# Python technicalities

Release 10.3

**The Sage Development Team** 

# **CONTENTS**

1	Utilities for interfacing with the standard library's atexit module.	3
2	String <-> bytes encoding/decoding	5
3	Various functions to debug Python internals	7
4	Variants of getattr()	11
5	Metaclasses for Cython extension types	17
6	Slot wrappers	19
7	Delete item from PyDict by exact value and hash	21
8	Indices and Tables	23
Python Module Index		25
Index		27

SageMath has various modules to provide access to low-level Python internals.

CONTENTS 1

2 CONTENTS

# UTILITIES FOR INTERFACING WITH THE STANDARD LIBRARY'S ATEXIT MODULE.

```
{\tt class} \ {\tt sage.cpython.atexit.restore\_atexit}
```

Bases: object

Context manager that restores the state of the atexit module to its previous state when exiting the context.

#### INPUT:

- run (bool, default: False) if True, when exiting the context (but before restoring the old exit functions), run all atexit functions which were added inside the context.
- clear (bool, default: equal to run) if True, clear already registered atexit handlers upon entering the context.

**Warning:** The combination run=True and clear=False will cause already-registered exit functions to be run twice: once when exiting the context and again when exiting Python.

#### **EXAMPLES:**

For this example we will wrap the entire example with restore\_atexit (clear=True) so as to start with a fresh atexit module state for the sake of the example.

Note that the function atexit.\_run\_exitfuncs() runs all registered handlers, and then clears the list of handlers, so we can use it to test manipulation of the atexit state:

```
sage: import atexit
sage: from sage.cpython.atexit import restore_atexit
sage: def handler(*args, **kwargs):
....: import sys
        # see https://github.com/sagemath/sage/issues/25270#comment:56
. . . . :
        sys.stdout.write(str((args, kwargs)))
         sys.stdout.write(' \ n')
sage: atexit.register(handler, 1, 2, c=3)
<function handler at 0x...>
sage: atexit.register(handler, 4, 5, d=6)
<function handler at 0x...>
sage: with restore_atexit(clear=True):
        atexit._run_exitfuncs() # Should be none registered
         atexit.register(handler, 1, 2, c=3)
         with restore_atexit():
             atexit._run_exitfuncs() # Run just registered handler
       atexit._run_exitfuncs() # Handler should be run again
<function handler at 0x...>
```

```
((1, 2), {'c': 3})
((1, 2), {'c': 3})
```

We test the run option:

```
sage: with restore_atexit(run=True):
....:  # this handler is run when exiting the context
....:  _ = atexit.register(handler, 7, 8, e=9)
((7, 8), {'e': 9})
sage: with restore_atexit(clear=False, run=True):
....:  # original handlers are run when exiting the context
....: pass
((4, 5), {'d': 6})
((1, 2), {'c': 3})
```

The original handlers are still in place:

```
sage: atexit._run_exitfuncs()
((4, 5), {'d': 6})
((1, 2), {'c': 3})
```

## STRING <-> BYTES ENCODING/DECODING

```
\verb|sage.cpython.string.bytes_to_str(b, encoding=None, errors=None)|\\
```

Convert bytes to str.

This decodes the given bytes to a Python 3 unicode str using the specified encoding. It is a no-op on str input.

## **EXAMPLES:**

```
sage: from sage.cpython.string import bytes_to_str
sage: s = bytes_to_str(b'\xcf\x80')
sage: s == u'\pi'
True
sage: bytes_to_str([])
Traceback (most recent call last):
...
TypeError: expected bytes, list found
```

sage.cpython.string.str\_to\_bytes(s, encoding=None, errors=None)

Convert str or unicode to bytes.

It encodes the given str to a Python 3 bytes using the specified encoding. It is a no-op on bytes input.

## **EXAMPLES:**

```
sage: from sage.cpython.string import str_to_bytes
sage: bs = [str_to_bytes(u'\pi')]
sage: all(b == b'\xcf\x80' for b in bs)
True
sage: str_to_bytes([])
Traceback (most recent call last):
...
TypeError: expected str... list found
```

## **VARIOUS FUNCTIONS TO DEBUG PYTHON INTERNALS**

sage.cpython.debug.getattr\_debug(obj, name, default='\_no\_default')

A re-implementation of getattr() with lots of debugging info.

This will correctly use \_\_getattr\_\_ if needed. On the other hand, it assumes a generic (not overridden) implementation of \_\_getattribute\_\_. Note that Cython implements \_\_getattr\_\_ for a cdef class using \_\_getattribute\_\_, so this will not detect a \_\_getattr\_\_ in that case.

#### INPUT:

- obj the object whose attribute is requested
- name (string) the name of the attribute
- default default value to return if attribute was not found

## **EXAMPLES:**

```
sage: _ = getattr_debug(list, "reverse") # not tested - broken in python 3.12
getattr_debug(obj=<class 'list'>, name='reverse'):
 type(obj) = <class 'type'>
 object has __dict__ slot (<class 'dict'>)
 did not find 'reverse' in MRO classes
 found 'reverse' in object __dict_
 returning <method 'reverse' of 'list' objects> (<class 'method_descriptor'>)
sage: _ = getattr_debug([], "reverse")
getattr_debug(obj=[], name='reverse'):
 type(obj) = <class 'list'>
 object does not have __dict__ slot
 found 'reverse' in dict of <class 'list'>
 got <method 'reverse' of 'list' objects> (<class 'method_descriptor'>)
 attribute is ordinary descriptor (has __get__)
 calling __get__()
 returning <built-in method reverse of list object at 0x... (<class 'builtin_
→function_or_method'>)
sage: _ = getattr_debug([], "__doc__")
getattr_debug(obj=[], name='__doc__'):
 type(obj) = <class 'list'>
 object does not have __dict__ slot
 found '__doc__' in dict of <class 'list'>
 got ... 'str'>)
 returning ... 'str'>)
sage: _ = getattr_debug(gp(1), "log")
⇔needs sage.libs.pari
getattr_debug(obj=1, name='log'):
  type(obj) = <class 'sage.interfaces.gp.GpElement'>
 object has __dict__ slot (<class 'dict'>)
```

```
did not find 'log' in MRO classes
 object __dict__ does not have 'log'
 calling __getattr__()
 returning log (<class 'sage.interfaces.expect.FunctionElement'>)
sage: from ipywidgets import IntSlider
sage: _ = getattr_debug(IntSlider(), "value")
getattr_debug(obj=IntSlider(value=0), name='value'):
 type(obj) = <class 'ipywidgets.widgets.widget_int.IntSlider'>
 object has __dict__ slot (<class 'dict'>)
 found 'value' in dict of <class 'ipywidgets.widgets.widget_int._Int'>
 got <traitlets.traitlets.CInt object at ... (<class 'traitlets.traitlets.CInt'>)
 attribute is data descriptor (has __get__ and __set__)
 ignoring __dict__ because we have a data descriptor
 calling __get__()
 returning 0 (<class 'int'>)
sage: _ = getattr_debug(1, "foo")
Traceback (most recent call last):
AttributeError: 'sage.rings.integer' object has no attribute 'foo'...
sage: _ = getattr_debug(1, "foo", "xyz")
getattr_debug(obj=1, name='foo'):
 type(obj) = <class 'sage.rings.integer.Integer'>
 object does not have __dict__ slot
 did not find 'foo' in MRO classes
 class does not have __getattr__
 attribute not found
 returning default 'xyz'
```

sage.cpython.debug.shortrepr(obj, max=50)

Return repr (obj) bounded to max characters. If the string is too long, it is truncated and ~~~ is added to the end.

## EXAMPLES:

```
sage: from sage.cpython.debug import shortrepr
sage: print(shortrepr("Hello world!"))
'Hello world!'
sage: print(shortrepr("Hello world!" * 4))
'Hello world!Hello world!Hello world!'
sage: print(shortrepr("Hello world!Hello world!'
sage: print(shortrepr("Hello world!" * 5))
'Hello world!Hello world!Hello world!Hello worl~~~
```

 $\verb|sage.cpython.debug.type_debug| (cls)$ 

Print all internals of the type cls

#### EXAMPLES:

```
sage: type_debug(object) # random
<class 'object'> (0x7fc57da7f040)
  ob_refcnt: 9739
  ob_type: <class 'type'>
  tp_name: object
  tp_basicsize: 16
  tp_itemsize: 0
  tp_dictoffset: 0
  tp_weaklistoffset: 0
  tp_base (__base__): NULL
```

```
tp_bases (__bases__): tuple:
tp_mro (__mro__): tuple:
 <class 'object'>
tp_dict (__dict__): dict:
   __setattr__': <slot wrapper '__setattr__' of 'object' objects>
  '__reduce_ex__': <method '__reduce_ex__' of 'object' objects>
  '__new__': <built-in method __new__ of type object at 0x7fc57da7f040>
    _reduce__': <method '__reduce__' of 'object' objects>
    _str__': <slot wrapper '__str__' of 'object' objects>
  '__format__': <method '__format__' of 'object' objects>
  '__getattribute__': <slot wrapper '__getattribute__' of 'object' objects>
  '__class__': <attribute '__class__' of 'object' objects>
  '__delattr__': <slot wrapper '__delattr__' of 'object' objects>
  '__subclasshook__': <method '__subclasshook__' of 'object' objects>
  '__repr__': <slot wrapper '__repr__' of 'object' objects>
  '_hash__': <slot wrapper '_hash__' of 'object' objects>
  '__sizeof__': <method '__sizeof__' of 'object' objects>
  '__doc__': 'The most base type'
  '__init__': <slot wrapper '__init__' of 'object' objects>
tp_alloc: PyType_GenericAlloc
tp_new (__new__): 0x7fc57d7594f0
tp_init (__init__): 0x7fc57d758ee0
tp_dealloc (__dealloc__): 0x7fc57d757010
tp_free: PyObject_Del
tp_repr (__repr__): 0x7fc57d75b990
tp_print: NULL
tp_hash (__hash__): _Py_HashPointer
tp_call (__call__): NULL
tp_str (__str__): 0x7fc57d757020
tp_compare (__cmp__): NULL
tp_richcompare (__richcmp__): NULL
tp_getattr (__getattribute__): NULL
tp_setattr (__setattribute__): NULL
tp_getattro (__getattribute__): PyObject_GenericGetAttr
tp_setattro (__setattribute__): PyObject_GenericSetAttr
tp_iter (__iter__): NULL
tp_iternext (__next__): NULL
tp_descr_get (__get__): NULL
tp_descr_set (__set__): NULL
tp_cache: NULL
tp_weaklist: NULL
tp traverse: NULL
tp_clear: NULL
tp_is_gc: NULL
tp_as_number: NULL
tp_as_sequence: NULL
tp_as_mapping: NULL
tp_as_buffer: NULL
tp_flags:
 HAVE_GETCHARBUFFER
 HAVE_SEQUENCE_IN
 HAVE_INPLACEOPS
 HAVE_RICHCOMPARE
 HAVE_WEAKREFS
 HAVE ITER
  HAVE_CLASS
  BASETYPE
```

```
READY
HAVE_INDEX
HAVE_VERSION_TAG
VALID_VERSION_TAG
tp_version_tag: 2
sage: type_debug(None)
Traceback (most recent call last):
...
TypeError: None is not a type
```

## **FOUR**

## **VARIANTS OF GETATTR()**

```
class sage.cpython.getattr.AttributeErrorMessage
    Bases: object
```

Tries to emulate the standard Python AttributeError message.

**Note:** The typical fate of an attribute error is being caught. Hence, under normal circumstances, nobody will ever see the error message. The idea for this class is to provide an object that is fast to create and whose string representation is an attribute error's message. That string representation is only created if someone wants to see it.

#### **EXAMPLES:**

```
sage: from sage.cpython.getattr import AttributeErrorMessage
sage: AttributeErrorMessage(int(1), 'bla')
'int' object has no attribute 'bla'
```

### AUTHOR:

• Simon King (2011-05-21)

cls

name

```
sage.cpython.getattr.dir_with_other_class(self, *cls)
```

Emulates dir(self), as if self was also an instance cls, right after caller\_class in the method resolution order(self.\_\_class\_\_.mro())

**EXAMPLES**:

```
sage: class A():
....: a = 1
• • • b = 2
....: c = 3
sage: class B():
....: b = 2
....: c = 3
d = 4
sage: x = A()
sage: x.c = 1; x.e = 1
sage: from sage.cpython.getattr import dir_with_other_class
sage: dir_with_other_class(x, B)
[..., 'a', 'b', 'c', 'd', 'e']
sage: class C():
\dots: f = 6
sage: dir_with_other_class(x, B, C)
[..., 'a', 'b', 'c', 'd', 'e', 'f']
```

Check that objects without dicts are well handled:

```
sage: # needs sage.misc.cython
sage: cython("cdef class A:\n cdef public int a")
sage: cython("cdef class B:\n cdef public int b")
sage: x = A()
sage: x.a = 1
sage: hasattr(x,'__dict__')
False
sage: dir_with_other_class(x, B)
[..., 'a', 'b']
```

sage.cpython.getattr\_from\_other\_class(self, cls, name)

Emulate getattr(self, name), as if self was an instance of cls.

#### **INPUT:**

- self some object
- cls a new-style class
- name a string

If self is an instance of cls, raises an AttributeError, to avoid a double lookup. This function is intended to be called from \_\_getattr\_\_, and so should not be called if name is an attribute of self.

## **EXAMPLES:**

```
sage: from sage.cpython.getattr import getattr_from_other_class
sage: class A():
....: def inc(self):
              return self + 1
. . . . :
. . . . :
...: @staticmethod
. . . . :
         def greeting():
               print("Hello World!")
. . . . :
       @lazy_attribute
. . . . :
          def lazy_attribute(self):
. . . . :
               return repr(self)
sage: getattr_from_other_class(1, A, "inc")
```

Static methods work:

```
sage: getattr_from_other_class(1, A, "greeting")()
Hello World!
```

Caveat: lazy attributes work with extension types only if they allow attribute assignment or have a public attribute \_cached\_methods of type <dict>. This condition is satisfied, e.g., by any class that is derived from Parent:

The integer ring is a parent, so, lazy attributes work:

In general, descriptors are not yet well supported, because they often do not accept to be cheated with the type of their instance:

```
sage: A.__weakref__._get__(1)
Traceback (most recent call last):
...
TypeError: descriptor '__weakref__' for 'A' objects doesn't apply
to ...'sage.rings.integer.Integer' object
```

When this occurs, an AttributeError is raised:

```
sage: getattr_from_other_class(1, A, "__weakref__")
Traceback (most recent call last):
...
AttributeError: 'sage.rings.integer.Integer' object has no attribute '__weakref__

''...
```

This was caught by github issue #8296 for which we do a couple more tests:

```
sage: "__weakref__" in dir(A)
True
sage: 1.__weakref__
Traceback (most recent call last):
...
AttributeError: 'sage.rings.integer.Integer' object has no attribute '__weakref__
(continues on next page)
```

Caveat: When \_\_call\_\_ is not defined for instances, using A.\_\_call\_\_ yields the method \_\_call\_\_ of the class. We use a workaround but there is no guarantee for robustness.

sage: getattr\_from\_other\_class(1, A, "\_\_call\_\_") Traceback (most recent call last): ... AttributeError: 'sage.rings.integer.Integer' object has no attribute '\_\_call\_\_'...

```
sage.cpython.getattr.raw_getattr(obj, name)
```

Like getattr (obj, name) but without invoking the binding behavior of descriptors under normal attribute access. This can be used to easily get unbound methods or other descriptors.

This ignores \_\_getattribute\_\_ hooks but it does support \_\_getattr\_\_.

**Note:** For Cython classes, \_\_getattr\_\_ is actually implemented as \_\_getattribute\_\_, which means that it is not supported by raw\_getattr.

## **EXAMPLES:**

```
sage: class X:
....: @property
        def prop(self):
. . . . :
             return 42
        def method(self):
. . . . :
            pass
. . . . :
        def __getattr__(self, name):
. . . . :
             return "magic " + name
sage: raw_getattr(X, "prop")
property object at ...>
sage: raw_getattr(X, "method")
<function ...method at ...>
sage: raw_getattr(X, "attr")
Traceback (most recent call last):
AttributeError: '...' object has no attribute 'attr'...
sage: x = X()
sage: raw_getattr(x, "prop")
property object at ...>
sage: raw_getattr(x, "method")
<function ...method at ...>
sage: raw_getattr(x, "attr")
'magic attr'
sage: x.__dict__["prop"] = 'no'
sage: x.__dict__["method"] = 'yes'
sage: x.__dict__["attr"] = 'ok'
sage: raw_getattr(x, "prop")
property object at ...>
```

```
sage: raw_getattr(x, "method")
'yes'
sage: raw_getattr(x, "attr")
'ok'
```

The same tests with an inherited new-style class:

```
sage: class Y(X, object):
. . . . :
        pass
sage: raw_getattr(Y, "prop")
property object at ...>
sage: raw_getattr(Y, "method")
<function ...method at ...>
sage: raw_getattr(Y, "attr")
Traceback (most recent call last):
AttributeError: '...' object has no attribute 'attr'...
sage: y = Y()
sage: raw_getattr(y, "prop")
sage: raw_getattr(y, "method")
<function ...method at ...>
sage: raw_getattr(y, "attr")
'magic attr'
sage: y.__dict__["prop"] = 'no'
sage: y.__dict__["method"] = 'yes'
sage: y.__dict__["attr"] = 'ok'
sage: raw_getattr(y, "prop")
property object at ...>
sage: raw_getattr(y, "method")
'yes'
sage: raw_getattr(y, "attr")
'ok'
```

**CHAPTER** 

**FIVE** 

## METACLASSES FOR CYTHON EXTENSION TYPES

Cython does not support metaclasses, but this module can be used to implement metaclasses for extension types.

**Warning:** This module has many caveats and you can easily get segfaults if you make a mistake. It relies on undocumented Python and Cython behaviour, so things might break in future versions.

## 5.1 How to use

To enable this metaclass mechanism, you need to put cimport sage.cpython.cython\_metaclass in your module (in the .pxd file if you are using one).

In the extension type (a.k.a. cdef class) for which you want to define a metaclass, define a method \_\_get-metaclass\_\_ with a single unused argument, and turn off the Cython directive always\_allow\_keywords. This method should return a type to be used as metaclass:

```
cimport cython
cimport sage.cpython.cython_metaclass
cdef class MyCustomType():
    @cython.always_allow_keywords(False)
    def __getmetaclass__(_):
        from foo import MyMetaclass
        return MyMetaclass
```

**Warning:** \_\_getmetaclass\_\_ must be defined as an ordinary method taking a single argument, but this argument should not be used in the method (it will be None).

When a type cls is being constructed with metaclass meta, then meta.\_\_init\_\_(cls, None, None, None) is called from Cython. In Python, this would be meta.\_\_init\_\_(cls, name, bases, dict).

Warning: The \_\_getmetaclass\_\_ method is called while the type is being created during the import of the module. Therefore, \_\_getmetaclass\_\_ should not refer to any global objects, including the type being created or other types defined or imported in the module (unless you are very careful). Note that non-imported cdef functions are not Python objects, so those are safe to call.

The same warning applies to the \_\_init\_\_ method of the metaclass.

**Warning:** The \_\_new\_\_ method of the metaclass (including the \_\_cinit\_\_ method for Cython extension types) is never called if you're using this from Cython. In particular, the metaclass cannot have any attributes or virtual methods.

#### **EXAMPLES:**

```
sage: cython(
                                                                                      #__
→needs sage.misc.cython
....: 111
....: cimport cython
....: cimport sage.cpython.cython_metaclass
....: cdef class MyCustomType():
        @cython.always_allow_keywords(False)
         def __getmetaclass__(_):
             class MyMetaclass(type):
                def __init__(*args):
                     print("Calling MyMetaclass.__init__{{}}".format(args))
              return MyMetaclass
. . . . :
....: cdef class MyDerivedType (MyCustomType):
. . . . :
          pass
....: ''')
Calling MyMetaclass.__init__(<class '...MyCustomType'>, None, None, None)
Calling MyMetaclass.__init__(<class '...MyDerivedType'>, None, None, None)
sage: MyCustomType.__class_
→needs sage.misc.cython
<class '...MyMetaclass'>
sage: class MyPythonType (MyDerivedType):
→needs sage.misc.cython
         pass
Calling MyMetaclass.__init__(<class '...MyPythonType'>, 'MyPythonType', (<class '...
→MyDerivedType'>,), {...})
```

# 5.2 Implementation

All this is implemented by defining

```
#define PyTypeReady(t) Sage_PyType_Ready(t)
```

and then implementing the function <code>Sage\_PyType\_Ready(t)</code> which first calls <code>PyType\_Ready(t)</code> and then handles the metaclass stuff.

**CHAPTER** 

SIX

## **SLOT WRAPPERS**

A slot wrapper is installed in the dict of an extension type to access a special method implemented in C. For example, object.\_\_init\_\_ or Integer.\_\_lt\_\_. Note that slot wrappers are always unbound (there is a bound variant called method-wrapper).

#### **EXAMPLES:**

```
sage: int.__add__
<slot wrapper '__add__' of 'int' objects>
```

Pure Python classes have normal methods, not slot wrappers:

```
sage: class X():
....:     def __add__(self, other):
....:     return NotImplemented
sage: X.__add__
<function X.__add__ at ...>
```

sage.cpython.wrapperdescr.wrapperdescr\_call(slotwrapper, self, \*args, \*\*kwds)

Call a slot wrapper without any type checks.

The main reason to use this is to call arithmetic slots like \_\_mul\_\_ without having to worry about whether to call T.\_\_mul\_\_(a, b) or T.\_\_rmul\_\_(b, a).

### INPUT:

- slotwrapper a slot wrapper (for example int.\_\_add\_\_).
- self the first positional argument. Normally, this should be of the correct type (an int when calling int.\_\_add\_\_). However, this check is skipped: you can pass an arbitrary object.
- \*args, \*\*kwds further arguments.

**Warning:** Since this skips type checks, it can easily crash Python if used incorrectly.

## **EXAMPLES:**

```
sage: from sage.cpython.wrapperdescr import wrapperdescr_call
sage: wrapperdescr_call(Integer.__mul__, 6, 9)
54
sage: wrapperdescr_call(Integer.__mul__, 7/5, 9)
63/5
sage: from sage.structure.element import Element
sage: wrapperdescr_call(Element.__mul__, 6, 9)
```

## DELETE ITEM FROM PYDICT BY EXACT VALUE AND HASH

Beware that the implementation of the routine here relies on implementation details of CPython's dict that go beyond the published API.

## **AUTHORS**:

• Nils Bruin (2017-05)

```
sage.cpython.dict_del_by_value.test_del_dictitem_by_exact_value(D, value, h)
```

This function helps testing some cdef function used to delete dictionary items.

## INPUT:

- D a Python <dict>.
- value an object that is value D.
- h the hash of the key under which to find value in D.

The underlying cdef function deletes an item from D that is in the hash bucket determined by h and whose value is identic with value. Of course, this only makes sense if the pairs (h, value) corresponding to items in D are pair-wise distinct.

If a matching item cannot be found, the function does nothing and silently returns.

# CHAPTER

# **EIGHT**

# **INDICES AND TABLES**

- Index
- Module Index
- Search Page

# **PYTHON MODULE INDEX**

## С

```
sage.cpython.atexit, 3
sage.cpython.cython_metaclass, 17
sage.cpython.debug, 7
sage.cpython.dict_del_by_value, 21
sage.cpython.getattr, 11
sage.cpython.string, 5
sage.cpython.wrapperdescr, 19
```

26 Python Module Index

## **INDEX**

```
Α
                                                 sage.cpython.cython_metaclass
                                                     module, 17
AttributeErrorMessage
                                 (class
                                             in
                                                 sage.cpython.debug
        sage.cpython.getattr), 11
                                                     module, 7
В
                                                 sage.cpython.dict_del_by_value
                                                     module, 21
bytes_to_str() (in module sage.cpython.string), 5
                                                 sage.cpython.getattr
C
                                                     module, 11
                                                 sage.cpython.string
cls (sage.cpython.getattr.AttributeErrorMessage attribute),
                                                     module, 5
        11
                                                 sage.cpython.wrapperdescr
                                                     module, 19
D
                                                 shortrepr() (in module sage.cpython.debug), 8
dir_with_other_class()
                                         module
                                 (in
                                                 str_to_bytes() (in module sage.cpython.string), 5
        sage.cpython.getattr), 11
G
                                                 test_del_dictitem_by_exact_value()
                                                                                              (in
getattr_debug() (in module sage.cpython.debug), 7
                                                         module sage.cpython.dict_del_by_value), 21
getattr_from_other_class()
                                         module
                                                 type_debug() (in module sage.cpython.debug), 8
        sage.cpython.getattr), 12
                                                 W
M
                                                                                           module
                                                 wrapperdescr call()
                                                                                (in
module
                                                         sage.cpython.wrapperdescr), 19
    sage.cpython.atexit,3
    sage.cpython.cython_metaclass, 17
    sage.cpython.debug, 7
    sage.cpython.dict_del_by_value, 21
    sage.cpython.getattr, 11
    sage.cpython.string,5
    sage.cpython.wrapperdescr, 19
Ν
name
       (sage.cpython.getattr. Attribute Error Message \ )
        tribute), 11
R
raw_getattr() (in module sage.cpython.getattr), 14
restore_atexit (class in sage.cpython.atexit), 3
S
sage.cpython.atexit
    module, 3
```