<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Utilities for interfacing with the standard library’s atexit module.</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>String &lt;-&gt; bytes encoding/decoding</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Various functions to debug Python internals</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Variants of getattr()</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>Metaclasses for Cython extension types</td>
<td>17</td>
</tr>
<tr>
<td>5.1</td>
<td>How to use</td>
<td>17</td>
</tr>
<tr>
<td>5.2</td>
<td>Implementation</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>Slot wrappers</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>Delete item from PyDict by exact value and hash</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>Indices and Tables</td>
<td>23</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td></td>
<td>27</td>
</tr>
</tbody>
</table>
SageMath has various modules to provide access to low-level Python internals.
UTILITIES FOR INTERFACING WITH THE STANDARD LIBRARY’S ATEXIT MODULE.

```python
class 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:

```
We test the `run` option:

```python
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:

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

**STRING <-> BYTES ENCODING/DECODING**

```python
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:**

```python
sage: from sage.cpython.string import bytes_to_str
sage: s = bytes_to_str(b'\xcf\x80')
sage: s == u'\u03c0'
True
sage: bytes_to_str([])
Traceback (most recent call last):
  ...TypeError: expected bytes, list found
```

```python
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:**

```python
sage: from sage.cpython.string import str_to_bytes
sage: bs = [str_to_bytes(u'\u03c0')]
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:

```python
sage: _ = getattr_debug(list, "reverse")
getattr_debug(obj=<type 'list'>, name='reverse'):
  type(obj) = <type 'type'>
  object has __dict__ slot (<type 'dict'>)
  did not find 'reverse' in MRO classes
  found 'reverse' in object __dict__
  returning <method 'reverse' of 'list' objects> (<type 'method_descriptor'>)

sage: _ = getattr_debug([], "reverse")
getattr_debug(obj=[], name='reverse'):
  type(obj) = <type 'list'>
  object does not have __dict__ slot
  found 'reverse' in dict of <type 'list'>
  got <method 'reverse' of 'list' objects> (<type 'method_descriptor'>)
  attribute is ordinary descriptor (has __get__)
  calling __get__()
  returning <built-in method reverse of list object at 0x... (<type 'builtin_function_or_method'>)

sage: _ = getattr_debug([], "__doc__")
getattr_debug(obj=[], name='__doc__'):
  type(obj) = <type 'list'>
  object does not have __dict__ slot
  found '__doc__' in dict of <type 'list'>
  got ...
  returning ...

sage: _ = getattr_debug(gp(1), "log")
getattr_debug(obj=1, name='log'):
  type(obj) = <class 'sage.interfaces.gp.GpElement'>
  object has __dict__ slot (<type 'dict'>)
  did not find 'log' in MRO classes
```

(continues on next page)
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 (<type '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 (<type 'int'>)
sage: _ = getattr_debug(1, 'foo')
Traceback (most recent call last):
... AttributeError: 'sage.rings.integer.Integer' object has no attribute 'foo'
sage: _ = getattr_debug(1, 'foo', 'xyz')
getattr_debug(obj=1, name='foo'):
type(obj) = <type '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!Hello world!'
sage: print(shortrepr("Hello world!" * 5))
'Hello world!Hello world!Hello world!Hello world!Hello world!'

sage.cpython.debug.type_debug(cls)
Print all internals of the type cls

EXAMPLES:

sage: type_debug(object)  # random
<type 'object'> (0x7fc57da7f040)
  ob_refcnt: 9739
  ob_type: <type 'type'>
  tp_name: object
  tp_basicsize: 16
  tp_itemsize: 0
  tp_dictoffset: 0
  tp_weaklistoffset: 0
  tp_bases (__bases__): tuple:
tp_mro ().__mro__): tuple:
  type '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
CHAPTER
FOUR

VARIANTS OF GETATTR()

```python
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: 1.bla  # indirect doctest
    Traceback (most recent call last):
    ...
    AttributeError: 'sage.rings.integer.Integer' object has no attribute 'bla'
    >>> sage: QQ[x].gen().bla
    Traceback (most recent call last):
    ...
    AttributeError: 'sage.rings.polynomial.polynomial_rational_flint.Polynomial_rational_flint' object has no attribute 'bla'
```

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

AUTHOR:
- Simon King (2011-05-21)

```python
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(object):
    ....:     a = 1
    ....:     b = 2
    ....:     c = 3
    (continues on next page)
```
sage: class B(object):
....:    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(object):
....:    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: cython("cdef class A:
...    cdef public int a")
sage: cython("cdef class B:
...    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.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(object):
....:    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")
<bound method A.inc of 1>
sage: getattr_from_other_class(1, A, "inc")()
2

Static methods work:

```python
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`:

```python
getattr_from_other_class(1, A, "lazy_attribute")
```

Traceback (most recent call last):
...
AttributeError: 'sage.rings.integer.Integer' object has no attribute 'lazy_attribute'

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

```python
getattr_from_other_class(ZZ, A, "lazy_attribute")
'Integer Ring'
```

```python
getattr_from_other_class(PolynomialRing(QQ, name='x', sparse=True).one(), A, "lazy_attribute")
'1'
```

```python
getattr_from_other_class(17, A, "lazy_attribute")
```

Traceback (most recent call last):
...
AttributeError: 'sage.rings.integer.Integer' object has no attribute 'lazy_attribute'

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

```python
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:

```python
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 trac ticket #8296 for which we do a couple more tests:

```python
__weakref__ in dir(A)
True
```

```python
__weakref__ in dir(1)  # py2
False
```

```python
1.__weakref__
```

Traceback (most recent call last):
...
AttributeError: 'sage.rings.integer.Integer' object has no attribute '__weakref__'

```python
n = 1
```

```python
ip = get_ipython()  # not tested: only works in interactive shell
```

```python
ip.magic_psearch('n.N')  # not tested: only works in interactive shell
```

(continues on next page)
Caveat: When \texttt{\_call\_} is not defined for instances, using \texttt{A.\_call\_} yields the method \texttt{\_call\_} of the class. We use a workaround but there is no guarantee for robustness.

\begin{verbatim}
A:\_call\_

global x
x = A()
x.__dict__["\_call\_"]
raise AttributeError("")
\end{verbatim}

\begin{verbatim}
sage: A.__call__
\texttt{\_call\_} of \texttt{\_call\_}
\end{verbatim}

Like \texttt{getattr} 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 \texttt{\_getattribute\_} hooks but it does support \texttt{\_getattribute\_}.

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

\begin{verbatim}
EXAMPLES:

\begin{verbatim}
raw\_getattr(X, "prop")
\end{verbatim}

\begin{verbatim}
x = X()
raw\_getattr(x, "prop")
\end{verbatim}

\begin{verbatim}
x.__dict__["prop"]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_"]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
x.__dict__["\_call\_\
\_call\_""]
\end{verbatim}

\begin{verbatim}
```python
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")
<property object at ...>
_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'
```
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 `__getmetaclass__` with a single unused argument. This method should return a type to be used as metaclass:

```cython
from foo import MyMetaclass
cdef class MyCustomType(object):
    def __getmetaclass__(self):
        return MyMetaclass
```

The above `__getmetaclass__` method is analogous to `__metaclass__ = MyMetaclass` in Python 2.

**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 \_
c\_init\_
\_ 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(''
    ....: cimport sage.cpython.cython_metaclass
    ....: cdef class MyCustomType(object):
    ....:     def __getmetaclass__(\_):
    ....:         class MyMetaclass(type):
    ....:             def __init__(\*args):
    ....:                 print("Calling MyMetaclass.__init__\{\}".format(args))
    ....:             return MyMetaclass
    ....:     
    ....:     cdef class MyDerivedType(MyCustomType):
    ....:         pass
    ....: 
    Calling MyMetaclass.__init__(<type \'...MyCustomType\'>, None, None, None)
    Calling MyMetaclass.__init__(<type \'...MyDerivedType\'>, None, None, None)
    sage: MyCustomType.__class__
    <class \'...MyMetaclass\'>
    sage: class MyPythonType(MyDerivedType):
    ....:     pass
    Calling MyMetaclass.__init__(<class \'...MyPythonType\'>, 'MyPythonType', (<type \'...
    _MyDerivedType\'>,), {...})
```

5.2 Implementation

All this is implemented by defining

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

and then implementing the function `Sage_PyType_Ready(t)` which first calls `PyType_Ready(t)` 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:**

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

Pure Python classes have normal methods, not slot wrappers:

```python
sage: class X(object):
    ...:     def __add__(self, other):
    ...:         return NotImplemented
sage: X.__add__
# py2
<unbound method X.__add__>
sage: X.__add__
# py3
<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:**

```python
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)
54
sage: wrapperdescr_call(Element.__mul__, 7/5, 9)
63/5
sage: from sage.numerical.mip import MixedIntegerLinearProgram
sage: wrapperdescr_call(type.__call__, MixedIntegerLinearProgram,
    maximization=False)
Mixed Integer Program (no objective, 0 variables, 0 constraints)
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.init_lookdict()
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 identical 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

C
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
INDEX

A
AttributeErrorMessage (class in sage.cpython.getattr), 11

B
bytes_to_str() (in module sage.cpython.string), 5

C
cls (sage.cpython.getattr.AttributeErrorMessage attribute), 11

D
dir_with_other_class() (in module sage.cpython.getattr), 11

G
getattr_debug() (in module sage.cpython.debug), 7
getattr_from_other_class() (in module sage.cpython.getattr), 12

I
init_lookdict() (in module sage.cpython.dict_del_by_value), 21

M
module
   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

N
name (sage.cpython.getattr.AttributeErrorMessage attribute), 11

R
raw_getattr() (in module sage.cpython.getattr), 14
restore_atexit (class in sage.cpython.atexit), 3
S
sage.cpython.atexit
    module, 3
sage.cpython.cython_metaclass
    module, 17
sage.cpython.debug
    module, 7
sage.cpython.dict_del_by_value
    module, 21
sage.cpython.getattr
    module, 11
sage.cpython.string
    module, 5
sage.cpython.wrapperdescr
    module, 19
shortrepr() (in module sage.cpython.debug), 8
str_to_bytes() (in module sage.cpython.string), 5

T
test_del_dictitem_by_exact_value() (in module sage.cpython.dict_del_by_value), 21
type_debug() (in module sage.cpython.debug), 8

W
wrapperdescr_call() (in module sage.cpython.wrapperdescr), 19