:mod:`inspect` --- Inspect live objects

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

Unknown directive type "module".

```
.. module:: inspect
   :synopsis: Extract information and source code from live objects.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 7)

Unknown directive type "moduleauthor".

```
.. moduleauthor:: Ka-Ping Yee <ping@lfw.org>
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 8)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Ka-Ping Yee <ping@lfw.org>
```

Source code: :source:`Lib/inspect.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 10); *backlink*

Unknown interpreted text role "source".

The :mod:`inspect` module provides several useful functions to help get information about live objects such as modules, classes, methods, functions, tracebacks, frame objects, and code objects. For example, it can help you examine the contents of a class, retrieve the source code of a method, extract and format the argument list for a function, or get all the information you need to display a detailed traceback.

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

Unknown interpreted text role "mod".

There are four main kinds of services provided by this module: type checking, getting source code, inspecting classes and functions, and examining the interpreter stack.

Types and members

The :func:`getmembers` function retrieves the members of an object such as a class or module. The functions whose names begin with "is" are mainly provided as convenient choices for the second argument to :func:`getmembers`. They also help you determine when you can expect to find the following special attributes:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 31); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 31); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 38)

Unknown interpreted text role "func".

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 38)

Substitution definition contains illegal element <problematic>:

```
<problematic ids="id12" refid="id11">
    :func:`sys.set_coroutine_origin_tracking_depth`

.. |coroutine-origin-link| replace:: :func:`sys.set_coroutine_origin_tracking_depth`
```

Type	Attribute	Description
module	__doc__	documentation string
	__file__	filename (missing for built-in modules)
class	__doc__	documentation string
	__name__	name with which this class was defined
	__qualname__	qualified name
	__module__	name of module in which this class was defined
method	__doc__	documentation string
	__name__	name with which this method was defined
	__qualname__	qualified name
	__func__	function object containing implementation of method
	__self__	instance to which this method is bound, or `None`
	__module__	name of module in which this method was defined
function	__doc__	documentation string
	__name__	name with which this function was defined
	__qualname__	qualified name
	__code__	code object containing compiled function :term:`bytecode` System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 84); *backlink* Unknown interpreted text role "term".
	__defaults__	tuple of any default values for positional or keyword parameters
	__kwdefaults__	mapping of any default values for keyword-only parameters
	__globals__	global namespace in which this function was defined
	__builtins__	builtins namespace
	__annotations__	mapping of parameters names to annotations; `"return"` key is reserved for return annotations.
	__module__	name of module in which this function was defined
traceback	tb_frame	frame object at this level
	tb_lasti	index of last attempted instruction in bytecode
	tb_lineno	current line number in Python source code
	tb_next	next inner traceback object (called by this level)
frame	f_back	next outer frame object (this frame's caller)
	f_builtins	builtins namespace seen by this frame
	f_code	code object being executed in this frame
	f_globals	global namespace seen by this frame
	f_lasti	index of last attempted instruction in bytecode
	f_lineno	current line number in Python source code
	f_locals	local namespace seen by this frame
	f_trace	tracing function for this frame, or `None`
code	co_argcount	number of arguments (not including keyword only arguments, * or ** args)
	co_code	string of raw compiled bytecode
	co_cellvars	tuple of names of cell variables (referenced by containing scopes)
	co_consts	tuple of constants used in the bytecode
	co_filename	name of file in which this code object was created
	co_firstlineno	number of first line in Python source code

Type	Attribute	Description
	co_flags	bitmap of `CO_*` flags, read more :ref:`here <inspect-module-co-flags>`
	co_lnotab	encoded mapping of line numbers to bytecode indices
	co_freevars	tuple of names of free variables (referenced via a function's closure)
	co_posonlyargcount	number of positional only arguments
	co_kwonlyargcount	number of keyword only arguments (not including ** arg)
	co_name	name with which this code object was defined
	co_qualname	fully-qualified name with which this code object was defined
	co_names	tuple of names other than arguments and function locals
	co_nlocals	number of local variables
	co_stacksize	virtual machine stack space required
	co_varnames	tuple of names of arguments and local variables
generator	__name__	name
	__qualname__	qualified name
	gi_frame	frame
	gi_running	is the generator running?
	gi_code	code
	gi_yieldfrom	object being iterated by `yield from`, or `None`
coroutine	__name__	name
	__qualname__	qualified name
	cr_await	object being awaited on, or `None`
	cr_frame	frame
	cr_running	is the coroutine running?
	cr_code	code
	cr_origin	where coroutine was created, or `None`. See |coroutine-origin-link|
builtin	__doc__	documentation string
	__name__	original name of this function or method
	__qualname__	qualified name
	__self__	instance to which a method is bound, or `None`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 169); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 250)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.5

   Add ``__qualname__`` and ``gi_yieldfrom`` attributes to generators.

   The ``__name__`` attribute of generators is now set from the function
   name, instead of the code name, and it can now be modified.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 257)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7

   Add ``cr_origin`` attribute to coroutines.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 261)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10

   Add ``__builtins__`` attribute to functions.
```

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

main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 265)

Unknown directive type "function".

```
.. function:: getmembers(object[, predicate])

   Return all the members of an object in a list of ``(name, value)``
   pairs sorted by name. If the optional *predicate* argumentâ€”which will be
   called with the ``value`` object of each memberâ€”is supplied, only members
   for which the predicate returns a true value are included.

   .. note::

      :func:`getmembers` will only return class attributes defined in the
      metaclass when the argument is a class and those attributes have been
      listed in the metaclass' custom :meth:`__dir__`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 279)

Unknown directive type "function".

```
.. function:: getmembers_static(object[, predicate])

    Return all the members of an object in a list of ``(name, value)``
    pairs sorted by name without triggering dynamic lookup via the descriptor
    protocol, __getattr__ or __getattribute__. Optionally, only return members
    that satisfy a given predicate.

    .. note::

        :func:`getmembers_static` may not be able to retrieve all members
        that getmembers can fetch (like dynamically created attributes)
        and may find members that getmembers can't (like descriptors
        that raise AttributeError). It can also return descriptor objects
        instead of instance members in some cases.

    .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 297)

Unknown directive type "function".

```
.. function:: getmodulename(path)

   Return the name of the module named by the file *path*, without including the
   names of enclosing packages. The file extension is checked against all of
   the entries in :func:`importlib.machinery.all_suffixes`. If it matches,
   the final path component is returned with the extension removed.
   Otherwise, ``None`` is returned.

   Note that this function *only* returns a meaningful name for actual
   Python modules - paths that potentially refer to Python packages will
   still return ``None``.

   .. versionchanged:: 3.3
      The function is based directly on :mod:`importlib`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 313)

Unknown directive type "function".

```
.. function:: ismodule(object)

   Return ``True`` if the object is a module.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 318)

Unknown directive type "function".

```
.. function:: isclass(object)

   Return ``True`` if the object is a class, whether built-in or created in Python
   code.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 324)

Unknown directive type "function".

```
.. function:: ismethod(object)

   Return ``True`` if the object is a bound method written in Python.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 329)

Unknown directive type "function".

```
.. function:: isfunction(object)

   Return ``True`` if the object is a Python function, which includes functions
   created by a :term:`lambda` expression.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 335)

Unknown directive type "function".

```
.. function:: isgeneratorfunction(object)

   Return ``True`` if the object is a Python generator function.

   .. versionchanged:: 3.8
      Functions wrapped in :func:`functools.partial` now return ``True`` if the
      wrapped function is a Python generator function.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 344)

Unknown directive type "function".

```
.. function:: isgenerator(object)

   Return ``True`` if the object is a generator.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 349)

Unknown directive type "function".

```
.. function:: iscoroutinefunction(object)

   Return ``True`` if the object is a :term:`coroutine function`
   (a function defined with an :keyword:`async def` syntax).

   .. versionadded:: 3.5

   .. versionchanged:: 3.8
      Functions wrapped in :func:`functools.partial` now return ``True`` if the
      wrapped function is a :term:`coroutine function`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 361)

Unknown directive type "function".

```
.. function:: iscoroutine(object)

   Return ``True`` if the object is a :term:`coroutine` created by an
   :keyword:`async def` function.

   .. versionadded:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 369)

Unknown directive type "function".

```
.. function:: isawaitable(object)
```

```
   Return ``True`` if the object can be used in :keyword:`await` expression.

   Can also be used to distinguish generator-based coroutines from regular
   generators::

      def gen():
          yield
      @types.coroutine
      def gen_coro():
          yield

      assert not isawaitable(gen())
      assert isawaitable(gen_coro())

   .. versionadded:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 388)

Unknown directive type "function".

```
.. function:: isasyncgenfunction(object)

   Return ``True`` if the object is an :term:`asynchronous generator` function,
   for example::

    >>> async def agen():
    ...     yield 1
    ...
    >>> inspect.isasyncgenfunction(agen)
    True

   .. versionadded:: 3.6

   .. versionchanged:: 3.8
      Functions wrapped in :func:`functools.partial` now return ``True`` if the
      wrapped function is a :term:`asynchronous generator` function.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 406)

Unknown directive type "function".

```
.. function:: isasyncgen(object)

   Return ``True`` if the object is an :term:`asynchronous generator iterator`
   created by an :term:`asynchronous generator` function.

   .. versionadded:: 3.6
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 413)

Unknown directive type "function".

```
.. function:: istraceback(object)

   Return ``True`` if the object is a traceback.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 418)

Unknown directive type "function".

```
.. function:: isframe(object)

   Return ``True`` if the object is a frame.
```

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

Unknown directive type "function".

```
.. function:: iscode(object)

   Return ``True`` if the object is a code.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 428)

Unknown directive type "function".

```
.. function:: isbuiltin(object)

   Return ``True`` if the object is a built-in function or a bound built-in method.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 433)

Unknown directive type "function".

```
.. function:: ismethodwrapper(object)

   Return ``True`` if the type of object is a :class:`~types.MethodWrapperType`.

   These are instances of :class:`~types.MethodWrapperType`, such as :meth:`~object().__str__`,
   :meth:`~object().__eq__` and :meth:`~object().__repr__`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 441)

Unknown directive type "function".

```
.. function:: isroutine(object)

   Return ``True`` if the object is a user-defined or built-in function or method.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 446)

Unknown directive type "function".

```
.. function:: isabstract(object)

   Return ``True`` if the object is an abstract base class.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 451)

Unknown directive type "function".

```
.. function:: ismethoddescriptor(object)

   Return ``True`` if the object is a method descriptor, but not if
   :func:`ismethod`, :func:`isclass`, :func:`isfunction` or :func:`isbuiltin`
   are true.

   This, for example, is true of ``int.__add__``.  An object passing this test
   has a :meth:`~object.__get__` method but not a :meth:`~object.__set__`
   method, but beyond that the set of attributes varies.  A
   :attr:`~definition.__name__` attribute is usually
   sensible, and :attr:`__doc__` often is.

   Methods implemented via descriptors that also pass one of the other tests
   return ``False`` from the :func:`ismethoddescriptor` test, simply because the
   other tests promise more -- you can, e.g., count on having the
   :attr:`__func__` attribute (etc) when an object passes :func:`ismethod`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 469)

Unknown directive type "function".

```
.. function:: isdatadescriptor(object)

   Return ``True`` if the object is a data descriptor.

   Data descriptors have a :attr:`~object.__set__` or a :attr:`~object.__delete__` method.
   Examples are properties (defined in Python), getsets, and members.  The
   latter two are defined in C and there are more specific tests available for
   those types, which is robust across Python implementations.  Typically, data
   descriptors will also have :attr:`~definition.__name__` and :attr:`__doc__` attributes
   (properties, getsets, and members have both of these attributes), but this is
   not guaranteed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 482)

Unknown directive type "function".

```
.. function:: isgetsetdescriptor(object)

   Return ``True`` if the object is a getset descriptor.

   .. impl-detail::

      getsets are attributes defined in extension modules via
      :c:type:`PyGetSetDef` structures.  For Python implementations without such
      types, this method will always return ``False``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 493)

Unknown directive type "function".

```
.. function:: ismemberdescriptor(object)

   Return ``True`` if the object is a member descriptor.

   .. impl-detail::

      Member descriptors are attributes defined in extension modules via
      :c:type:`PyMemberDef` structures.  For Python implementations without such
      types, this method will always return ``False``.
```

Retrieving source code

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 509)

Unknown directive type "function".

```
.. function:: getdoc(object)

   Get the documentation string for an object, cleaned up with :func:`cleandoc`.
   If the documentation string for an object is not provided and the object is
   a class, a method, a property or a descriptor, retrieve the documentation
   string from the inheritance hierarchy.

   .. versionchanged:: 3.5
      Documentation strings are now inherited if not overridden.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 520)

Unknown directive type "function".

```
.. function:: getcomments(object)

   Return in a single string any lines of comments immediately preceding the
   object's source code (for a class, function, or method), or at the top of the
   Python source file (if the object is a module).  If the object's source code
   is unavailable, return ``None``.  This could happen if the object has been
   defined in C or the interactive shell.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 529)

Unknown directive type "function".

```
.. function:: getfile(object)

   Return the name of the (text or binary) file in which an object was defined.
   This will fail with a :exc:`TypeError` if the object is a built-in module,
   class, or function.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 536)

Unknown directive type "function".

```
.. function:: getmodule(object)

   Try to guess which module an object was defined in.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 541)

Unknown directive type "function".

```
.. function:: getsourcefile(object)

   Return the name of the Python source file in which an object was defined.  This
   will fail with a :exc:`TypeError` if the object is a built-in module, class, or
   function.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 548)

Unknown directive type "function".

```
.. function:: getsourcelines(object)

   Return a list of source lines and starting line number for an object. The
   argument may be a module, class, method, function, traceback, frame, or code
   object.  The source code is returned as a list of the lines corresponding to the
   object and the line number indicates where in the original source file the first
   line of code was found.  An :exc:`OSError` is raised if the source code cannot
   be retrieved.

   .. versionchanged:: 3.3
      :exc:`OSError` is raised instead of :exc:`IOError`, now an alias of the
      former.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 562)

Unknown directive type "function".

```
.. function:: getsource(object)

   Return the text of the source code for an object. The argument may be a module,
   class, method, function, traceback, frame, or code object.  The source code is
   returned as a single string.  An :exc:`OSError` is raised if the source code
   cannot be retrieved.

   .. versionchanged:: 3.3
      :exc:`OSError` is raised instead of :exc:`IOError`, now an alias of the
      former.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 574)

Unknown directive type "function".

```
.. function:: cleandoc(doc)

   Clean up indentation from docstrings that are indented to line up with blocks
   of code.

   All leading whitespace is removed from the first line.  Any leading whitespace
   that can be uniformly removed from the second line onwards is removed.  Empty
   lines at the beginning and end are subsequently removed.  Also, all tabs are
   expanded to spaces.
```

Introspecting callables with the Signature object

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 590)

Unknown directive type "versionadded".

```
.. versionadded:: 3.3
```

The Signature object represents the call signature of a callable object and its return annotation. To retrieve a Signature object, use the :func:`signature` function.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 592); *backlink*

Unknown interpreted text role "func".

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

Unknown directive type "function".

```
.. function:: signature(callable, *, follow_wrapped=True, globals=None, locals=None, eval_str=False)

   Return a :class:`Signature` object for the given ``callable``::

      >>> from inspect import signature
      >>> def foo(a, *, b:int, **kwargs):
      ...     pass

      >>> sig = signature(foo)

      >>> str(sig)
      '(a, *, b:int, **kwargs)'

      >>> str(sig.parameters['b'])
      'b:int'

      >>> sig.parameters['b'].annotation
      <class 'int'>

   Accepts a wide range of Python callables, from plain functions and classes to
   :func:`functools.partial` objects.

   For objects defined in modules using stringized annotations
   (``from __future__ import annotations``), :func:`signature` will
   attempt to automatically un-stringize the annotations using
   :func:`inspect.get_annotations()`.  The
   ``global``, ``locals``, and ``eval_str`` parameters are passed
   into :func:`inspect.get_annotations()` when resolving the
   annotations; see the documentation for :func:`inspect.get_annotations()`
   for instructions on how to use these parameters.

   Raises :exc:`ValueError` if no signature can be provided, and
   :exc:`TypeError` if that type of object is not supported.  Also,
   if the annotations are stringized, and ``eval_str`` is not false,
   the ``eval()`` call(s) to un-stringize the annotations could
   potentially raise any kind of exception.

   A slash(/) in the signature of a function denotes that the parameters prior
   to it are positional-only. For more info, see
   :ref:`the FAQ entry on positional-only parameters <faq-positional-only-arguments>`.

   .. versionadded:: 3.5
      ``follow_wrapped`` parameter. Pass ``False`` to get a signature of
      ``callable`` specifically (``callable.__wrapped__`` will not be used to
      unwrap decorated callables.)

   .. versionadded:: 3.10
      ``globals``, ``locals``, and ``eval_str`` parameters.

   .. note::

      Some callables may not be introspectable in certain implementations of
      Python.  For example, in CPython, some built-in functions defined in
      C provide no metadata about their arguments.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 652)

Invalid class attribute value for "class" directive: "Signature(parameters=None, *, return_annotation=Signature.empty)".

```
.. class:: Signature(parameters=None, *, return_annotation=Signature.empty)

   A Signature object represents the call signature of a function and its return
   annotation.  For each parameter accepted by the function it stores a
   :class:`Parameter` object in its :attr:`parameters` collection.

   The optional *parameters* argument is a sequence of :class:`Parameter`
   objects, which is validated to check that there are no parameters with
   duplicate names, and that the parameters are in the right order, i.e.
   positional-only first, then positional-or-keyword, and that parameters with
   defaults follow parameters without defaults.

   The optional *return_annotation* argument, can be an arbitrary Python object,
```

```
   is the "return" annotation of the callable.

   Signature objects are *immutable*.  Use :meth:`Signature.replace` to make a
   modified copy.

   .. versionchanged:: 3.5
      Signature objects are picklable and hashable.

   .. attribute:: Signature.empty

      A special class-level marker to specify absence of a return annotation.

   .. attribute:: Signature.parameters

      An ordered mapping of parameters' names to the corresponding
      :class:`Parameter` objects.  Parameters appear in strict definition
      order, including keyword-only parameters.

      .. versionchanged:: 3.7
         Python only explicitly guaranteed that it preserved the declaration
         order of keyword-only parameters as of version 3.7, although in practice
         this order had always been preserved in Python 3.

   .. attribute:: Signature.return_annotation

      The "return" annotation for the callable.  If the callable has no "return"
      annotation, this attribute is set to :attr:`Signature.empty`.

   .. method:: Signature.bind(*args, **kwargs)

      Create a mapping from positional and keyword arguments to parameters.
      Returns :class:`BoundArguments` if ``*args`` and ``**kwargs`` match the
      signature, or raises a :exc:`TypeError`.

   .. method:: Signature.bind_partial(*args, **kwargs)

      Works the same way as :meth:`Signature.bind`, but allows the omission of
      some required arguments (mimics :func:`functools.partial` behavior.)
      Returns :class:`BoundArguments`, or raises a :exc:`TypeError` if the
      passed arguments do not match the signature.

   .. method:: Signature.replace(*[, parameters][, return_annotation])

      Create a new Signature instance based on the instance replace was invoked
      on.  It is possible to pass different ``parameters`` and/or
      ``return_annotation`` to override the corresponding properties of the base
      signature.  To remove return_annotation from the copied Signature, pass in
      :attr:`Signature.empty`.

      ::

         >>> def test(a, b):
         ...     pass
         >>> sig = signature(test)
         >>> new_sig = sig.replace(return_annotation="new return anno")
         >>> str(new_sig)
         "(a, b) -> 'new return anno'"

   .. classmethod:: Signature.from_callable(obj, *, follow_wrapped=True, globalns=None, localns=None)

      Return a :class:`Signature` (or its subclass) object for a given callable
      ``obj``.  Pass ``follow_wrapped=False`` to get a signature of ``obj``
      without unwrapping its ``__wrapped__`` chain. ``globalns`` and
      ``localns`` will be used as the namespaces when resolving annotations.

      This method simplifies subclassing of :class:`Signature`::

        class MySignature(Signature):
            pass
        sig = MySignature.from_callable(min)
        assert isinstance(sig, MySignature)

      .. versionadded:: 3.5

      .. versionadded:: 3.10
         ``globalns`` and ``localns`` parameters.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 743)

Invalid class attribute value for "class" directive: "Parameter(name, kind, *, default=Parameter.empty, annotation=Parameter.empty)".

```
   .. class:: Parameter(name, kind, *, default=Parameter.empty, annotation=Parameter.empty)

      Parameter objects are *immutable*.  Instead of modifying a Parameter object,
      you can use :meth:`Parameter.replace` to create a modified copy.

      .. versionchanged:: 3.5
         Parameter objects are picklable and hashable.
```

```
.. attribute:: Parameter.empty

   A special class-level marker to specify absence of default values and
   annotations.

.. attribute:: Parameter.name

   The name of the parameter as a string.  The name must be a valid
   Python identifier.

   .. impl-detail::

      CPython generates implicit parameter names of the form ``.0`` on the
      code objects used to implement comprehensions and generator
      expressions.

      .. versionchanged:: 3.6
         These parameter names are exposed by this module as names like
         ``implicit0``.

.. attribute:: Parameter.default

   The default value for the parameter.  If the parameter has no default
   value, this attribute is set to :attr:`Parameter.empty`.

.. attribute:: Parameter.annotation

   The annotation for the parameter.  If the parameter has no annotation,
   this attribute is set to :attr:`Parameter.empty`.

.. attribute:: Parameter.kind

   Describes how argument values are bound to the parameter.  Possible values
   (accessible via :class:`Parameter`, like ``Parameter.KEYWORD_ONLY``):

   .. tabularcolumns:: |l|L|

   +------------------------+----------------------------------------------+
   |    Name                | Meaning                                      |
   +========================+==============================================+
   | *POSITIONAL_ONLY*      | Value must be supplied as a positional       |
   |                        | argument. Positional only parameters are     |
   |                        | those which appear before a ``/`` entry (if  |
   |                        | present) in a Python function definition.    |
   +------------------------+----------------------------------------------+
   | *POSITIONAL_OR_KEYWORD*| Value may be supplied as either a keyword or |
   |                        | positional argument (this is the standard    |
   |                        | binding behaviour for functions implemented  |
   |                        | in Python.)                                  |
   +------------------------+----------------------------------------------+
   | *VAR_POSITIONAL*       | A tuple of positional arguments that aren't  |
   |                        | bound to any other parameter. This           |
   |                        | corresponds to a ``*args`` parameter in a    |
   |                        | Python function definition.                  |
   +------------------------+----------------------------------------------+
   | *KEYWORD_ONLY*         | Value must be supplied as a keyword argument.|
   |                        | Keyword only parameters are those which      |
   |                        | appear after a ``*`` or ``*args`` entry in a |
   |                        | Python function definition.                  |
   +------------------------+----------------------------------------------+
   | *VAR_KEYWORD*          | A dict of keyword arguments that aren't bound|
   |                        | to any other parameter. This corresponds to a|
   |                        | ``**kwargs`` parameter in a Python function  |
   |                        | definition.                                  |
   +------------------------+----------------------------------------------+

   Example: print all keyword-only arguments without default values::

      >>> def foo(a, b, *, c, d=10):
      ...     pass

      >>> sig = signature(foo)
      >>> for param in sig.parameters.values():
      ...     if (param.kind == param.KEYWORD_ONLY and
      ...                       param.default is param.empty):
      ...         print('Parameter:', param)
      Parameter: c

.. attribute:: Parameter.kind.description

   Describes a enum value of Parameter.kind.

   .. versionadded:: 3.8

   Example: print all descriptions of arguments::

      >>> def foo(a, b, *, c, d=10):
      ...     pass

      >>> sig = signature(foo)
      >>> for param in sig.parameters.values():
      ...     print(param.kind.description)
```

```
            positional or keyword
            positional or keyword
            keyword-only
            keyword-only

      .. method:: Parameter.replace(*[, name][, kind][, default][, annotation])

         Create a new Parameter instance based on the instance replaced was invoked
         on.  To override a :class:`Parameter` attribute, pass the corresponding
         argument.  To remove a default value or/and an annotation from a
         Parameter, pass :attr:`Parameter.empty`.

         ::

            >>> from inspect import Parameter
            >>> param = Parameter('foo', Parameter.KEYWORD_ONLY, default=42)
            >>> str(param)
            'foo=42'

            >>> str(param.replace()) # Will create a shallow copy of 'param'
            'foo=42'

            >>> str(param.replace(default=Parameter.empty, annotation='spam'))
            "foo:'spam'"

      .. versionchanged:: 3.4
         In Python 3.3 Parameter objects were allowed to have ``name`` set
         to ``None`` if their ``kind`` was set to ``POSITIONAL_ONLY``.
         This is no longer permitted.
```

Result of a :meth:`Signature.bind` or :meth:`Signature.bind_partial` call. Holds the mapping of arguments to the function's parameters.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 875); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 875); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 878)

Unknown directive type "attribute".

```
   .. attribute:: BoundArguments.arguments

      A mutable mapping of parameters' names to arguments' values.
      Contains only explicitly bound arguments.  Changes in :attr:`arguments`
      will reflect in :attr:`args` and :attr:`kwargs`.

      Should be used in conjunction with :attr:`Signature.parameters` for any
      argument processing purposes.

      .. note::

         Arguments for which :meth:`Signature.bind` or
         :meth:`Signature.bind_partial` relied on a default value are skipped.
         However, if needed, use :meth:`BoundArguments.apply_defaults` to add
         them.

      .. versionchanged:: 3.9
         :attr:`arguments` is now of type :class:`dict`. Formerly, it was of
         type :class:`collections.OrderedDict`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 898)

Unknown directive type "attribute".

```
   .. attribute:: BoundArguments.args

      A tuple of positional arguments values.  Dynamically computed from the
      :attr:`arguments` attribute.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 903)

Unknown directive type "attribute".

```
   .. attribute:: BoundArguments.kwargs
```

```
A dict of keyword arguments values.  Dynamically computed from the
:attr:`arguments` attribute.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 908)

Unknown directive type "attribute".

```
.. attribute:: BoundArguments.signature

   A reference to the parent :class:`Signature` object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 912)

Unknown directive type "method".

```
.. method:: BoundArguments.apply_defaults()

   Set default values for missing arguments.

   For variable-positional arguments (``*args``) the default is an
   empty tuple.

   For variable-keyword arguments (``**kwargs``) the default is an
   empty dict.

   ::

     >>> def foo(a, b='ham', *args): pass
     >>> ba = inspect.signature(foo).bind('spam')
     >>> ba.apply_defaults()
     >>> ba.arguments
     {'a': 'spam', 'b': 'ham', 'args': ()}

   .. versionadded:: 3.5
```

The :attr:`args` and :attr:`kwargs` properties can be used to invoke functions:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 932); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 932); *backlink*

Unknown interpreted text role "attr".

```
def test(a, *, b):
    ...

sig = signature(test)
ba = sig.bind(10, b=20)
test(*ba.args, **ba.kwargs)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 943)

Unknown directive type "seealso".

```
.. seealso::

   :pep:`362` - Function Signature Object.
      The detailed specification, implementation details and examples.
```

Classes and functions

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 954)

Unknown directive type "function".

```
.. function:: getclasstree(classes, unique=False)

   Arrange the given list of classes into a hierarchy of nested lists. Where a
   nested list appears, it contains classes derived from the class whose entry
   immediately precedes the list.  Each entry is a 2-tuple containing a class and a
   tuple of its base classes.  If the *unique* argument is true, exactly one entry
   appears in the returned structure for each class in the given list.  Otherwise,
```

```
   classes using multiple inheritance and their descendants will appear multiple
   times.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 965)

Unknown directive type "function".

```
.. function:: getfullargspec(func)

   Get the names and default values of a Python function's parameters.  A
   :term:`named tuple` is returned:

   ``FullArgSpec(args, varargs, varkw, defaults, kwonlyargs, kwonlydefaults,
   annotations)``

   *args* is a list of the positional parameter names.
   *varargs* is the name of the ``*`` parameter or ``None`` if arbitrary
   positional arguments are not accepted.
   *varkw* is the name of the ``**`` parameter or ``None`` if arbitrary
   keyword arguments are not accepted.
   *defaults* is an *n*-tuple of default argument values corresponding to the
   last *n* positional parameters, or ``None`` if there are no such defaults
   defined.
   *kwonlyargs* is a list of keyword-only parameter names in declaration order.
   *kwonlydefaults* is a dictionary mapping parameter names from *kwonlyargs*
   to the default values used if no argument is supplied.
   *annotations* is a dictionary mapping parameter names to annotations.
   The special key ``"return"`` is used to report the function return value
   annotation (if any).

   Note that :func:`signature` and
   :ref:`Signature Object <inspect-signature-object>` provide the recommended
   API for callable introspection, and support additional behaviours (like
   positional-only arguments) that are sometimes encountered in extension module
   APIs. This function is retained primarily for use in code that needs to
   maintain compatibility with the Python 2 ``inspect`` module API.

   .. versionchanged:: 3.4
      This function is now based on :func:`signature`, but still ignores
      ``__wrapped__`` attributes and includes the already bound first
      parameter in the signature output for bound methods.

   .. versionchanged:: 3.6
      This method was previously documented as deprecated in favour of
      :func:`signature` in Python 3.5, but that decision has been reversed
      in order to restore a clearly supported standard interface for
      single-source Python 2/3 code migrating away from the legacy
      :func:`getargspec` API.

   .. versionchanged:: 3.7
      Python only explicitly guaranteed that it preserved the declaration
      order of keyword-only parameters as of version 3.7, although in practice
      this order had always been preserved in Python 3.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1013)

Unknown directive type "function".

```
.. function:: getargvalues(frame)

   Get information about arguments passed into a particular frame.  A
   :term:`named tuple` ``ArgInfo(args, varargs, keywords, locals)`` is
   returned. *args* is a list of the argument names.  *varargs* and *keywords*
   are the names of the ``*`` and ``**`` arguments or ``None``.  *locals* is the
   locals dictionary of the given frame.

   .. note::
      This function was inadvertently marked as deprecated in Python 3.5.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1025)

Unknown directive type "function".

```
.. function:: formatargvalues(args[, varargs, varkw, locals, formatarg, formatvarargs, formatvarkw, forma

   Format a pretty argument spec from the four values returned by
   :func:`getargvalues`.  The format\* arguments are the corresponding optional
   formatting functions that are called to turn names and values into strings.

   .. note::
      This function was inadvertently marked as deprecated in Python 3.5.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1035)

Unknown directive type "function".

```
.. function:: getmro(cls)

   Return a tuple of class cls's base classes, including cls, in method resolution
   order.  No class appears more than once in this tuple. Note that the method
   resolution order depends on cls's type.  Unless a very peculiar user-defined
   metatype is in use, cls will be the first element of the tuple.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1043)

Unknown directive type "function".

```
.. function:: getcallargs(func, /, *args, **kwds)

   Bind the *args* and *kwds* to the argument names of the Python function or
   method *func*, as if it was called with them. For bound methods, bind also the
   first argument (typically named ``self``) to the associated instance. A dict
   is returned, mapping the argument names (including the names of the ``*`` and
   ``**`` arguments, if any) to their values from *args* and *kwds*. In case of
   invoking *func* incorrectly, i.e. whenever ``func(*args, **kwds)`` would raise
   an exception because of incompatible signature, an exception of the same type
   and the same or similar message is raised. For example::

    >>> from inspect import getcallargs
    >>> def f(a, b=1, *pos, **named):
    ...     pass
    >>> getcallargs(f, 1, 2, 3) == {'a': 1, 'named': {}, 'b': 2, 'pos': (3,)}
    True
    >>> getcallargs(f, a=2, x=4) == {'a': 2, 'named': {'x': 4}, 'b': 1, 'pos': ()}
    True
    >>> getcallargs(f)
    Traceback (most recent call last):
    ...
    TypeError: f() missing 1 required positional argument: 'a'

   .. versionadded:: 3.2

   .. deprecated:: 3.5
      Use :meth:`Signature.bind` and :meth:`Signature.bind_partial` instead.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1072)

Unknown directive type "function".

```
.. function:: getclosurevars(func)

   Get the mapping of external name references in a Python function or
   method *func* to their current values. A
   :term:`named tuple` ``ClosureVars(nonlocals, globals, builtins, unbound)``
   is returned. *nonlocals* maps referenced names to lexical closure
   variables, *globals* to the function's module globals and *builtins* to
   the builtins visible from the function body. *unbound* is the set of names
   referenced in the function that could not be resolved at all given the
   current module globals and builtins.

   :exc:`TypeError` is raised if *func* is not a Python function or method.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1088)

Unknown directive type "function".

```
.. function:: unwrap(func, *, stop=None)

   Get the object wrapped by *func*. It follows the chain of :attr:`__wrapped__`
   attributes returning the last object in the chain.

   *stop* is an optional callback accepting an object in the wrapper chain
   as its sole argument that allows the unwrapping to be terminated early if
   the callback returns a true value. If the callback never returns a true
   value, the last object in the chain is returned as usual. For example,
   :func:`signature` uses this to stop unwrapping if any object in the
```

```
chain has a ``__signature__`` attribute defined.

:exc:`ValueError` is raised if a cycle is encountered.

.. versionadded:: 3.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1105)

Unknown directive type "function".

```
.. function:: get_annotations(obj, *, globals=None, locals=None, eval_str=False)

   Compute the annotations dict for an object.

   ``obj`` may be a callable, class, or module.
   Passing in an object of any other type raises :exc:`TypeError`.

   Returns a dict.  ``get_annotations()`` returns a new dict every time
   it's called; calling it twice on the same object will return two
   different but equivalent dicts.

   This function handles several details for you:

   * If ``eval_str`` is true, values of type ``str`` will
     be un-stringized using :func:`eval()`.  This is intended
     for use with stringized annotations
     (``from __future__ import annotations``).
   * If ``obj`` doesn't have an annotations dict, returns an
     empty dict.  (Functions and methods always have an
     annotations dict; classes, modules, and other types of
     callables may not.)
   * Ignores inherited annotations on classes.  If a class
     doesn't have its own annotations dict, returns an empty dict.
   * All accesses to object members and dict values are done
     using ``getattr()`` and ``dict.get()`` for safety.
   * Always, always, always returns a freshly-created dict.

   ``eval_str`` controls whether or not values of type ``str`` are replaced
   with the result of calling :func:`eval()` on those values:

   * If eval_str is true, :func:`eval()` is called on values of type ``str``.
     (Note that ``get_annotations`` doesn't catch exceptions; if :func:`eval()`
     raises an exception, it will unwind the stack past the ``get_annotations``
     call.)
   * If eval_str is false (the default), values of type ``str`` are unchanged.

   ``globals`` and ``locals`` are passed in to :func:`eval()`; see the documentation
   for :func:`eval()` for more information.  If ``globals`` or ``locals``
   is ``None``, this function may replace that value with a context-specific
   default, contingent on ``type(obj)``:

   * If ``obj`` is a module, ``globals`` defaults to ``obj.__dict__``.
   * If ``obj`` is a class, ``globals`` defaults to
     ``sys.modules[obj.__module__].__dict__`` and ``locals`` defaults
     to the ``obj`` class namespace.
   * If ``obj`` is a callable, ``globals`` defaults to ``obj.__globals__``,
     although if ``obj`` is a wrapped function (using
     ``functools.update_wrapper()``) it is first unwrapped.

   Calling ``get_annotations`` is best practice for accessing the
   annotations dict of any object.  See :ref:`annotations-howto` for
   more information on annotations best practices.

   .. versionadded:: 3.10
```

The interpreter stack

When the following functions return "frame records," each record is a :term:`named tuple` `FrameInfo(frame, filename, lineno, function, code_context, index)`. The tuple contains the frame object, the filename, the line number of the current line, the function name, a list of lines of context from the source code, and the index of the current line within that list.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1166); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1174)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.5
   Return a named tuple instead of a tuple.
```

Note

Keeping references to frame objects, as found in the first element of the frame records these functions return, can cause your program to create reference cycles. Once a reference cycle has been created, the lifespan of all objects which can be accessed from the objects which form the cycle can become much longer even if Python's optional cycle detector is enabled. If such cycles must be created, it is important to ensure they are explicitly broken to avoid the delayed destruction of objects and increased memory consumption which occurs.

Though the cycle detector will catch these, destruction of the frames (and local variables) can be made deterministic by removing the cycle in a :keyword:`finally` clause. This is also important if the cycle detector was disabled when Python was compiled or using :func:`gc.disable`. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1187); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1187); *backlink*

Unknown interpreted text role "func".

```
def handle_stackframe_without_leak():
    frame = inspect.currentframe()
    try:
        # do something with the frame
    finally:
        del frame
```

If you want to keep the frame around (for example to print a traceback later), you can also break reference cycles by using the :meth:`frame.clear` method.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1199); *backlink*

Unknown interpreted text role "meth".

The optional *context* argument supported by most of these functions specifies the number of lines of context to return, which are centered around the current line.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1208)

Unknown directive type "function".

```
.. function:: getframeinfo(frame, context=1)

   Get information about a frame or traceback object.  A :term:`named tuple`
   ``Traceback(filename, lineno, function, code_context, index)`` is returned.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1214)

Unknown directive type "function".

```
.. function:: getouterframes(frame, context=1)

   Get a list of frame records for a frame and all outer frames.  These frames
   represent the calls that lead to the creation of *frame*. The first entry in the
   returned list represents *frame*; the last entry represents the outermost call
   on *frame*'s stack.

   .. versionchanged:: 3.5
      A list of :term:`named tuples <named tuple>`
      ``FrameInfo(frame, filename, lineno, function, code_context, index)``
      is returned.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1227)

Unknown directive type "function".

```
.. function:: getinnerframes(traceback, context=1)
```

```
Get a list of frame records for a traceback's frame and all inner frames.  These
frames represent calls made as a consequence of *frame*.  The first entry in the
list represents *traceback*; the last entry represents where the exception was
raised.

.. versionchanged:: 3.5
   A list of :term:`named tuples <named tuple>`
   ``FrameInfo(frame, filename, lineno, function, code_context, index)``
   is returned.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1240)

Unknown directive type "function".

```
.. function:: currentframe()

   Return the frame object for the caller's stack frame.

   .. impl-detail::

      This function relies on Python stack frame support in the interpreter,
      which isn't guaranteed to exist in all implementations of Python.  If
      running in an implementation without Python stack frame support this
      function returns ``None``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1252)

Unknown directive type "function".

```
.. function:: stack(context=1)

   Return a list of frame records for the caller's stack.  The first entry in the
   returned list represents the caller; the last entry represents the outermost
   call on the stack.

   .. versionchanged:: 3.5
      A list of :term:`named tuples <named tuple>`
      ``FrameInfo(frame, filename, lineno, function, code_context, index)``
      is returned.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1264)

Unknown directive type "function".

```
.. function:: trace(context=1)

   Return a list of frame records for the stack between the current frame and the
   frame in which an exception currently being handled was raised in.  The first
   entry in the list represents the caller; the last entry represents where the
   exception was raised.

   .. versionchanged:: 3.5
      A list of :term:`named tuples <named tuple>`
      ``FrameInfo(frame, filename, lineno, function, code_context, index)``
      is returned.
```

Fetching attributes statically

Both :func:`getattr` and :func:`hasattr` can trigger code execution when fetching or checking for the existence of attributes. Descriptors, like properties, will be invoked and :meth:`__getattr__` and :meth:`__getattribute__` may be called.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1280); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1280); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1280); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1280); *backlink*

Unknown interpreted text role "meth".

For cases where you want passive introspection, like documentation tools, this can be inconvenient. :func:`getattr_static` has the same signature as :func:`getattr` but avoids executing code when it fetches attributes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1285); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1285); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1289)

Unknown directive type "function".

```
.. function:: getattr_static(obj, attr, default=None)

   Retrieve attributes without triggering dynamic lookup via the
   descriptor protocol, :meth:`__getattr__` or :meth:`__getattribute__`.

   Note: this function may not be able to retrieve all attributes
   that getattr can fetch (like dynamically created attributes)
   and may find attributes that getattr can't (like descriptors
   that raise AttributeError). It can also return descriptors objects
   instead of instance members.

   If the instance :attr:`~object.__dict__` is shadowed by another member (for
   example a property) then this function will be unable to find instance
   members.

   .. versionadded:: 3.2
```

:func:`getattr_static` does not resolve descriptors, for example slot descriptors or getset descriptors on objects implemented in C. The descriptor object is returned instead of the underlying attribute.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1306); *backlink*

Unknown interpreted text role "func".

You can handle these with code like the following. Note that for arbitrary getset descriptors invoking these may trigger code execution:

```
# example code for resolving the builtin descriptor types
class _foo:
    __slots__ = ['foo']

slot_descriptor = type(_foo.foo)
getset_descriptor = type(type(open(__file__)).name)
wrapper_descriptor = type(str.__dict__['__add__'])
descriptor_types = (slot_descriptor, getset_descriptor, wrapper_descriptor)

result = getattr_static(some_object, 'foo')
if type(result) in descriptor_types:
    try:
        result = result.__get__()
    except AttributeError:
        # descriptors can raise AttributeError to
        # indicate there is no underlying value
        # in which case the descriptor itself will
        # have to do
        pass
```

Current State of Generators and Coroutines

When implementing coroutine schedulers and for other advanced uses of generators, it is useful to determine whether a generator is currently executing, is waiting to start or resume or execution, or has already terminated. :func:`getgeneratorstate` allows the current state of a generator to be determined easily.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1338); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1344)

Unknown directive type "function".

```
.. function:: getgeneratorstate(generator)

   Get current state of a generator-iterator.

   Possible states are:
    * GEN_CREATED: Waiting to start execution.
    * GEN_RUNNING: Currently being executed by the interpreter.
    * GEN_SUSPENDED: Currently suspended at a yield expression.
    * GEN_CLOSED: Execution has completed.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1356)

Unknown directive type "function".

```
.. function:: getcoroutinestate(coroutine)

   Get current state of a coroutine object.  The function is intended to be
   used with coroutine objects created by :keyword:`async def` functions, but
   will accept any coroutine-like object that has ``cr_running`` and
   ``cr_frame`` attributes.

   Possible states are:
    * CORO_CREATED: Waiting to start execution.
    * CORO_RUNNING: Currently being executed by the interpreter.
    * CORO_SUSPENDED: Currently suspended at an await expression.
    * CORO_CLOSED: Execution has completed.

   .. versionadded:: 3.5
```

The current internal state of the generator can also be queried. This is mostly useful for testing purposes, to ensure that internal state is being updated as expected:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1375)

Unknown directive type "function".

```
.. function:: getgeneratorlocals(generator)

   Get the mapping of live local variables in *generator* to their current
   values.  A dictionary is returned that maps from variable names to values.
   This is the equivalent of calling :func:`locals` in the body of the
   generator, and all the same caveats apply.

   If *generator* is a :term:`generator` with no currently associated frame,
   then an empty dictionary is returned.  :exc:`TypeError` is raised if
   *generator* is not a Python generator object.

   .. impl-detail::

      This function relies on the generator exposing a Python stack frame
      for introspection, which isn't guaranteed to be the case in all
      implementations of Python. In such cases, this function will always
      return an empty dictionary.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1395)

Unknown directive type "function".

```
.. function:: getcoroutinelocals(coroutine)

   This function is analogous to :func:`~inspect.getgeneratorlocals`, but
   works for coroutine objects created by :keyword:`async def` functions.

   .. versionadded:: 3.5
```

Code Objects Bit Flags

Python code objects have a `co_flags` attribute, which is a bitmap of the following flags:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1411)

Unknown directive type "data".

```
.. data:: CO_OPTIMIZED

   The code object is optimized, using fast locals.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1415)

Unknown directive type "data".

```
.. data:: CO_NEWLOCALS

   If set, a new dict will be created for the frame's ``f_locals`` when
   the code object is executed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1420)

Unknown directive type "data".

```
.. data:: CO_VARARGS

   The code object has a variable positional parameter (``*args``-like).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1424)

Unknown directive type "data".

```
.. data:: CO_VARKEYWORDS

   The code object has a variable keyword parameter (``**kwargs``-like).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1428)

Unknown directive type "data".

```
.. data:: CO_NESTED

   The flag is set when the code object is a nested function.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1432)

Unknown directive type "data".

```
.. data:: CO_GENERATOR

   The flag is set when the code object is a generator function, i.e.
   a generator object is returned when the code object is executed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1437)

Unknown directive type "data".

```
.. data:: CO_COROUTINE

   The flag is set when the code object is a coroutine function.
   When the code object is executed it returns a coroutine object.
   See :pep:`492` for more details.

   .. versionadded:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1445)

Unknown directive type "data".

```
.. data:: CO_ITERABLE_COROUTINE

   The flag is used to transform generators into generator-based
   coroutines.  Generator objects with this flag can be used in
   ``await`` expression, and can ``yield from`` coroutine objects.
```

```
See :pep:`492` for more details.

.. versionadded:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1454)

Unknown directive type "data".

```
.. data:: CO_ASYNC_GENERATOR

   The flag is set when the code object is an asynchronous generator
   function.  When the code object is executed it returns an
   asynchronous generator object.  See :pep:`525` for more details.

   .. versionadded:: 3.6
```

Note

The flags are specific to CPython, and may not be defined in other Python implementations. Furthermore, the flags are an implementation detail, and can be removed or deprecated in future Python releases. It's recommended to use public APIs from the :mod:`inspect` module for any introspection needs.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1463); *backlink*

Unknown interpreted text role "mod".

Command Line Interface

The :mod:`inspect` module also provides a basic introspection capability from the command line.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1475); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1478)

Unknown directive type "program".

```
.. program:: inspect
```

By default, accepts the name of a module and prints the source of that module. A class or function within the module can be printed instead by appended a colon and the qualified name of the target object.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 1484)

Unknown directive type "cmdoption".

```
.. cmdoption:: --details

   Print information about the specified object rather than the source code
```

Docutils System Messages

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]inspect.rst, line 234); *backlink*

Undefined substitution referenced: "coroutine-origin-link".