This is the usual behaviour
   in Python 3.1 and later.  Otherwise, ``float_repr_style`` has value
   ``'legacy'`` and ``repr(x)`` behaves in the same way as it did in
   versions of Python prior to 3.1.

   .. versionadded:: 3.1
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 609)

Unknown directive type "function".

```
.. function:: getallocatedblocks()

   Return the number of memory blocks currently allocated by the interpreter,
   regardless of their size.  This function is mainly useful for tracking
   and debugging memory leaks.  Because of the interpreter's internal
   caches, the result can vary from call to call; you may have to call
   :func:`_clear_type_cache()` and :func:`gc.collect()` to get more
   predictable results.

   If a Python build or implementation cannot reasonably compute this
   information, :func:`getallocatedblocks()` is allowed to return 0 instead.

   .. versionadded:: 3.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 624)

Unknown directive type "function".

```
.. function:: getandroidapilevel()

   Return the build time API version of Android as an integer.

   .. availability:: Android.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 633)

Unknown directive type "function".

```
.. function:: getdefaultencoding()

   Return the name of the current default string encoding used by the Unicode
   implementation.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 639)

Unknown directive type "function".

```
.. function:: getdlopenflags()

   Return the current value of the flags that are used for
   :c:func:`dlopen` calls.  Symbolic names for the flag values can be
   found in the :mod:`os` module (``RTLD_xxx`` constants, e.g.
   :data:`os.RTLD_LAZY`).

   .. availability:: Unix.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 649)

Unknown directive type "function".

```
.. function:: getfilesystemencoding()

   Get the :term:`filesystem encoding <filesystem encoding and error handler>`:
   the encoding used with the :term:`filesystem error handler <filesystem
   encoding and error handler>` to convert between Unicode filenames and bytes
```

```
   filenames. The filesystem error handler is returned from
   :func:`getfilesystemencoding`.

   For best compatibility, str should be used for filenames in all cases,
   although representing filenames as bytes is also supported. Functions
   accepting or returning filenames should support either str or bytes and
   internally convert to the system's preferred representation.

   :func:`os.fsencode` and :func:`os.fsdecode` should be used to ensure that
   the correct encoding and errors mode are used.

   The :term:`filesystem encoding and error handler` are configured at Python
   startup by the :c:func:`PyConfig_Read` function: see
   :c:member:`~PyConfig.filesystem_encoding` and
   :c:member:`~PyConfig.filesystem_errors` members of :c:type:`PyConfig`.

   .. versionchanged:: 3.2
      :func:`getfilesystemencoding` result cannot be ``None`` anymore.

   .. versionchanged:: 3.6
      Windows is no longer guaranteed to return ``'mbcs'``. See :pep:`529`
      and :func:`_enablelegacywindowsfsencoding` for more information.

   .. versionchanged:: 3.7
      Return ``'utf-8'`` if the :ref:`Python UTF-8 Mode <utf8-mode>` is
      enabled.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 682)

Unknown directive type "function".

```
.. function:: getfilesystemencodeerrors()

   Get the :term:`filesystem error handler <filesystem encoding and error
   handler>`: the error handler used with the :term:`filesystem encoding
   <filesystem encoding and error handler>` to convert between Unicode
   filenames and bytes filenames. The filesystem encoding is returned from
   :func:`getfilesystemencoding`.

   :func:`os.fsencode` and :func:`os.fsdecode` should be used to ensure that
   the correct encoding and errors mode are used.

   The :term:`filesystem encoding and error handler` are configured at Python
   startup by the :c:func:`PyConfig_Read` function: see
   :c:member:`~PyConfig.filesystem_encoding` and
   :c:member:`~PyConfig.filesystem_errors` members of :c:type:`PyConfig`.

   .. versionadded:: 3.6
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 700)

Unknown directive type "function".

```
.. function:: getrefcount(object)

   Return the reference count of the *object*.  The count returned is generally one
   higher than you might expect, because it includes the (temporary) reference as
   an argument to :func:`getrefcount`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 707)

Unknown directive type "function".

```
.. function:: getrecursionlimit()

   Return the current value of the recursion limit, the maximum depth of the Python
   interpreter stack.  This limit prevents infinite recursion from causing an
   overflow of the C stack and crashing Python.  It can be set by
   :func:`setrecursionlimit`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 715)

Unknown directive type "function".

```
.. function:: getsizeof(object[, default])

   Return the size of an object in bytes. The object can be any type of
   object. All built-in objects will return correct results, but this
   does not have to hold true for third-party extensions as it is implementation
   specific.

   Only the memory consumption directly attributed to the object is
   accounted for, not the memory consumption of objects it refers to.

   If given, *default* will be returned if the object does not provide means to
   retrieve the size.  Otherwise a :exc:`TypeError` will be raised.

   :func:`getsizeof` calls the object's ``__sizeof__`` method and adds an
   additional garbage collector overhead if the object is managed by the garbage
   collector.

   See `recursive sizeof recipe <https://code.activestate.com/recipes/577504>`_
   for an example of using :func:`getsizeof` recursively to find the size of
   containers and all their contents.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 736)

Unknown directive type "function".

```
.. function:: getswitchinterval()

   Return the interpreter's "thread switch interval"; see
   :func:`setswitchinterval`.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 744)

Unknown directive type "function".

```
.. function:: _getframe([depth])

   Return a frame object from the call stack.  If optional integer *depth* is
   given, return the frame object that many calls below the top of the stack.  If
   that is deeper than the call stack, :exc:`ValueError` is raised.  The default
   for *depth* is zero, returning the frame at the top of the call stack.

   .. audit-event:: sys._getframe "" sys._getframe

   .. impl-detail::

      This function should be used for internal and specialized purposes only.
      It is not guaranteed to exist in all implementations of Python.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 759)

Unknown directive type "function".

```
.. function:: getprofile()

   .. index::
      single: profile function
      single: profiler

   Get the profiler function as set by :func:`setprofile`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 768)

Unknown directive type "function".

```
.. function:: gettrace()

   .. index::
      single: trace function
      single: debugger

   Get the trace function as set by :func:`settrace`.

   .. impl-detail::

      The :func:`gettrace` function is intended only for implementing debuggers,
      profilers, coverage tools and the like.  Its behavior is part of the
      implementation platform, rather than part of the language definition, and
      thus may not be available in all Python implementations.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 784)

Unknown directive type "function".

```
.. function:: getwindowsversion()

   Return a named tuple describing the Windows version
   currently running.  The named elements are *major*, *minor*,
   *build*, *platform*, *service_pack*, *service_pack_minor*,
   *service_pack_major*, *suite_mask*, *product_type* and
   *platform_version*. *service_pack* contains a string,
   *platform_version* a 3-tuple and all other values are
   integers. The components can also be accessed by name, so
   ``sys.getwindowsversion()[0]`` is equivalent to
   ``sys.getwindowsversion().major``. For compatibility with prior
   versions, only the first 5 elements are retrievable by indexing.

   *platform* will be :const:`2 (VER_PLATFORM_WIN32_NT)`.

   *product_type* may be one of the following values:

   +---------------------------------------+---------------------------------+
   | Constant                              | Meaning                         |
   +=======================================+=================================+
   | :const:`1 (VER_NT_WORKSTATION)`       | The system is a workstation.    |
   +---------------------------------------+---------------------------------+
   | :const:`2 (VER_NT_DOMAIN_CONTROLLER)` | The system is a domain          |
   |                                       | controller.                     |
   +---------------------------------------+---------------------------------+
   | :const:`3 (VER_NT_SERVER)`            | The system is a server, but not |
   |                                       | a domain controller.            |
   +---------------------------------------+---------------------------------+

   This function wraps the Win32 :c:func:`GetVersionEx` function; see the
   Microsoft documentation on :c:func:`OSVERSIONINFOEX` for more information
   about these fields.

   *platform_version* returns the major version, minor version and
   build number of the current operating system, rather than the version that
   is being emulated for the process. It is intended for use in logging rather
   than for feature detection.

   .. note::
      *platform_version* derives the version from kernel32.dll which can be of a different
      version than the OS version. Please use :mod:`platform` module for achieving accurate
      OS version.

   .. availability:: Windows.

   .. versionchanged:: 3.2
      Changed to a named tuple and added *service_pack_minor*,
      *service_pack_major*, *suite_mask*, and *product_type*.

   .. versionchanged:: 3.6
      Added *platform_version*
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 837)

Unknown directive type "function".

```
.. function:: get_asyncgen_hooks()

   Returns an *asyncgen_hooks* object, which is similar to a
   :class:`~collections.namedtuple` of the form `(firstiter, finalizer)`,
   where *firstiter* and *finalizer* are expected to be either ``None`` or
   functions which take an :term:`asynchronous generator iterator` as an
   argument, and are used to schedule finalization of an asynchronous
   generator by an event loop.

   .. versionadded:: 3.6
      See :pep:`525` for more details.

   .. note::
      This function has been added on a provisional basis (see :pep:`411`
      for details.)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 854)

Unknown directive type "function".

```
.. function:: get_coroutine_origin_tracking_depth()

   Get the current coroutine origin tracking depth, as set by
   :func:`set_coroutine_origin_tracking_depth`.

   .. versionadded:: 3.7

   .. note::
      This function has been added on a provisional basis (see :pep:`411`
      for details.)  Use it only for debugging purposes.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 866)

Unknown directive type "data".

```
.. data:: hash_info

   A :term:`named tuple` giving parameters of the numeric hash
   implementation.  For more details about hashing of numeric types, see
   :ref:`numeric-hash`.

   +---------------------+---------------------------------------------------+
   | attribute           | explanation                                       |
   +=====================+===================================================+
   | :const:`width`      | width in bits used for hash values                |
   +---------------------+---------------------------------------------------+
   | :const:`modulus`    | prime modulus P used for numeric hash scheme      |
   +---------------------+---------------------------------------------------+
   | :const:`inf`        | hash value returned for a positive infinity       |
   +---------------------+---------------------------------------------------+
   | :const:`nan`        | (this attribute is no longer used)                |
   +---------------------+---------------------------------------------------+
   | :const:`imag`       | multiplier used for the imaginary part of a       |
   |                     | complex number                                    |
   +---------------------+---------------------------------------------------+
   | :const:`algorithm`  | name of the algorithm for hashing of str, bytes,  |
   |                     | and memoryview                                    |
   +---------------------+---------------------------------------------------+
   | :const:`hash_bits`  | internal output size of the hash algorithm        |
   +---------------------+---------------------------------------------------+
   | :const:`seed_bits`  | size of the seed key of the hash algorithm        |
   +---------------------+---------------------------------------------------+


   .. versionadded:: 3.2

   .. versionchanged:: 3.4
      Added *algorithm*, *hash_bits* and *seed_bits*
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 901)

Unknown directive type "data".

```
.. data:: hexversion

   The version number encoded as a single integer.  This is guaranteed to increase
   with each version, including proper support for non-production releases.  For
   example, to test that the Python interpreter is at least version 1.5.2, use::

      if sys.hexversion >= 0x010502F0:
          # use some advanced feature
          ...
      else:
          # use an alternative implementation or warn the user
          ...

   This is called ``hexversion`` since it only really looks meaningful when viewed
   as the result of passing it to the built-in :func:`hex` function.  The
   :term:`named tuple`  :data:`sys.version_info` may be used for a more
   human-friendly encoding of the same information.

   More details of ``hexversion`` can be found at :ref:`apiabiversion`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 922)

Unknown directive type "data".

```
.. data:: implementation

   An object containing information about the implementation of the
   currently running Python interpreter.  The following attributes are
   required to exist in all Python implementations.

   *name* is the implementation's identifier, e.g. ``'cpython'``.  The actual
   string is defined by the Python implementation, but it is guaranteed to be
   lower case.

   *version* is a named tuple, in the same format as
   :data:`sys.version_info`.  It represents the version of the Python
   *implementation*.  This has a distinct meaning from the specific
   version of the Python *language* to which the currently running
   interpreter conforms, which ``sys.version_info`` represents.  For
   example, for PyPy 1.8 ``sys.implementation.version`` might be
   ``sys.version_info(1, 8, 0, 'final', 0)``, whereas ``sys.version_info``
   would be ``sys.version_info(2, 7, 2, 'final', 0)``.  For CPython they
   are the same value, since it is the reference implementation.

   *hexversion* is the implementation version in hexadecimal format, like
   :data:`sys.hexversion`.

   *cache_tag* is the tag used by the import machinery in the filenames of
   cached modules.  By convention, it would be a composite of the
   implementation's name and version, like ``'cpython-33'``.  However, a
   Python implementation may use some other value if appropriate.  If
   ``cache_tag`` is set to ``None``, it indicates that module caching should
   be disabled.

   :data:`sys.implementation` may contain additional attributes specific to
   the Python implementation.  These non-standard attributes must start with
   an underscore, and are not described here.  Regardless of its contents,
   :data:`sys.implementation` will not change during a run of the interpreter,
   nor between implementation versions.  (It may change between Python
   language versions, however.)  See :pep:`421` for more information.

   .. versionadded:: 3.3

   .. note::

      The addition of new required attributes must go through the normal PEP
      process. See :pep:`421` for more information.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 966)

Unknown directive type "data".

```
.. data:: int_info

   A :term:`named tuple` that holds information about Python's internal
   representation of integers.  The attributes are read only.

   .. tabularcolumns:: |l|L|

   +-------------------------+-----------------------------------------+
   | Attribute               | Explanation                             |
   +=========================+=========================================+
   | :const:`bits_per_digit` | number of bits held in each digit.  Python   |
   |                         | integers are stored internally in base       |
   |                         | ``2**int_info.bits_per_digit``               |
   +-------------------------+-----------------------------------------+
   | :const:`sizeof_digit`   | size in bytes of the C type used to          |
   |                         | represent a digit                       |
   +-------------------------+-----------------------------------------+

   .. versionadded:: 3.1
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 987)

Unknown directive type "data".

```
.. data:: __interactivehook__

   When this attribute exists, its value is automatically called (with no
   arguments) when the interpreter is launched in :ref:`interactive mode
   <tut-interactive>`.  This is done after the :envvar:`PYTHONSTARTUP` file is
   read, so that you can set this hook there.  The :mod:`site` module
   :ref:`sets this <rlcompleter-config>`.

   .. audit-event:: cpython.run_interactivehook hook sys.__interactivehook__

      Raises an :ref:`auditing event <auditing>`
      ``cpython.run_interactivehook`` with the hook object as the argument when
      the hook is called on startup.

   .. versionadded:: 3.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1004)

Unknown directive type "function".

```
.. function:: intern(string)

   Enter *string* in the table of "interned" strings and return the interned string
   -- which is *string* itself or a copy. Interning strings is useful to gain a
   little performance on dictionary lookup -- if the keys in a dictionary are
   interned, and the lookup key is interned, the key comparisons (after hashing)
   can be done by a pointer compare instead of a string compare.  Normally, the
   names used in Python programs are automatically interned, and the dictionaries
   used to hold module, class or instance attributes have interned keys.

   Interned strings are not immortal; you must keep a reference to the return
   value of :func:`intern` around to benefit from it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1018)

Unknown directive type "function".

```
.. function:: is_finalizing()

   Return :const:`True` if the Python interpreter is
```

```
   :term:`shutting down <interpreter shutdown>`, :const:`False` otherwise.

   .. versionadded:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1026)

Unknown directive type "data".

```
.. data:: last_type
          last_value
          last_traceback

   These three variables are not always defined; they are set when an exception is
   not handled and the interpreter prints an error message and a stack traceback.
   Their intended use is to allow an interactive user to import a debugger module
   and engage in post-mortem debugging without having to re-execute the command
   that caused the error.  (Typical use is ``import pdb; pdb.pm()`` to enter the
   post-mortem debugger; see :mod:`pdb` module for
   more information.)

   The meaning of the variables is the same as that of the return values from
   :func:`exc_info` above.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1042)

Unknown directive type "data".

```
.. data:: maxsize

   An integer giving the maximum value a variable of type :c:type:`Py_ssize_t` can
   take.  It's usually ``2**31 - 1`` on a 32-bit platform and ``2**63 - 1`` on a
   64-bit platform.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1049)

Unknown directive type "data".

```
.. data:: maxunicode

   An integer giving the value of the largest Unicode code point,
   i.e. ``1114111`` (``0x10FFFF`` in hexadecimal).

   .. versionchanged:: 3.3
      Before :pep:`393`, ``sys.maxunicode`` used to be either ``0xFFFF``
      or ``0x10FFFF``, depending on the configuration option that specified
      whether Unicode characters were stored as UCS-2 or UCS-4.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1060)

Unknown directive type "data".

```
.. data:: meta_path

    A list of :term:`meta path finder` objects that have their
    :meth:`~importlib.abc.MetaPathFinder.find_spec` methods called to see if one
    of the objects can find the module to be imported. The
    :meth:`~importlib.abc.MetaPathFinder.find_spec` method is called with at
    least the absolute name of the module being imported. If the module to be
    imported is contained in a package, then the parent package's :attr:`__path__`
    attribute is passed in as a second argument. The method returns a
    :term:`module spec`, or ``None`` if the module cannot be found.

    .. seealso::

        :class:`importlib.abc.MetaPathFinder`
          The abstract base class defining the interface of finder objects on
```

```
          :data:`meta_path`.
        :class:`importlib.machinery.ModuleSpec`
          The concrete class which
          :meth:`~importlib.abc.MetaPathFinder.find_spec` should return
          instances of.

    .. versionchanged:: 3.4

        :term:`Module specs <module spec>` were introduced in Python 3.4, by
        :pep:`451`. Earlier versions of Python looked for a method called
        :meth:`~importlib.abc.MetaPathFinder.find_module`.
        This is still called as a fallback if a :data:`meta_path` entry doesn't
        have a :meth:`~importlib.abc.MetaPathFinder.find_spec` method.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1089)

Unknown directive type "data".

```
.. data:: modules

   This is a dictionary that maps module names to modules which have already been
   loaded.  This can be manipulated to force reloading of modules and other tricks.
   However, replacing the dictionary will not necessarily work as expected and
   deleting essential items from the dictionary may cause Python to fail.  If
   you want to iterate over this global dictionary always use
   ``sys.modules.copy()`` or ``tuple(sys.modules)`` to avoid exceptions as its
   size may change during iteration as a side effect of code or activity in
   other threads.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1101)

Unknown directive type "data".

```
.. data:: orig_argv

   The list of the original command line arguments passed to the Python
   executable.

   See also :data:`sys.argv`.

   .. versionadded:: 3.10
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1111)

Unknown directive type "data".

```
.. data:: path

   .. index:: triple: module; search; path

   A list of strings that specifies the search path for modules. Initialized from
   the environment variable :envvar:`PYTHONPATH`, plus an installation-dependent
   default.

   As initialized upon program startup, the first item of this list, ``path[0]``,
   is the directory containing the script that was used to invoke the Python
   interpreter.  If the script directory is not available (e.g.  if the interpreter
   is invoked interactively or if the script is read from standard input),
   ``path[0]`` is the empty string, which directs Python to search modules in the
   current directory first.  Notice that the script directory is inserted *before*
   the entries inserted as a result of :envvar:`PYTHONPATH`.

   The initialization of :data:`sys.path` is documented at :ref:`sys-path-init`.

   A program is free to modify this list for its own purposes.  Only strings
   and bytes should be added to :data:`sys.path`; all other data types are
   ignored during import.


   .. seealso::
      * Module :mod:`site` This describes how to use .pth files to
```

```
        extend :data:`sys.path`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1138)

Unknown directive type "data".

```
.. data:: path_hooks

   A list of callables that take a path argument to try to create a
   :term:`finder` for the path. If a finder can be created, it is to be
   returned by the callable, else raise :exc:`ImportError`.

   Originally specified in :pep:`302`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1147)

Unknown directive type "data".

```
.. data:: path_importer_cache

   A dictionary acting as a cache for :term:`finder` objects. The keys are
   paths that have been passed to :data:`sys.path_hooks` and the values are
   the finders that are found. If a path is a valid file system path but no
   finder is found on :data:`sys.path_hooks` then ``None`` is
   stored.

   Originally specified in :pep:`302`.

   .. versionchanged:: 3.3
      ``None`` is stored instead of :class:`imp.NullImporter` when no finder
      is found.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1162)

Unknown directive type "data".

```
.. data:: platform

   This string contains a platform identifier that can be used to append
   platform-specific components to :data:`sys.path`, for instance.

   For Unix systems, except on Linux and AIX, this is the lowercased OS name as
   returned by ``uname -s`` with the first part of the version as returned by
   ``uname -r`` appended, e.g. ``'sunos5'`` or ``'freebsd8'``, *at the time
   when Python was built*.  Unless you want to test for a specific system
   version, it is therefore recommended to use the following idiom::

      if sys.platform.startswith('freebsd'):
          # FreeBSD-specific code here...
      elif sys.platform.startswith('linux'):
          # Linux-specific code here...
      elif sys.platform.startswith('aix'):
          # AIX-specific code here...

   For other systems, the values are:

   ================ ===========================
   System           ``platform`` value
   ================ ===========================
   AIX              ``'aix'``
   Linux            ``'linux'``
   Windows          ``'win32'``
   Windows/Cygwin   ``'cygwin'``
   macOS            ``'darwin'``
   ================ ===========================

   .. versionchanged:: 3.3
      On Linux, :attr:`sys.platform` doesn't contain the major version anymore.
      It is always ``'linux'``, instead of ``'linux2'`` or ``'linux3'``.  Since
      older Python versions include the version number, it is recommended to
      always use the ``startswith`` idiom presented above.
```

```
   .. versionchanged:: 3.8
      On AIX, :attr:`sys.platform` doesn't contain the major version anymore.
      It is always ``'aix'``, instead of ``'aix5'`` or ``'aix7'``.  Since
      older Python versions include the version number, it is recommended to
      always use the ``startswith`` idiom presented above.

   .. seealso::

      :attr:`os.name` has a coarser granularity.  :func:`os.uname` gives
      system-dependent version information.

      The :mod:`platform` module provides detailed checks for the
      system's identity.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1213)

Unknown directive type "data".

```
.. data:: platlibdir

   Name of the platform-specific library directory. It is used to build the
   path of standard library and the paths of installed extension modules.

   It is equal to ``"lib"`` on most platforms. On Fedora and SuSE, it is equal
   to ``"lib64"`` on 64-bit platforms which gives the following ``sys.path``
   paths (where ``X.Y`` is the Python ``major.minor`` version):

   * ``/usr/lib64/pythonX.Y/``:
     Standard library (like ``os.py`` of the :mod:`os` module)
   * ``/usr/lib64/pythonX.Y/lib-dynload/``:
     C extension modules of the standard library (like the :mod:`errno` module,
     the exact filename is platform specific)
   * ``/usr/lib/pythonX.Y/site-packages/`` (always use ``lib``, not
     :data:`sys.platlibdir`): Third-party modules
   * ``/usr/lib64/pythonX.Y/site-packages/``:
     C extension modules of third-party packages

   .. versionadded:: 3.9
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1235)

Unknown directive type "data".

```
.. data:: prefix

   A string giving the site-specific directory prefix where the platform
   independent Python files are installed; on Unix, the default is
   ``'/usr/local'``.  This can be set at build time with the ``--prefix``
   argument to the :program:`configure` script.  See
   :ref:`installation_paths` for derived paths.

   .. note:: If a :ref:`virtual environment <venv-def>` is in effect, this
      value will be changed in ``site.py`` to point to the virtual
      environment. The value for the Python installation will still be
      available, via :data:`base_prefix`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1249)

Unknown directive type "data".

```
.. data:: ps1
          ps2

   .. index::
      single: interpreter prompts
      single: prompts, interpreter
      single: >>>; interpreter prompt
      single: ...; interpreter prompt
```

```
   Strings specifying the primary and secondary prompt of the interpreter.  These
   are only defined if the interpreter is in interactive mode.  Their initial
   values in this case are ``'>>> '`` and ``'... '``.  If a non-string object is
   assigned to either variable, its :func:`str` is re-evaluated each time the
   interpreter prepares to read a new interactive command; this can be used to
   implement a dynamic prompt.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1266)

Unknown directive type "function".

```
.. function:: setdlopenflags(n)

   Set the flags used by the interpreter for :c:func:`dlopen` calls, such as when
   the interpreter loads extension modules.  Among other things, this will enable a
   lazy resolving of symbols when importing a module, if called as
   ``sys.setdlopenflags(0)``.  To share symbols across extension modules, call as
   ``sys.setdlopenflags(os.RTLD_GLOBAL)``.  Symbolic names for the flag values
   can be found in the :mod:`os` module (``RTLD_xxx`` constants, e.g.
   :data:`os.RTLD_LAZY`).

   .. availability:: Unix.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1278)

Unknown directive type "function".

```
.. function:: setprofile(profilefunc)

   .. index::
      single: profile function
      single: profiler

   Set the system's profile function, which allows you to implement a Python source
   code profiler in Python.  See chapter :ref:`profile` for more information on the
   Python profiler.  The system's profile function is called similarly to the
   system's trace function (see :func:`settrace`), but it is called with different events,
   for example it isn't called for each executed line of code (only on call and return,
   but the return event is reported even when an exception has been set). The function is
   thread-specific, but there is no way for the profiler to know about context switches between
   threads, so it does not make sense to use this in the presence of multiple threads. Also,
   its return value is not used, so it can simply return ``None``.  Error in the profile
   function will cause itself unset.

   Profile functions should have three arguments: *frame*, *event*, and
   *arg*. *frame* is the current stack frame.  *event* is a string: ``'call'``,
   ``'return'``, ``'c_call'``, ``'c_return'``, or ``'c_exception'``. *arg* depends
   on the event type.

   .. audit-event:: sys.setprofile "" sys.setprofile

   The events have the following meaning:

   ``'call'``
      A function is called (or some other code block entered).  The
      profile function is called; *arg* is ``None``.

   ``'return'``
      A function (or other code block) is about to return.  The profile
      function is called; *arg* is the value that will be returned, or ``None``
      if the event is caused by an exception being raised.

   ``'c_call'``
      A C function is about to be called.  This may be an extension function or
      a built-in.  *arg* is the C function object.

   ``'c_return'``
      A C function has returned. *arg* is the C function object.

   ``'c_exception'``
      A C function has raised an exception.  *arg* is the C function object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1323)

Unknown directive type "function".

```
.. function:: setrecursionlimit(limit)

   Set the maximum depth of the Python interpreter stack to *limit*.  This limit
   prevents infinite recursion from causing an overflow of the C stack and crashing
   Python.

   The highest possible limit is platform-dependent.  A user may need to set the
   limit higher when they have a program that requires deep recursion and a platform
   that supports a higher limit.  This should be done with care, because a too-high
   limit can lead to a crash.

   If the new limit is too low at the current recursion depth, a
   :exc:`RecursionError` exception is raised.

   .. versionchanged:: 3.5.1
      A :exc:`RecursionError` exception is now raised if the new limit is too
      low at the current recursion depth.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1342)

Unknown directive type "function".

```
.. function:: setswitchinterval(interval)

   Set the interpreter's thread switch interval (in seconds).  This floating-point
   value determines the ideal duration of the "timeslices" allocated to
   concurrently running Python threads.  Please note that the actual value
   can be higher, especially if long-running internal functions or methods
   are used.  Also, which thread becomes scheduled at the end of the interval
   is the operating system's decision.  The interpreter doesn't have its
   own scheduler.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1355)

Unknown directive type "function".

```
.. function:: settrace(tracefunc)

   .. index::
      single: trace function
      single: debugger

   Set the system's trace function, which allows you to implement a Python
   source code debugger in Python.  The function is thread-specific; for a
   debugger to support multiple threads, it must register a trace function using
   :func:`settrace` for each thread being debugged or use :func:`threading.settrace`.

   Trace functions should have three arguments: *frame*, *event*, and
   *arg*. *frame* is the current stack frame.  *event* is a string: ``'call'``,
   ``'line'``, ``'return'``, ``'exception'`` or ``'opcode'``.  *arg* depends on
   the event type.

   The trace function is invoked (with *event* set to ``'call'``) whenever a new
   local scope is entered; it should return a reference to a local trace
   function to be used for the new scope, or ``None`` if the scope shouldn't be
   traced.

   The local trace function should return a reference to itself (or to another
   function for further tracing in that scope), or ``None`` to turn off tracing
   in that scope.

   If there is any error occurred in the trace function, it will be unset, just
   like ``settrace(None)`` is called.

   The events have the following meaning:
```

```
   ``'call'``
      A function is called (or some other code block entered).  The
      global trace function is called; *arg* is ``None``; the return value
      specifies the local trace function.

   ``'line'``
      The interpreter is about to execute a new line of code or re-execute the
      condition of a loop.  The local trace function is called; *arg* is
      ``None``; the return value specifies the new local trace function.  See
      :file:`Objects/lnotab_notes.txt` for a detailed explanation of how this
      works.
      Per-line events may be disabled for a frame by setting
      :attr:`f_trace_lines` to :const:`False` on that frame.

   ``'return'``
      A function (or other code block) is about to return.  The local trace
      function is called; *arg* is the value that will be returned, or ``None``
      if the event is caused by an exception being raised.  The trace function's
      return value is ignored.

   ``'exception'``
      An exception has occurred.  The local trace function is called; *arg* is a
      tuple ``(exception, value, traceback)``; the return value specifies the
      new local trace function.

   ``'opcode'``
      The interpreter is about to execute a new opcode (see :mod:`dis` for
      opcode details).  The local trace function is called; *arg* is
      ``None``; the return value specifies the new local trace function.
      Per-opcode events are not emitted by default: they must be explicitly
      requested by setting :attr:`f_trace_opcodes` to :const:`True` on the
      frame.

   Note that as an exception is propagated down the chain of callers, an
   ``'exception'`` event is generated at each level.

   For more fine-grained usage, it's possible to set a trace function by
   assigning ``frame.f_trace = tracefunc`` explicitly, rather than relying on
   it being set indirectly via the return value from an already installed
   trace function. This is also required for activating the trace function on
   the current frame, which :func:`settrace` doesn't do. Note that in order
   for this to work, a global tracing function must have been installed
   with :func:`settrace` in order to enable the runtime tracing machinery,
   but it doesn't need to be the same tracing function (e.g. it could be a
   low overhead tracing function that simply returns ``None`` to disable
   itself immediately on each frame).

   For more information on code and frame objects, refer to :ref:`types`.

   .. audit-event:: sys.settrace "" sys.settrace

   .. impl-detail::

      The :func:`settrace` function is intended only for implementing debuggers,
      profilers, coverage tools and the like.  Its behavior is part of the
      implementation platform, rather than part of the language definition, and
      thus may not be available in all Python implementations.

   .. versionchanged:: 3.7

      ``'opcode'`` event type added; :attr:`f_trace_lines` and
      :attr:`f_trace_opcodes` attributes added to frames
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1448)

Unknown directive type "function".

```
.. function:: set_asyncgen_hooks(firstiter, finalizer)

   Accepts two optional keyword arguments which are callables that accept an
   :term:`asynchronous generator iterator` as an argument. The *firstiter*
   callable will be called when an asynchronous generator is iterated for the
   first time. The *finalizer* will be called when an asynchronous generator
   is about to be garbage collected.

   .. audit-event:: sys.set_asyncgen_hooks_firstiter "" sys.set_asyncgen_hooks

   .. audit-event:: sys.set_asyncgen_hooks_finalizer "" sys.set_asyncgen_hooks
```

```
   Two auditing events are raised because the underlying API consists of two
   calls, each of which must raise its own event.

   .. versionadded:: 3.6
      See :pep:`525` for more details, and for a reference example of a
      *finalizer* method see the implementation of
      ``asyncio.Loop.shutdown_asyncgens`` in
      :source:`Lib/asyncio/base_events.py`

   .. note::
      This function has been added on a provisional basis (see :pep:`411`
      for details.)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1473)

Unknown directive type "function".

```
.. function:: set_coroutine_origin_tracking_depth(depth)

   Allows enabling or disabling coroutine origin tracking. When
   enabled, the ``cr_origin`` attribute on coroutine objects will
   contain a tuple of (filename, line number, function name) tuples
   describing the traceback where the coroutine object was created,
   with the most recent call first. When disabled, ``cr_origin`` will
   be None.

   To enable, pass a *depth* value greater than zero; this sets the
   number of frames whose information will be captured. To disable,
   pass set *depth* to zero.

   This setting is thread-specific.

   .. versionadded:: 3.7

   .. note::
      This function has been added on a provisional basis (see :pep:`411`
      for details.)  Use it only for debugging purposes.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1494)

Unknown directive type "function".

```
.. function:: _enablelegacywindowsfsencoding()

   Changes the :term:`filesystem encoding and error handler` to 'mbcs' and
   'replace' respectively, for consistency with versions of Python prior to
   3.6.

   This is equivalent to defining the :envvar:`PYTHONLEGACYWINDOWSFSENCODING`
   environment variable before launching Python.

   See also :func:`sys.getfilesystemencoding` and
   :func:`sys.getfilesystemencodeerrors`.

   .. availability:: Windows.

   .. versionadded:: 3.6
      See :pep:`529` for more details.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1511)

Unknown directive type "data".

```
.. data:: stdin
          stdout
          stderr

   :term:`File objects <file object>` used by the interpreter for standard
   input, output and errors:

   * ``stdin`` is used for all interactive input (including calls to
     :func:`input`);
   * ``stdout`` is used for the output of :func:`print` and :term:`expression`
```

```
  statements and for the prompts of :func:`input`;
* The interpreter's own prompts and its error messages go to ``stderr``.

These streams are regular :term:`text files <text file>` like those
returned by the :func:`open` function.  Their parameters are chosen as
follows:

* The encoding and error handling are is initialized from
  :c:member:`PyConfig.stdio_encoding` and :c:member:`PyConfig.stdio_errors`.

  On Windows, UTF-8 is used for the console device.  Non-character
  devices such as disk files and pipes use the system locale
  encoding (i.e. the ANSI codepage).  Non-console character
  devices such as NUL (i.e. where ``isatty()`` returns ``True``) use the
  value of the console input and output codepages at startup,
  respectively for stdin and stdout/stderr. This defaults to the
  system :term:`locale encoding` if the process is not initially attached
  to a console.

  The special behaviour of the console can be overridden
  by setting the environment variable PYTHONLEGACYWINDOWSSTDIO
  before starting Python. In that case, the console codepages are
  used as for any other character device.

  Under all platforms, you can override the character encoding by
  setting the :envvar:`PYTHONIOENCODING` environment variable before
  starting Python or by using the new :option:`-X` ``utf8`` command
  line option and :envvar:`PYTHONUTF8` environment variable.  However,
  for the Windows console, this only applies when
  :envvar:`PYTHONLEGACYWINDOWSSTDIO` is also set.

* When interactive, the ``stdout`` stream is line-buffered. Otherwise,
  it is block-buffered like regular text files.  The ``stderr`` stream
  is line-buffered in both cases.  You can make both streams unbuffered
  by passing the :option:`-u` command-line option or setting the
  :envvar:`PYTHONUNBUFFERED` environment variable.

.. versionchanged:: 3.9
   Non-interactive ``stderr`` is now line-buffered instead of fully
   buffered.

.. note::

   To write or read binary data from/to the standard streams, use the
   underlying binary :data:`~io.TextIOBase.buffer` object.  For example, to
   write bytes to :data:`stdout`, use ``sys.stdout.buffer.write(b'abc')``.

   However, if you are writing a library (and do not control in which
   context its code will be executed), be aware that the standard streams
   may be replaced with file-like objects like :class:`io.StringIO` which
   do not support the :attr:`~io.BufferedIOBase.buffer` attribute.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1574)

Unknown directive type "data".

```
.. data:: __stdin__
          __stdout__
          __stderr__

   These objects contain the original values of ``stdin``, ``stderr`` and
   ``stdout`` at the start of the program.  They are used during finalization,
   and could be useful to print to the actual standard stream no matter if the
   ``sys.std*`` object has been redirected.

   It can also be used to restore the actual files to known working file objects
   in case they have been overwritten with a broken object.  However, the
   preferred way to do this is to explicitly save the previous stream before
   replacing it, and restore the saved object.

   .. note::
       Under some conditions ``stdin``, ``stdout`` and ``stderr`` as well as the
       original values ``__stdin__``, ``__stdout__`` and ``__stderr__`` can be
       ``None``. It is usually the case for Windows GUI apps that aren't connected
       to a console and Python apps started with :program:`pythonw`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1595)

Unknown directive type "data".

```
.. data:: stdlib_module_names

   A frozenset of strings containing the names of standard library modules.

   It is the same on all platforms. Modules which are not available on
   some platforms and modules disabled at Python build are also listed.
   All module kinds are listed: pure Python, built-in, frozen and extension
   modules. Test modules are excluded.

   For packages, only the main package is listed: sub-packages and sub-modules
   are not listed. For example, the ``email`` package is listed, but the
   ``email.mime`` sub-package and the ``email.message`` sub-module are not
   listed.

   See also the :attr:`sys.builtin_module_names` list.

   .. versionadded:: 3.10
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1614)

Unknown directive type "data".

```
.. data:: thread_info

   A :term:`named tuple` holding information about the thread
   implementation.

   .. tabularcolumns:: |l|p{0.7\linewidth}|

   +------------------+---------------------------------------------------------+
   | Attribute        | Explanation                                             |
   +==================+=========================================================+
   | :const:`name`    | Name of the thread implementation:                      |
   |                  |                                                         |
   |                  |  * ``'nt'``: Windows threads                            |
   |                  |  * ``'pthread'``: POSIX threads                         |
   |                  |  * ``'solaris'``: Solaris threads                       |
   +------------------+---------------------------------------------------------+
   | :const:`lock`    | Name of the lock implementation:                        |
   |                  |                                                         |
   |                  |  * ``'semaphore'``: a lock uses a semaphore             |
   |                  |  * ``'mutex+cond'``: a lock uses a mutex                |
   |                  |    and a condition variable                             |
   |                  |  * ``None`` if this information is unknown              |
   +------------------+---------------------------------------------------------+
   | :const:`version` | Name and version of the thread library. It is a string, |
   |                  | or ``None`` if this information is unknown.             |
   +------------------+---------------------------------------------------------+

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1644)

Unknown directive type "data".

```
.. data:: tracebacklimit

   When this variable is set to an integer value, it determines the maximum number
   of levels of traceback information printed when an unhandled exception occurs.
   The default is ``1000``.  When set to ``0`` or less, all traceback information
   is suppressed and only the exception type and value are printed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1652)

Unknown directive type "function".

```
.. function:: unraisablehook(unraisable, /)

   Handle an unraisable exception.

   Called when an exception has occurred but there is no way for Python to
   handle it. For example, when a destructor raises an exception or during
   garbage collection (:func:`gc.collect`).

   The *unraisable* argument has the following attributes:

   * *exc_type*: Exception type.
   * *exc_value*: Exception value, can be ``None``.
   * *exc_traceback*: Exception traceback, can be ``None``.
   * *err_msg*: Error message, can be ``None``.
   * *object*: Object causing the exception, can be ``None``.

   The default hook formats *err_msg* and *object* as:
   ``f'{err_msg}: {object!r}'``; use "Exception ignored in" error message
   if *err_msg* is ``None``.

   :func:`sys.unraisablehook` can be overridden to control how unraisable
   exceptions are handled.

   Storing *exc_value* using a custom hook can create a reference cycle. It
   should be cleared explicitly to break the reference cycle when the
   exception is no longer needed.

   Storing *object* using a custom hook can resurrect it if it is set to an
   object which is being finalized. Avoid storing *object* after the custom
   hook completes to avoid resurrecting objects.

   See also :func:`excepthook` which handles uncaught exceptions.

   .. audit-event:: sys.unraisablehook hook,unraisable sys.unraisablehook

      Raise an auditing event ``sys.unraisablehook`` with arguments
      ``hook``, ``unraisable`` when an exception that cannot be handled occurs.
      The ``unraisable`` object is the same as what will be passed to the hook.
      If no hook has been set, ``hook`` may be ``None``.

   .. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1694)

Unknown directive type "data".

```
.. data:: version

   A string containing the version number of the Python interpreter plus additional
   information on the build number and compiler used.  This string is displayed
   when the interactive interpreter is started.  Do not extract version information
   out of it, rather, use :data:`version_info` and the functions provided by the
   :mod:`platform` module.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1703)

Unknown directive type "data".

```
.. data:: api_version

   The C API version for this interpreter.  Programmers may find this useful when
   debugging version conflicts between Python and extension modules.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1709)

Unknown directive type "data".

```
.. data:: version_info
```

```
   A tuple containing the five components of the version number: *major*, *minor*,
   *micro*, *releaselevel*, and *serial*.  All values except *releaselevel* are
   integers; the release level is ``'alpha'``, ``'beta'``, ``'candidate'``, or
   ``'final'``.  The ``version_info`` value corresponding to the Python version 2.0
   is ``(2, 0, 0, 'final', 0)``.  The components can also be accessed by name,
   so ``sys.version_info[0]`` is equivalent to ``sys.version_info.major``
   and so on.

   .. versionchanged:: 3.1
      Added named component attributes.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1722)

Unknown directive type "data".

```
.. data:: warnoptions

   This is an implementation detail of the warnings framework; do not modify this
   value.  Refer to the :mod:`warnings` module for more information on the warnings
   framework.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1729)

Unknown directive type "data".

```
.. data:: winver

   The version number used to form registry keys on Windows platforms. This is
   stored as string resource 1000 in the Python DLL.  The value is normally the
   first three characters of :const:`version`.  It is provided in the :mod:`sys`
   module for informational purposes; modifying this value has no effect on the
   registry keys used by Python.

   .. availability:: Windows.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]sys.rst, line 1740)

Unknown directive type "data".

```
.. data:: _xoptions

   A dictionary of the various implementation-specific flags passed through
   the :option:`-X` command-line option.  Option names are either mapped to
   their values, if given explicitly, or to :const:`True`.  Example:

   .. code-block:: shell-session

      $ ./python -Xpycache_prefix=some_path -Xdev
      Python 3.2a3+ (py3k, Oct 16 2010, 20:14:50)
      [GCC 4.4.3] on linux2
      Type "help", "copyright", "credits" or "license" for more information.
      >>> import sys
      >>> sys._xoptions
      {'pycache_prefix': 'some_path', 'dev': True}

   .. impl-detail::

      This is a CPython-specific way of accessing options passed through
      :option:`-X`.  Other implementations may export them through other
      means, or not at all.

   .. versionadded:: 3.2
```

Citations

[C99] ISO/IEC 9899:1999. "Programming languages -- C." A public draft of this standard is available at http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf.