## Contents

1 Indices and tables 3

2 Package contents 5
   2.1 Constants .................................................. 5
   2.2 Object model compatibility ................................ 6
   2.3 Syntax compatibility ......................................... 7
   2.4 Binary and text data .......................................... 9
   2.5 unittest assertions .......................................... 10
   2.6 Renamed modules and attributes compatibility ............ 10

Python Module Index 17

Index 19
Six provides simple utilities for wrapping over differences between Python 2 and Python 3. It is intended to support codebases that work on both Python 2 and 3 without modification. six consists of only one Python file, so it is painless to copy into a project.

Six can be downloaded on PyPI. Its bug tracker and code hosting is on GitHub.

The name, “six”, comes from the fact that 2*3 equals 6. Why not addition? Multiplication is more powerful, and, anyway, “five” has already been snatched away by the (admittedly now moribund) Zope Five project.
CHAPTER 1

Indices and tables

- genindex
- search
2.1 Constants

Six provides constants that may differ between Python versions. Ones ending _types are mostly useful as the second argument to isinstance or issubclass.

**six.class_types**
Possible class types. In Python 2, this encompasses old-style types.ClassType and new-style type classes. In Python 3, this is just type.

**six.integer_types**
Possible integer types. In Python 2, this is long and int, and in Python 3, just int.

**six.string_types**
Possible types for text data. This is basestring() in Python 2 and str in Python 3.

**six.text_type**
Type for representing (Unicode) textual data. This is unicode() in Python 2 and str in Python 3.

**six.binary_type**
Type for representing binary data. This is str in Python 2 and bytes in Python 3. Python 2.6 and 2.7 include bytes as a builtin alias of str, so six’s version is only necessary for Python 2.5 compatibility.

**six.MAXSIZE**
The maximum size of a container like list or dict. This is equivalent to sys.maxsize in Python 2.6 and later (including 3.x). Note, this is temptingly similar to, but not the same as sys.maxint in Python 2. There is no direct equivalent to sys.maxint in Python 3 because its integer type has no limits aside from memory.

Here’s example usage of the module:
import six

def dispatch_types(value):
    if isinstance(value, six.integer_types):
        handle_integer(value)
    elif isinstance(value, six.class_types):
        handle_class(value)
    elif isinstance(value, six.string_types):
        handle_string(value)

2.2 Object model compatibility

Python 3 renamed the attributes of several interpreter data structures. The following accessors are available. Note that
the recommended way to inspect functions and methods is the stdlib inspect module.

six.get_unbound_function(meth)
Get the function out of unbound method meth. In Python 3, unbound methods don’t exist, so this function just returns
meth unchanged. Example usage:

```python
from six import get_unbound_function

class X(object):
    def method(self):
        pass

method_function = get_unbound_function(X.method)
```

six.get_method_function(meth)
Get the function out of method object meth.

six.get_method_self(meth)
Get the self of bound method meth.

six.get_function_closure(func)
Get the closure (list of cells) associated with func. This is equivalent to func.__closure__ on Python 2.6+ and
func.func_closure on Python 2.5.

six.get_function_code(func)
Get the code object associated with func. This is equivalent to func.__code__ on Python 2.6+ and func.
func_code on Python 2.5.

six.get_function_defaults(func)
Get the defaults tuple associated with func. This is equivalent to func.__defaults__ on Python 2.6+ and func.
func_defaults on Python 2.5.

six.get_function_globals(func)
Get the globals of func. This is equivalent to func.__globals__ on Python 2.6+ and func.
func_globals on Python 2.5.

six.next(it)
six.advance_iterator(it)
Get the next item of iterator it. StopIteration is raised if the iterator is exhausted. This is a replacement
for calling it.next() in Python 2 and next(it) in Python 3. Python 2.6 and above have a builtin next
function, so six’s version is only necessary for Python 2.5 compatibility.

six.callable(obj)
Check if obj can be called. Note callable has returned in Python 3.2, so using six’s version is only necessary
when supporting Python 3.0 or 3.1.
six.iterkeys(dictionary, **kwargs)
Returns an iterator over dictionary’s keys. This replaces dictionary.iterkeys() on Python 2 and
dictionary.keys() on Python 3. kwargs are passed through to the underlying method.

six.itervalues(dictionary, **kwargs)
Returns an iterator over dictionary’s values. This replaces dictionary.itervalues() on Python 2 and
dictionary.values() on Python 3. kwargs are passed through to the underlying method.

six.iteritems(dictionary, **kwargs)
Returns an iterator over dictionary’s items. This replaces dictionary.iteritems() on Python 2 and
dictionary.items() on Python 3. kwargs are passed through to the underlying method.

six.iterlists(dictionary, **kwargs)
Calls dictionary.iterlists() on Python 2 and dictionary.lists() on Python 3. No builtin Python mapping type has such a method; this method is intended for use with multi-valued dictionaries like Werkzeug’s. kwargs are passed through to the underlying method.

six.viewkeys(dictionary)
Return a view over dictionary’s keys. This replaces dict.viewkeys() on Python 2.7 and dict.keys() on Python 3.

six.viewvalues(dictionary)
Return a view over dictionary’s values. This replaces dict.viewvalues() on Python 2.7 and dict.values() on Python 3.

six.viewitems(dictionary)
Return a view over dictionary’s items. This replaces dict.viewitems() on Python 2.7 and dict.items() on Python 3.

six.create_bound_method(func, obj)
Return a method object wrapping func and bound to obj. On both Python 2 and 3, this will return a types.MethodType object. The reason this wrapper exists is that on Python 2, the MethodType constructor requires the obj’s class to be passed.

six.create_unbound_method(func, cls)
Return an unbound method object wrapping func. In Python 2, this will return a types.MethodType object. In Python 3, unbound methods do not exist and this wrapper will simply return func.

class six.Iterator
A class for making portable iterators. The intention is that it be subclassed and subclasses provide a __next__ method. In Python 2, Iterator has one method: next. It simply delegates to __next__. An alternate way to do this would be to simply alias next to __next__. However, this interacts badly with subclasses that override __next__. Iterator is empty on Python 3. (In fact, it is just aliased to object.)

@six.wraps(wrapped, assigned=functools.WRAPPER_ASSIGNMENTS, updated=functools.WRAPPER_UPDATES)
This is Python 3.2’s functools.wraps() decorator. It sets the __wrapped__ attribute on what it decorates. It doesn’t raise an error if any of the attributes mentioned in assigned and updated are missing on wrapped object.

2.3 Syntax compatibility

These functions smooth over operations which have different syntaxes between Python 2 and 3.

six.exec_(code, globals=None, locals=None)
Execute code in the scope of globals and locals. code can be a string or a code object. If globals or locals are not given, they will default to the scope of the caller. If just globals is given, it will also be used as locals.
six Documentation, Release 1.14.0

Note: Python 3’s `exec()` doesn’t take keyword arguments, so calling `exec()` with them should be avoided.

```
six.print_(*args, *, file=sys.stdout, end="n", sep=" ", flush=False)
```

Print `args` into `file`. Each argument will be separated with `sep` and `end` will be written to the file after the last argument is printed. If `flush` is true, `file.flush()` will be called after all data is written.

Note: In Python 2, this function imitates Python 3’s `print()` by not having softspace support. If you don’t know what that is, you’re probably ok. :)

```
six.raise_from(exc_value, exc_value_from)
```

Raise an exception from a context. On Python 3, this is equivalent to `raise exc_value from exc_value_from`. On Python 2, which does not support exception chaining, it is equivalent to `raise exc_value`.

```
six.reraise(exc_type, exc_value, exc_traceback=None)
```

Reraise an exception, possibly with a different traceback. In the simple case, `reraise(*sys.exc_info())` with an active exception (in an except block) reraises the current exception with the last traceback. A different traceback can be specified with the `exc_traceback` parameter. Note that since the exception reraising is done within the `reraise()` function, Python will attach the call frame of `reraise()` to whatever traceback is raised.

```
six.with_metaclass(metaclass, *bases)
```

Create a new class with base classes `bases` and metaclass `metaclass`. This is designed to be used in class declarations like this:

```python
from six import with_metaclass

class Meta(type):
    pass

class Base(object):
    pass

class MyClass(with_metaclass(Meta, Base)):
    pass
```

Another way to set a metaclass on a class is with the `add_metaclass()` decorator.

```
@six.add_metaclass(metaclass)
```

Class decorator that replaces a normally-constructed class with a metaclass-constructed one. Example usage:

```python
@add_metaclass(Meta)
class MyClass(object):
    pass
```

That code produces a class equivalent to

```python
class MyClass(object, metaclass=Meta):
    pass
```

on Python 3 or

```python
class MyClass(object):
    __metaclass__ = Meta
```
on Python 2.

Note that class decorators require Python 2.6. However, the effect of the decorator can be emulated on Python 2.5 like so:

```python
class MyClass(object):
    pass
MyClass = add_metaclass(Meta)(MyClass)
```

## 2.4 Binary and text data

Python 3 enforces the distinction between byte strings and text strings far more rigorously than Python 2 does; binary data cannot be automatically coerced to or from text data. six provides several functions to assist in classifying string data in all Python versions.

**six.b(data)**

A “fake” bytes literal. *data* should always be a normal string literal. In Python 2, `b()` returns an 8-bit string. In Python 3, *data* is encoded with the latin-1 encoding to bytes.

**Note:** Since all Python versions 2.6 and after support the `b` prefix, code without 2.5 support doesn’t need `b()`.

**six.u(text)**

A “fake” unicode literal. *text* should always be a normal string literal. In Python 2, `u()` returns unicode, and in Python 3, a string. Also, in Python 2, the string is decoded with the `unicode-escape` codec, which allows unicode escapes to be used in it.

**Note:** In Python 3.3, the `u` prefix has been reintroduced. Code that only supports Python 3 versions of 3.3 and higher thus does not need `u()`.

**Note:** On Python 2, `u()` doesn’t know what the encoding of the literal is. Each byte is converted directly to the unicode codepoint of the same value. Because of this, it’s only safe to use `u()` with strings of ASCII data.

**six.unichr(c)**

Return the (Unicode) string representing the codepoint *c*. This is equivalent to `unichr()` on Python 2 and `chr()` on Python 3.

**six.int2byte(i)**

Converts *i* to a byte. *i* must be in `range(0, 256)`. This is equivalent to `chr()` in Python 2 and `bytes((i,))` in Python 3.

**six.byte2int(bs)**

Converts the first byte of *bs* to an integer. This is equivalent to `ord(bs[0])` on Python 2 and `bs[0]` on Python 3.

**six.indexbytes(buf, i)**

Return the byte at index *i* of *buf* as an integer. This is equivalent to indexing a bytes object in Python 3.

**six.iterbytes(buf)**

Return an iterator over bytes in *buf* as integers. This is equivalent to a bytes object iterator in Python 3.

**six.ensure_binary(s, encoding='utf-8', errors='strict')**

Coerce *s* to `binary_type`. *encoding*, *errors* are the same as `str.encode()`
six Documentation, Release 1.14.0

six.**ensure_str**(s, encoding='utf-8', errors='strict')

Coerce s to str. encoding, errors are the same as str.encode()

six.**ensure_text**(s, encoding='utf-8', errors='strict')

Coerce s to text_type. encoding, errors are the same as bytes.decode()

six.**StringIO**

This is a fake file object for textual data. It’s an alias for StringIO.StringIO in Python 2 and io.StringIO in Python 3.

six.**BytesIO**

This is a fake file object for binary data. In Python 2, it’s an alias for StringIO.StringIO, but in Python 3, it’s an alias for io.BytesIO.

@six.**python_2_unicode_compatible**

A class decorator that takes a class defining a __str__ method. On Python 3, the decorator does nothing. On Python 2, it aliases the __str__ method to __unicode__ and creates a new __str__ method that returns the result of __unicode__() encoded with UTF-8.

2.5 unittest assertions

Six contains compatibility shims for unittest assertions that have been renamed. The parameters are the same as their aliases, but you must pass the test method as the first argument. For example:

```python
import six
import unittest

class TestAssertCountEqual(unittest.TestCase):
    def test(self):
        six.assertCountEqual(self, (1, 2), [2, 1])
```

Note these functions are only available on Python 2.7 or later.

six.**assertCountEqual**()

Alias for assertCountEqual() on Python 3 and assertItemsEqual() on Python 2.

six.**assertRaisesRegex**()

Alias for assertRaisesRegex() on Python 3 and assertRaisesRegexp() on Python 2.

six.**assertRegex**()

Alias for assertRegex() on Python 3 and assertRegexpMatches() on Python 2.

six.**assertNotRegex**()

Alias for assertNotRegex() on Python 3 and assertNotRegexpMatches() on Python 2.

2.6 Renamed modules and attributes compatibility

Python 3 reorganized the standard library and moved several functions to different modules. Six provides a consistent interface to them through the fake six.moves module. For example, to load the module for parsing HTML on Python 2 or 3, write:

```python
from six.moves import html_parser
```

Similarly, to get the function to reload modules, which was moved from the builtin module to the importlib module, use:

```python
import six
six.import_cloaked_module('importlib')
```
from six.moves import reload_module

For the most part, six.moves aliases are the names of the modules in Python 3. When the new Python 3 name is a package, the components of the name are separated by underscores. For example, html.parser becomes html_parser. In some cases where several modules have been combined, the Python 2 name is retained. This is so the appropriate modules can be found when running on Python 2. For example, BaseHTTPServer which is in http.server in Python 3 is aliased as BaseHTTPServer.

Some modules which had two implementations have been merged in Python 3. For example, cPickle no longer exists in Python 3; it was merged with pickle. In these cases, fetching the fast version will load the fast one on Python 2 and the merged module in Python 3.

The urllib, urllib2, and urlparse modules have been combined in the urllib package in Python 3. The six.moves.urllib package is a version-independent location for this functionality; its structure mimics the structure of the Python 3 urllib package.

Note: In order to make imports of the form:

```python
from six.moves.cPickle import loads
```

work, six places special proxy objects in sys.modules. These proxies lazily load the underlying module when an attribute is fetched. This will fail if the underlying module is not available in the Python interpreter. For example, sys.modules["six.moves.winreg"].LoadKey would fail on any non-Windows platform. Unfortunately, some applications try to load attributes on every module in sys.modules. six mitigates this problem for some applications by pretending attributes on unimportable modules do not exist. This hack does not work in every case, though. If you are encountering problems with the lazy modules and don’t use any from imports directly from six.moves modules, you can workaround the issue by removing the six proxy modules:

```python
d = [name for name in sys.modules if name.startswith("six.moves.")]
for name in d:
    del sys.modules[name]
```

Supported renames:

<table>
<thead>
<tr>
<th>Name</th>
<th>Python 2 name</th>
<th>Python 3 name</th>
</tr>
</thead>
<tbody>
<tr>
<td>builtins</td>
<td><strong>builtin</strong></td>
<td>builtins</td>
</tr>
<tr>
<td>configparser</td>
<td>ConfigParser</td>
<td>configparser</td>
</tr>
<tr>
<td>copyreg</td>
<td>copy_reg</td>
<td>copyreg</td>
</tr>
<tr>
<td>cPickle</td>
<td>cPickle</td>
<td>pickle</td>
</tr>
<tr>
<td>cStringIO</td>
<td>cStringIO.StringIO()</td>
<td>io.StringIO</td>
</tr>
<tr>
<td>collections_abc</td>
<td>collections</td>
<td>collections.abc(3.3+)</td>
</tr>
<tr>
<td>dbm_gnu</td>
<td>gdbm</td>
<td>dbm.gnu</td>
</tr>
<tr>
<td>dbm_ndbm</td>
<td>dbm</td>
<td>dbm.ndbm</td>
</tr>
<tr>
<td>_dummy_thread</td>
<td>dummy_thread</td>
<td>_dummy_thread(&lt;3.9)_thread(3.9+)</td>
</tr>
<tr>
<td>email_mime_base</td>
<td>email.MIMEBase</td>
<td>email.mime.base</td>
</tr>
<tr>
<td>email_mime_image</td>
<td>email.MIMEImage</td>
<td>email.mime.image</td>
</tr>
<tr>
<td>email_mime.multipart</td>
<td>email.MIMEMultipart</td>
<td>email.mime.multipart</td>
</tr>
<tr>
<td>email_mime_nonmultipart</td>
<td>email.MIMENonMultipart</td>
<td>email.mime.nonmultipart</td>
</tr>
<tr>
<td>email_mime_text</td>
<td>email.MIMEText</td>
<td>email.mime.text</td>
</tr>
<tr>
<td>filter</td>
<td>itertools.ifilter()</td>
<td>filter()</td>
</tr>
<tr>
<td>filterfalse</td>
<td>itertools.ifilterfalse()</td>
<td>itertools.filterfalse()</td>
</tr>
<tr>
<td>getcwd</td>
<td>os.getcwdu()</td>
<td>os.getcwdu()</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Python 2 name</th>
<th>Python 3 name</th>
</tr>
</thead>
<tbody>
<tr>
<td>getcwdb</td>
<td>os.getcwd()</td>
<td>os.getcwdb()</td>
</tr>
<tr>
<td>getoutput</td>
<td>commands.getoutput()</td>
<td>subprocess.getoutput()</td>
</tr>
<tr>
<td>http_cookiejar</td>
<td>cookielib</td>
<td>http.cookiejar</td>
</tr>
<tr>
<td>http_cookies</td>
<td>Cookie</td>
<td>http.cookies</td>
</tr>
<tr>
<td>html_entities</td>
<td>htmlentitydefs</td>
<td>html.entities</td>
</tr>
<tr>
<td>html_parser</td>
<td>HTMLParser</td>
<td>html.parser</td>
</tr>
<tr>
<td>http_client</td>
<td>httplib</td>
<td>http.client</td>
</tr>
<tr>
<td>BaseHTTPServer</td>
<td>BaseHTTPServer</td>
<td>http.server</td>
</tr>
<tr>
<td>CGIHTTPServer</td>
<td>CGIHTTPServer</td>
<td>http.server</td>
</tr>
<tr>
<td>SimpleHTTPServer</td>
<td>SimpleHTTPServer</td>
<td>http.server</td>
</tr>
<tr>
<td>input</td>
<td>raw_input()</td>
<td>input()</td>
</tr>
<tr>
<td>intern</td>
<td>intern()</td>
<td>sys.intern()</td>
</tr>
<tr>
<td>map</td>
<td>itertools.imap()</td>
<td>map()</td>
</tr>
<tr>
<td>queue</td>
<td>Queue</td>
<td>queue</td>
</tr>
<tr>
<td>range</td>
<td>xrange()</td>
<td>range</td>
</tr>
<tr>
<td>reduce</td>
<td>reduce()</td>
<td>functools.reduce()</td>
</tr>
<tr>
<td>reload_module</td>
<td>reload()</td>
<td>importlib.reload()</td>
</tr>
<tr>
<td>reprlib</td>
<td>repr</td>
<td>importlib.reload()</td>
</tr>
<tr>
<td>shlex_quote</td>
<td>pipes.quote</td>
<td>shlex.quote</td>
</tr>
<tr>
<td>socketserver</td>
<td>SocketServer</td>
<td>socketserver</td>
</tr>
<tr>
<td>_thread</td>
<td>thread</td>
<td>_thread</td>
</tr>
<tr>
<td>tkinter</td>
<td>Tkinter</td>
<td>tkinter</td>
</tr>
<tr>
<td>tkinter_dialog</td>
<td>Dialog</td>
<td>tkinter.dialog</td>
</tr>
<tr>
<td>tkinter_filedialog</td>
<td>FileDialog</td>
<td>tkinter.FileDialog</td>
</tr>
<tr>
<td>tkinter_scrolledtext</td>
<td>ScrolledText</td>
<td>tkinter.scrolledtext</td>
</tr>
<tr>
<td>tkinter_simpledialog</td>
<td>SimpleDialog</td>
<td>tkinter.simpledialog</td>
</tr>
<tr>
<td>tkinter_ttk</td>
<td>ttk</td>
<td>tkinter.ttk</td>
</tr>
<tr>
<td>tkinter_tix</td>
<td>Tix</td>
<td>tkinter.tix</td>
</tr>
<tr>
<td>tkinter_constants</td>
<td>Tkconstants</td>
<td>tkinter.constants</td>
</tr>
<tr>
<td>tkinter_dnd</td>
<td>TkDnd</td>
<td>tkinter.dnd</td>
</tr>
<tr>
<td>tkinter_colorchooser</td>
<td>tkColorChooser</td>
<td>tkinter.colorchooser</td>
</tr>
<tr>
<td>tkinter_commondialog</td>
<td>tkCommonDialog</td>
<td>tkinter.commondialog</td>
</tr>
<tr>
<td>tkinter_tkfiledialog</td>
<td>tkFileDialog</td>
<td>tkinter.tkfiledialog</td>
</tr>
<tr>
<td>tkinter_font</td>
<td>tkFont</td>
<td>tkinter.font</td>
</tr>
<tr>
<td>tkinter_messagebox</td>
<td>tkMessageBox</td>
<td>tkinter.messagebox</td>
</tr>
<tr>
<td>tkinter_tksimpledialog</td>
<td>tksimpleDialog</td>
<td>tkinter.tksimpledialog</td>
</tr>
<tr>
<td>urllib.parse</td>
<td>See six.moves.urllib.parse</td>
<td>urllib.parse</td>
</tr>
<tr>
<td>urllib.error</td>
<td>See six.moves.urllib.error</td>
<td>urllib.error</td>
</tr>
<tr>
<td>urllib.request</td>
<td>See six.moves.urllib.request</td>
<td>urllib.request</td>
</tr>
<tr>
<td>urllib.response</td>
<td>See six.moves.urllib.response</td>
<td>urllib.response</td>
</tr>
<tr>
<td>urllib.robotparser</td>
<td>robotparser</td>
<td>urllib.robotparser</td>
</tr>
<tr>
<td>urllib_robotparser</td>
<td>robotparser</td>
<td>urllib.robotparser</td>
</tr>
<tr>
<td>UserDict</td>
<td>UserDict.UserDict</td>
<td>collections.UserDict</td>
</tr>
<tr>
<td>UserList</td>
<td>UserList.UserList</td>
<td>collections.UserList</td>
</tr>
<tr>
<td>winreg</td>
<td>_winreg</td>
<td>winreg</td>
</tr>
<tr>
<td>xmlrpc_client</td>
<td>xmlrpclib</td>
<td>xmlrpc.client</td>
</tr>
<tr>
<td>xmlrpc_server</td>
<td>SimpleXMLRPCServer</td>
<td>xmlrpc.server</td>
</tr>
<tr>
<td>xrange</td>
<td>xrange()</td>
<td>range</td>
</tr>
</tbody>
</table>
## 2.6.1 urllib.parse

Contains functions from Python 3’s `urllib.parse` and Python 2’s:

**urllib.parse:**

- `urllib.parse.ParseResult`
- `urllib.parse.SplitResult`
- `urllib.parse.urlparse()`
- `urllib.parse.urlunparse()`
- `urllib.parse.parse_qs()`
- `urllib.parse.parse_qsl()`
- `urllib.parse.urljoin()`
- `urllib.parse.urldefrag()`
- `urllib.parse.ursplit()`
- `urllib.parse.urlunsplit()`
- `urllib.parse.splitquery`
- `urllib.parse.uses_fragment`
- `urllib.parse.uses_netloc`
- `urllib.parse.uses_params`
- `urllib.parse.uses_query`
- `urllib.parse.uses_relative`

and **urllib:**

- `urllib.quote()`
- `urllib.quote_plus()`
- `urllib.split`
- `urllib.splituser`
- `urllib.splitvalue`
- `urllib.unquote()` *(also exposed as `urllib.parse.unquote_to_bytes()`)*
- `urllib.unquote_plus()`
- `urllib.urlencode()`
2.6.2 urllib error

Contains exceptions from Python 3’s `urllib.error` and Python 2’s:

```
urllib:
  • urllib.ContentTooShortError

and urllib2:
  • urllib2.URLError
  • urllib2.HTTPError
```

2.6.3 urllib request

Contains items from Python 3’s `urllib.request` and Python 2’s:

```
urllib:
  • urllib.pathname2url()
  • urllib.url2pathname()
  • urllib.getproxies()
  • urllib.urlretrieve()
  • urllib.urlcleanup()
  • urllib.URLopener
  • urllib.FancyURLopener
  • urllib.proxy_bypass

and urllib2:
  • urllib2.urlopen()
  • urllib2.install_opener()
  • urllib2.build_opener()
  • urllib2.parse_http_list
  • urllib2.parse_keqv_list
  • urllib2.Request
  • urllib2.OpenerDirector
  • urllib2.HTTPDefaultErrorHandler
  • urllib2.URLError
  • urllib2.HTTPRedirectHandler
  • urllib2.HTTPCookieProcessor
  • urllib2.ProxyHandler
  • urllib2.BaseHandler
  • urllib2.HTTPPasswordMgr
  • urllib2.HTTPPasswordMgrWithDefaultRealm
  • urllib2.AbstractBasicAuthHandler
```
• urllib2.HTTPBasicAuthHandler
• urllib2.ProxyBasicAuthHandler
• urllib2.AbstractDigestAuthHandler
• urllib2.HTTPDigestAuthHandler
• urllib2.ProxyDigestAuthHandler
• urllib2.HTTPHandler
• urllib2.HTTPSHandler
• urllib2.FileHandler
• urllib2.FTPHandler
• urllib2.CacheFTPHandler
• urllib2.UnknownHandler
• urllib2.HTTPErrorProcessor

2.6.4 urllib response

Contains classes from Python 3’s urllib.response and Python 2’s:

urllib:
• urllib.addbase
• urllib.addclosehook
• urllib.addinfo
• urllib.addinfourl

2.6.5 Advanced - Customizing renames

It is possible to add additional names to the six.moves namespace.

six.add_move(item)
Add item to the six.moves mapping. item should be a MovedAttribute or MovedModule instance.

six.remove_move(name)
Remove the six.moves mapping called name. name should be a string.

Instances of the following classes can be passed to add_move(). Neither have any public members.

class six.MovedModule(name, old_mod, new_mod)
Create a mapping for six.moves called name that references different modules in Python 2 and 3. old_mod is the name of the Python 2 module. new_mod is the name of the Python 3 module.

class six.MovedAttribute(name, old_mod, new_mod, old_attr=None, new_attr=None)
Create a mapping for six.moves called name that references different attributes in Python 2 and 3. old_mod is the name of the Python 2 module. new_mod is the name of the Python 3 module. If new_attr is not given, it defaults to old_attr. If neither is given, they both default to name.
S
six, ??
six.moves, 10
six.moves.urllib.error, 14
six.moves.urllib.parse, 13
six.moves.urllib.request, 14
six.moves.urllib.response, 15
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>M</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>T</td>
</tr>
</tbody>
</table>
U
u() (in module six), 9
unichr() (in module six), 9

V
viewitems() (in module six), 7
viewkeys() (in module six), 7
viewvalues() (in module six), 7

W
with_metaclass() (in module six), 8
wraps() (in module six), 7