Contents

1 Fixers 1
   1.1 Default ................................................................. 1
   1.2 Opt-in ........................................................................ 6

2 Purpose of the project ................................................. 7

3 A note about handling text literals ................................. 9

4 Indices and tables .......................................................... 11
Fixers come in two types: Default and Opt-in. Default fixers should not break code except for corner cases, and are idempotent (applying them more than once to given source code will make no changes after the first application). Opt-in fixers are allowed to break these rules.

Python 2 code from Python 2.6 and older will be upgraded to code that is compatible with Python 2.6, 2.7, and Python 3.

If code is using a feature unique to Python 2.7, it will not be downgraded to work with Python 2.6. For example, `dict.viewitems()` usage will not be removed to make the code compatible with Python 2.6.

Some fixers rely on the latest release of the six project to work (see Fixers requiring six). If you wish to turn off these fixers to avoid an external dependency on six, then use the --no-six flag.

Fixers use the API defined by 2to3. For details of how this works, and how to implement your own fixers, see Extending 2to3 with your own fixers, at python3porting.com. python-modernize will try to load fixers whose full dotted-path is specified as a -f argument, but will fail if they are not found. By default, fixers will not be found in the current directory; use --fixers-here to make python-modernize look for them there, or see the Python tutorial on modules (in particular, the parts on the search path and packages) for more info on how Python finds modules.

### 1.1 Default

A default fixer will be enabled when:

- Either no -f/--fix options are used, or -f default/--fix=default is used, or the fixer is listed explicitly in an -f/--fix option; and
- The fixer is not listed in an --nofix option; and
- For fixers that are dependent on the six project, --no-six is not specified (see Fixers requiring six).

The --nofix and --no-six options always override fixers specified using -f/--fix. The --six-unicode and --future-unicode options also disable fixers that are not applicable for those options.
1.1.1 Fixers requiring six

The six project provides the six module which contains various tidbits in helping to support Python 2/3 code. All six-related fixers assume the latest version of six is installed.

**basestring**

Replaces all references to `basestring()` with six.string_types.

New in version 0.4.

**dict_six**

Fixes various methods on the dict type for getting all keys, values, or items. E.g.:

```python
x.values()
x.itervalues()
x.viewvalues()
```

becomes:

```python
list(x.values())
six.itervalues(x)
six.viewvalues(x)
```

Care is taken to only call `list()` when not in an iterating context (e.g. not the iterable for a for loop).

**filter**

When a call to `filter` is discovered, from six.moves import filter is added to the module. Wrapping the use in a call to `list()` is done when necessary.

**imports_six**

Uses six.moves to fix various renamed modules, e.g.:

```python
import six.moves.configparser
ConfigParser.ConfigParser()
```

becomes:

```python
import six.moves.configparser
six.moves.configparser.ConfigParser()
```

The modules in Python 2 whose renaming in Python 3 is supported are:

- __builtin__
- _winreg
- BaseHTTPServer
- CGIHTTPServer
- ConfigParser
- copy_reg
- Cookie
- cookielib
- cPickle
- Dialog
- dummy_thread
• FileDialog
• gdbm
• htmlentitydefs
• HTMLParser
• httplib
• Queue
• repr
• robotparser
• ScrolledText
• SimpleDialog
• SimpleHTTPServer
• SimpleXMLRPCServer
• SocketServer
• thread
• Tix
• tkColorChooser
• tkCommonDialog
• Tkconstants
• Tkdnd
• tkFileDialog
• tkFont
• Tkinter
• tkMessageBox
• tkSimpleDialog
• ttk
• xmlrpclib

New in version 0.4.

**input**

Changes:

```python
input(x)
raw_input(x)
```

to:

```python
from six.moves import input
eval(input(x))
```

New in version 0.4.

1.1. Default
int_long_tuple
Changes (int, long) or (long, int) to six.integer_types.
New in version 0.4.

map
If a call to map is discovered, from six.moves import map is added to the module. Wrapping the use in a call to list() is done when necessary.

metaclass
Changes:

```python
class Foo:
    __metaclass__ = Meta
```
to:

```python
import six
class Foo(six.with_metaclass(Meta)):
    pass
```

See also:
six.with_metaclass()

raise_six
Changes raise E, V, T to six.reraise(E, V, T).

unicode_type
Changes all reference of unicode to six.text_type.

urllib_six
Changes:

```python
from urllib import quote_plus
quote_plus('hello world')
```
to:

```python
from six.moves.urllib.parse import quote_plus
quote_plus('hello world')
```

unichr
Changes all reference of unichr to six.unichr.

xrange_six
Changes:

```python
w = xrange(x)
y = range(z)
```
to:

```python
from six.moves import range
w = range(x)
y = list(range(z))
```
Care is taken not to call list() when range() is used in an iterating context.
zip

If `zip` is called, from six.moves import zip is added to the module. Wrapping the use in a call to `list()` is done when necessary.

1.1.2 2to3 fixers

Some fixers from lib2to3 in Python’s standard library are run by default unmodified as their transformations are Python 2 compatible.

- apply
- except
- exec
- execfile
- exitfunc
- funcattrs
- has_key
- idioms
- long
- methodattrs
- ne
- numliterals
- operator
- paren
- reduce
- repr
- set_literal
- standarderror
- sys_exc
- throw
- tuple_params
- types
- ws_comma
- xreadlines

1.1.3 Fixers with no dependencies

file

Changes all calls to file to open.

New in version 0.4.
import
Changes implicit relative imports to explicit relative imports and adds from __future__ import absolute_import.
New in version 0.4.

next
Changes all method calls to x.next() to next(x).

print
Changes all usage of the print statement to use the print() function and adds from __future__ import print_function.

raise
Changes comma-based raise statements from:

```python
raise E, V
raise ((E, E1), E2), E3), V
```
to:

```python
raise E(V)
raise E(V)
```

1.2 Opt-in

To specify an opt-in fixer while also running all the default fixers, make sure to specify the -f default or --fix=default option, e.g.:

```bash
python-modernize -f default -f libmodernize.fixes.fix_open
```

classic_division
When a use of the division operator – / – is found, add from __future__ import division and change the operator to //. If from __future__ import division is already present, this fixer is skipped.

This is intended for use in programs where / is conventionally only used for integer division, or where it is intended to do a manual pass after running python-modernize to look for cases that should not have been changed to //. The results of division on non-integers may differ after running this fixer: for example, 3.5 / 2 == 1.75, but 3.5 // 2 == 1.0.

Some objects may override the __div__ method for a use other than division, and thus would break when changed to use a __floordiv__ method instead.

This fixer is opt-in because it may change the meaning of code as described above.
New in version 1.0.

open
When a call to open is discovered, add from io import open at the top of the module so as to use io.open() instead. This fixer is opt-in because it changes what object is returned by a call to open().
New in version 0.4.
Purpose of the project

This library is a very thin wrapper around lib2to3 to utilize it to make Python 2 code more modern with the intention of eventually porting it over to Python 3.

The python-modernize command works like 2to3. Here’s how you’d rewrite a single file:

```
python-modernize -w example.py
```

See the LICENSE file for the license of python-modernize. Using this tool does not affect licensing of the modernized code.

The project website can be found on GitHub and the PyPI project name is modernize
A note about handling text literals

- By default modernize does not change Unicode literals at all, which means that you can take advantage of PEP 414. This is the simplest option if you only want to support Python 3.3 and above along with Python 2.

- Alternatively, there is the --six-unicode flag which will wrap Unicode literals with the six helper function `six.u()` using the `libmodernize.fixes.fix_unicode` fixer. This is useful if you want to support Python 3.1 and Python 3.2 without bigger changes.

- The last alternative is the --future-unicode flag which imports the `unicode_literals` from the `__future__` module using the `libmodernize.fixes.fix_unicode_future` fixer. This requires Python 2.6 and later, and will require that you mark bytestrings with `b''` and native strings in `str('')` or something similar that survives the transformation.
CHAPTER 4

Indices and tables

- genindex
- search
Index

B
basestring (2to3 fixer), 2

C
classic_division (2to3 fixer), 6

D
dict_six (2to3 fixer), 2

F
file (2to3 fixer), 5
filter (2to3 fixer), 2

I
import (2to3 fixer), 5
imports_six (2to3 fixer), 2
input_six (2to3 fixer), 3
int_long_tuple (2to3 fixer), 3

M
map (2to3 fixer), 4
metaclass (2to3 fixer), 4

N
next (2to3 fixer), 6

O
open (2to3 fixer), 6

P
print (2to3 fixer), 6

R
raise (2to3 fixer), 6
raise_six (2to3 fixer), 4

U
unichr (2to3 fixer), 4

unicode_type (2to3 fixer), 4
urllib_six (2to3 fixer), 4

X
xrange_six (2to3 fixer), 4

Z
zip (2to3 fixer), 4