pep8 documentation

Release 1.4.6

Florent Xicluna

July 02, 2013

CONTENTS

Python style guide checker

pep8 is a tool to check your Python code against some of the style conventions in PEP 8.

Contents:

CHAPTER

ONE

INTRODUCTION

pep8 is a tool to check your Python code against some of the style conventions in PEP 8.

1.1 Features

- Plugin architecture: Adding new checks is easy.
- Parseable output: Jump to error location in your editor.
- Small: Just one Python file, requires only stdlib. You can use just the pep8.py file for this purpose.
- Comes with a comprehensive test suite.

1.2 Disclaimer

This utility does not enforce every single rule of PEP 8. It helps to verify that some coding conventions are applied but it does not intend to be exhaustive. Some rules cannot be expressed with a simple algorithm, and other rules are only guidelines which you could circumvent when you need to.

Always remember this statement from PEP 8:

> *A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is most important.*

Among other things, these features are currently not in the scope of the `pep8` library:

- **naming conventions**: this kind of feature is supported through plugins. Install flake8 and the pep8-naming extension to use this feature.
- **docstring conventions**: they are not in the scope of this library; see the pep257 project.
- **automatic fixing**: see the section *PEP8 Fixers* in the *related tools* page.

1.3 Installation

You can install, upgrade, uninstall pep8.py with these commands:

```
$ pip install pep8
$ pip install --upgrade pep8
$ pip uninstall pep8
```

There's also a package for Debian/Ubuntu, but it's not always the latest version:

```
$ sudo apt-get install pep8
```

1.4 Example usage and output

```
$ pep8 --first optparse.py
optparse.py:69:11: E401 multiple imports on one line
optparse.py:77:1: E302 expected 2 blank lines, found 1
optparse.py:88:5: E301 expected 1 blank line, found 0
optparse.py:222:34: W602 deprecated form of raising exception
optparse.py:347:31: E211 whitespace before '('
optparse.py:357:17: E201 whitespace after '{'
optparse.py:472:29: E221 multiple spaces before operator
optparse.py:544:21: W601 .has_key() is deprecated, use 'in'
```

You can also make pep8.py show the source code for each error, and even the relevant text from PEP 8:

```
$ pep8 --show-source --show-pep8 testsuite/E40.py
testsuite/E40.py:2:10: E401 multiple imports on one line
import os, sys
         ^
    Imports should usually be on separate lines.

    Okay: import os\nimport sys
    E401: import sys, os
```

Or you can display how often each error was found:

```
$ pep8 --statistics -qq Python-2.5/Lib
232     E201 whitespace after '['
599     E202 whitespace before ')'
631     E203 whitespace before ','
842     E211 whitespace before '('
2531    E221 multiple spaces before operator
4473    E301 expected 1 blank line, found 0
4006    E302 expected 2 blank lines, found 1
165     E303 too many blank lines (4)
325     E401 multiple imports on one line
3615    E501 line too long (82 characters)
612     W601 .has_key() is deprecated, use 'in'
1188    W602 deprecated form of raising exception
```

Quick help is available on the command line:

```
$ pep8 -h
Usage: pep8 [options] input ...
```

```
Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v, --verbose         print status messages, or debug with -vv
  -q, --quiet           report only file names, or nothing with -qq
  --first               show first occurrence of each error
  --exclude=patterns    exclude files or directories which match these comma
                        separated patterns (default: .svn,CVS,.bzr,.hg,.git)
  --filename=patterns   when parsing directories, only check filenames matching
                        these comma separated patterns (default: *.py)
  --select=errors       select errors and warnings (e.g. E,W6)
  --ignore=errors       skip errors and warnings (e.g. E4,W)
  --show-source         show source code for each error
  --show-pep8           show text of PEP 8 for each error (implies --first)
  --statistics          count errors and warnings
  --count               print total number of errors and warnings to standard
                        error and set exit code to 1 if total is not null
  --max-line-length=n   set maximum allowed line length (default: 79)
  --hang-closing        hang closing bracket instead of matching indentation of
                        opening bracket's line
  --format=format       set the error format [default|pylint|<custom>]
  --diff                report only lines changed according to the unified diff
                        received on STDIN

  Testing Options:
    --benchmark         measure processing speed

  Configuration:
    The project options are read from the [pep8] section of the tox.ini
    file or the setup.cfg file located in any parent folder of the path(s)
    being processed.  Allowed options are: exclude, filename, select,
    ignore, max-line-length, hang-closing, count, format, quiet, show-pep8,
    show-source, statistics, verbose.

    --config=path       user config file location (default: ~/.config/pep8)
```

1.5 Configuration

The behaviour may be configured at two levels.

The user settings are read from the `~/.config/pep8` file. Example:

```
[pep8]
ignore = E226,E302,E41
max-line-length = 160
```

At the project level, a `tox.ini` file or a `setup.cfg` file is read if present (`.pep8` file is also supported, but it is deprecated). If none of these files have a `[pep8]` section, no project specific configuration is loaded.

If the `ignore` option is not in the configuration and not in the arguments, only the error codes `E226` and `E241/E242` are ignored (see below).

1.6 Error codes

This is the current list of error and warning codes:

code	sample message
E1	*Indentation*
E101	indentation contains mixed spaces and tabs
E111	indentation is not a multiple of four
E112	expected an indented block
E113	unexpected indentation
E121 (^)	continuation line indentation is not a multiple of four
E122 (^)	continuation line missing indentation or outdented
E123 (*)	closing bracket does not match indentation of opening bracket's line
E124 (^)	closing bracket does not match visual indentation
E125 (^)	continuation line does not distinguish itself from next logical line
E126 (^)	continuation line over-indented for hanging indent
E127 (^)	continuation line over-indented for visual indent
E128 (^)	continuation line under-indented for visual indent
E133 (*)	closing bracket is missing indentation
E2	*Whitespace*
E201	whitespace after '('
E202	whitespace before ')'
E203	whitespace before ':'
E211	whitespace before '('
E221	multiple spaces before operator
E222	multiple spaces after operator
E223	tab before operator
E224	tab after operator
E225	missing whitespace around operator
E226 (*)	missing whitespace around arithmetic operator
E227	missing whitespace around bitwise or shift operator
E228	missing whitespace around modulo operator
E231	missing whitespace after ','
E241 (*)	multiple spaces after ','
E242 (*)	tab after ','
E251	unexpected spaces around keyword / parameter equals
E261	at least two spaces before inline comment
E262	inline comment should start with '# '
E271	multiple spaces after keyword
E272	multiple spaces before keyword
E273	tab after keyword
E274	tab before keyword
E3	*Blank line*
E301	expected 1 blank line, found 0
E302	expected 2 blank lines, found 0

Continued on next page

Table 1.1 – continued from previous page

code	sample message
E303	too many blank lines (3)
E304	blank lines found after function decorator
E4	*Import*
E401	multiple imports on one line
E5	*Line length*
E501 (^)	line too long (82 > 79 characters)
E502	the backslash is redundant between brackets
E7	*Statement*
E701	multiple statements on one line (colon)
E702	multiple statements on one line (semicolon)
E703	statement ends with a semicolon
E711 (^)	comparison to None should be 'if cond is None:'
E712 (^)	comparison to True should be 'if cond is True:' or 'if cond:'
E721	do not compare types, use 'isinstance()'
E9	*Runtime*
E901	SyntaxError or IndentationError
E902	IOError
W1	*Indentation warning*
W191	indentation contains tabs
W2	*Whitespace warning*
W291	trailing whitespace
W292	no newline at end of file
W293	blank line contains whitespace
W3	*Blank line warning*
W391	blank line at end of file
W6	*Deprecation warning*
W601	.has_key() is deprecated, use 'in'
W602	deprecated form of raising exception
W603	'<>' is deprecated, use '!='
W604	backticks are deprecated, use 'repr()'

(*) In the default configuration, the checks **E123**, **E133**, **E226**, **E241** and **E242** are ignored because they are not rules unanimously accepted, and PEP 8 does not enforce them. The check **E133** is mutually exclusive with check **E123**. Use switch `--hang-closing` to report **E133** instead of **E123**.

(^) These checks can be disabled at the line level using the `# noqa` special comment. This possibility should be reserved for special cases.

> *Special cases aren't special enough to break the rules.*

Note: most errors can be listed with such one-liner:

```
$ python pep8.py --first --select E,W testsuite/ --format '%(code)s: %(text)s'
```

1.7 Related tools

The flake8 checker is a wrapper around `pep8` and similar tools. It supports plugins.

Other tools which use `pep8` are referenced in the Wiki: list of related tools.

CHAPTER

TWO

ADVANCED USAGE

2.1 Automated tests

You can also execute *pep8* tests from Python code. For example, this can be highly useful for automated testing of coding style conformance in your project:

```
import unittest
import pep8


class TestCodeFormat(unittest.TestCase):

    def test_pep8_conformance(self):
        """Test that we conform to PEP8."""
        pep8style = pep8.StyleGuide(quiet=True)
        result = pep8style.check_files(['file1.py', 'file2.py'])
        self.assertEqual(result.total_errors, 0,
                         "Found code style errors (and warnings).")
```

If you are using *nosetests* for running tests, remove *quiet=True* since Nose suppresses stdout.

There's also a shortcut for checking a single file:

```
import pep8

fchecker = pep8.Checker('testsuite/E27.py', show_source=True)
file_errors = fchecker.check_all()

print("Found %s errors (and warnings)" % file_errors)
```

2.2 Skip file header

Another example is related to the feature request #143: skip a number of lines at the beginning and the end of a file. This use case is easy to implement through a custom wrapper for the PEP 8 library:

```
#!python
import pep8

LINES_SLICE = slice(14, -20)

class PEP8(pep8.StyleGuide):
    """This subclass of pep8.StyleGuide will skip the first and last lines
```

```
    of each file."""

    def input_file(self, filename, lines=None, expected=None, line_offset=0):
        if lines is None:
            assert line_offset == 0
            line_offset = LINES_SLICE.start or 0
            lines = pep8.readlines(filename)[LINES_SLICE]
        return super(PEP8, self).input_file(
            filename, lines=lines, expected=expected, line_offset=line_offset)

if __name__ == '__main__':
    pep8style = PEP8(parse_argv=True, config_file=True)
    report = pep8style.check_files()
    if report.total_errors:
        raise SystemExit(1)
```

This module declares a lines' window which skips 14 lines at the beginning and 20 lines at the end. If there's no line to skip at the end, it could be changed with `LINES_SLICE = slice(14, None)` for example.

You can save it in a file and use it with the same options as the original `pep8`.

CHAPTER

THREE

PEP8 API

The library provides classes which are usable by third party tools.

3.1 Checker Classes

The `StyleGuide` class is used to configure a style guide checker instance to check multiple files.

The `Checker` class can be used to check a single file.

class pep8.**StyleGuide**(*parse_argv=False*, *config_file=None*, *parser=None*, *paths=None*, *report=None*, ***kwargs*)

Initialize a PEP-8 instance with few options.

init_report(*reporter=None*)

Initialize the report instance.

check_files(*paths=None*)

Run all checks on the paths.

input_file(*filename*, *lines=None*, *expected=None*, *line_offset=0*)

Run all checks on a Python source file.

input_dir(*dirname*)

Check all files in this directory and all subdirectories.

excluded(*filename*, *parent=None*)

Check if options.exclude contains a pattern that matches filename.

ignore_code(*code*)

Check if the error code should be ignored.

If 'options.select' contains a prefix of the error code, return False. Else, if 'options.ignore' contains a prefix of the error code, return True.

get_checks(*argument_name*)

Find all globally visible functions where the first argument name starts with argument_name and which contain selected tests.

class pep8.**Checker**(*filename=None*, *lines=None*, *report=None*, ***kwargs*)

Load a Python source file, tokenize it, check coding style.

readline()
Get the next line from the input buffer.

readline_check_physical()
Check and return the next physical line. This method can be used to feed tokenize.generate_tokens.

run_check(*check*, *argument_names*)
Run a check plugin.

check_physical(*line*)
Run all physical checks on a raw input line.

build_tokens_line()
Build a logical line from tokens.

check_logical()
Build a line from tokens and run all logical checks on it.

check_ast()

generate_tokens()

check_all(*expected=None*, *line_offset=0*)
Run all checks on the input file.

3.2 Report Classes

class pep8.**BaseReport**(*options*)
Collect the results of the checks.

start()
Start the timer.

stop()
Stop the timer.

init_file(*filename*, *lines*, *expected*, *line_offset*)
Signal a new file.

increment_logical_line()
Signal a new logical line.

error(*line_number*, *offset*, *text*, *check*)
Report an error, according to options.

get_file_results()
Return the count of errors and warnings for this file.

get_count(*prefix=''*)
Return the total count of errors and warnings.

get_statistics(*prefix=''*)
Get statistics for message codes that start with the prefix.

prefix=” matches all errors and warnings prefix=’E’ matches all errors prefix=’W’ matches all warnings prefix=’E4’ matches all errors that have to do with imports

print_statistics(*prefix=''*)
Print overall statistics (number of errors and warnings).

print_benchmark()
Print benchmark numbers.

class pep8.**FileReport**(*options*)
Collect the results of the checks and print only the filenames.

class pep8.**StandardReport**(*options*)
Collect and print the results of the checks.

class pep8.**DiffReport**(*options*)
Collect and print the results for the changed lines only.

3.3 Utilities

pep8.**expand_indent**(*line*)
Return the amount of indentation. Tabs are expanded to the next multiple of 8.

```
>>> expand_indent('    ')
4
>>> expand_indent('\t')
8
>>> expand_indent('    \t')
8
>>> expand_indent('       \t')
8
>>> expand_indent('        \t')
16
```

pep8.**mute_string**(*text*)
Replace contents with 'xxx' to prevent syntax matching.

```
>>> mute_string('"abc"')
'"xxx"'
>>> mute_string("'''abc'''")
"'''xxx'''"
>>> mute_string("r'abc'")
"r'xxx'"
```

pep8.**read_config**(*options*, *args*, *arglist*, *parser*)
Read both user configuration and local configuration.

pep8.**process_options**(*arglist=None*, *parse_argv=False*, *config_file=None*)
Process options passed either via arglist or via command line args.

pep8.**register_check**(*func_or_cls*, *codes=None*)
Register a new check object.

CHAPTER

FOUR

DEVELOPER'S NOTES

4.1 Source code

The source code is currently available on GitHub under the terms and conditions of the *Expat license*. Fork away!

- Source code and issue tracker on GitHub.
- Continuous tests against Python 2.5 through 3.3 and PyPy, on Travis-CI platform.

4.2 Contribute

You can add checks to this program by writing plugins. Each plugin is a simple function that is called for each line of source code, either physical or logical.

Physical line:

- Raw line of text from the input file.

Logical line:

- Multi-line statements converted to a single line.
- Stripped left and right.
- Contents of strings replaced with `"xxx"` of same length.
- Comments removed.

The check function requests physical or logical lines by the name of the first argument:

```
def maximum_line_length(physical_line)
def extraneous_whitespace(logical_line)
def blank_lines(logical_line, blank_lines, indent_level, line_number)
```

The last example above demonstrates how check plugins can request additional information with extra arguments. All attributes of the `Checker` object are available. Some examples:

- `lines`: a list of the raw lines from the input file
- `tokens`: the tokens that contribute to this logical line
- `line_number`: line number in the input file
- `blank_lines`: blank lines before this one
- `indent_char`: first indentation character in this file (`" "` or `"\t"`)

- `indent_level`: indentation (with tabs expanded to multiples of 8)
- `previous_indent_level`: indentation on previous line
- `previous_logical`: previous logical line

The docstring of each check function shall be the relevant part of text from PEP 8. It is printed if the user enables `--show-pep8`. Several docstrings contain examples directly from the PEP 8 document.

```
Okay: spam(ham[1], {eggs: 2})
E201: spam( ham[1], {eggs: 2})
```

These examples are verified automatically when pep8.py is run with the `--doctest` option. You can add examples for your own check functions. The format is simple: `"Okay"` or error/warning code followed by colon and space, the rest of the line is example source code. If you put `'r'` before the docstring, you can use `\n` for newline and `\t` for tab.

Then be sure to pass the tests:

```
$ python pep8.py --testsuite testsuite
$ python pep8.py --doctest
$ python pep8.py --verbose pep8.py
```

4.3 Changes

4.3.1 1.4.6 (2013-07-02)

- Honor `# noqa` for errors E711 and E712. (Issue #180)
- When both a `tox.ini` and a `setup.cfg` are present in the project directory, merge their contents. The `tox.ini` file takes precedence (same as before). (Issue #182)
- Give priority to `--select` over `--ignore`. (Issue #188)
- Compare full path when excluding a file. (Issue #186)
- Correctly report other E12 errors when E123 is ignored. (Issue #103)
- New option `--hang-closing` to switch to the alternative style of closing bracket indentation for hanging indent. Add error E133 for closing bracket which is missing indentation. (Issue #103)
- Accept both styles of closing bracket indentation for hanging indent. Do not report error E123 in the default configuration. (Issue #103)
- Do not crash when running AST checks and the document contains null bytes. (Issue #184)
- Fix false positive E261/E262 when the file contains a BOM. (Issue #193)
- Fix E701, E702 and E703 not detected sometimes. (Issue #196)
- Fix E122 not detected in some cases. (Issue #201 and #208)
- Fix false positive E121 with multiple brackets. (Issue #203)

4.3.2 1.4.5 (2013-03-06)

- When no path is specified, do not try to read from stdin. The feature was added in 1.4.3, but it is not supported on Windows. Use − filename argument to read from stdin. This usage is supported since 1.3.4. (Issue #170)
- Do not require `setuptools` in setup.py. It works around an issue with `pip` and Python 3. (Issue #172)

- Add `__pycache__` to the ignore list.
- Change misleading message for E251. (Issue #171)
- Do not report false E302 when the source file has a coding cookie or a comment on the first line. (Issue #174)
- Reorganize the tests and add tests for the API and for the command line usage and options. (Issues #161 and #162)
- Ignore all checks which are not explicitly selected when `select` is passed to the `StyleGuide` constructor.

4.3.3 1.4.4 (2013-02-24)

- Report E227 or E228 instead of E225 for whitespace around bitwise, shift or modulo operators. (Issue #166)
- Change the message for E226 to make clear that it is about arithmetic operators.
- Fix a false positive E128 for continuation line indentation with tabs.
- Fix regression with the `--diff` option. (Issue #169)
- Fix the `TestReport` class to print the unexpected warnings and errors.

4.3.4 1.4.3 (2013-02-22)

- Hide the `--doctest` and `--testsuite` options when installed.
- Fix crash with AST checkers when the syntax is invalid. (Issue #160)
- Read from standard input if no path is specified.
- Initiate a graceful shutdown on `Control+C`.
- Allow to change the `checker_class` for the `StyleGuide`.

4.3.5 1.4.2 (2013-02-10)

- Support AST checkers provided by third-party applications.
- Register new checkers with `register_check(func_or_cls, codes)`.
- Allow to construct a `StyleGuide` with a custom parser.
- Accept visual indentation without parenthesis after the `if` statement. (Issue #151)
- Fix UnboundLocalError when using `# noqa` with continued lines. (Issue #158)
- Re-order the lines for the `StandardReport`.
- Expand tabs when checking E12 continuation lines. (Issue #155)
- Refactor the testing class `TestReport` and the specific test functions into a separate test module.

4.3.6 1.4.1 (2013-01-18)

- Allow sphinx.ext.autodoc syntax for comments. (Issue #110)
- Report E703 instead of E702 for the trailing semicolon. (Issue #117)
- Honor `# noqa` in addition to `# nopep8`. (Issue #149)

- Expose the `OptionParser` factory for better extensibility.

4.3.7 1.4 (2012-12-22)

- Report E226 instead of E225 for optional whitespace around common operators (`*`, `**`, `/`, `+` and `-`). This new error code is ignored in the default configuration because PEP 8 recommends to "use your own judgement". (Issue #96)
- Lines with a `# nopep8` at the end will not issue errors on line length E501 or continuation line indentation E12*. (Issue #27)
- Fix AssertionError when the source file contains an invalid line ending `"\r\r\n"`. (Issue #119)
- Read the `[pep8]` section of `tox.ini` or `setup.cfg` if present. (Issue #93 and #141)
- Add the Sphinx-based documentation, and publish it on http://pep8.readthedocs.org/. (Issue #105)

4.3.8 1.3.4 (2012-12-18)

- Fix false positive E124 and E128 with comments. (Issue #100)
- Fix error on stdin when running with bpython. (Issue #101)
- Fix false positive E401. (Issue #104)
- Report E231 for nested dictionary in list. (Issue #142)
- Catch E271 at the beginning of the line. (Issue #133)
- Fix false positive E126 for multi-line comments. (Issue #138)
- Fix false positive E221 when operator is preceded by a comma. (Issue #135)
- Fix `--diff` failing on one-line hunk. (Issue #137)
- Fix the `--exclude` switch for directory paths. (Issue #111)
- Use `-` filename to read from standard input. (Issue #128)

4.3.9 1.3.3 (2012-06-27)

- Fix regression with continuation line checker. (Issue #98)

4.3.10 1.3.2 (2012-06-26)

- Revert to the previous behaviour for `--show-pep8`: do not imply `--first`. (Issue #89)
- Add E902 for IO errors. (Issue #87)
- Fix false positive for E121, and missed E124. (Issue #92)
- Set a sensible default path for config file on Windows. (Issue #95)
- Allow `verbose` in the configuration file. (Issue #91)
- Show the enforced `max-line-length` in the error message. (Issue #86)

4.3.11 1.3.1 (2012-06-18)

- Explain which configuration options are expected. Accept and recommend the options names with hyphen instead of underscore. (Issue #82)
- Do not read the user configuration when used as a module (except if `config_file=True` is passed to the `StyleGuide` constructor).
- Fix wrong or missing cases for the E12 series.
- Fix cases where E122 was missed. (Issue #81)

4.3.12 1.3 (2012-06-15)

Warning: The internal API is backwards incompatible.

- Remove global configuration and refactor the library around a `StyleGuide` class; add the ability to configure various reporters. (Issue #35 and #66)
- Read user configuration from `~/.config/pep8` and local configuration from `./.pep8`. (Issue #22)
- Fix E502 for backslash embedded in multi-line string. (Issue #68)
- Fix E225 for Python 3 iterable unpacking (PEP 3132). (Issue #72)
- Enable the new checkers from the E12 series in the default configuration.
- Suggest less error-prone alternatives for E712 errors.
- Rewrite checkers to run faster (E22, E251, E27).
- Fixed a crash when parsed code is invalid (too many closing brackets).
- Fix E127 and E128 for continuation line indentation. (Issue #74)
- New option `--format` to customize the error format. (Issue #23)
- New option `--diff` to check only modified code. The unified diff is read from STDIN. Example: `hg diff | pep8 --diff` (Issue #39)
- Correctly report the count of failures and set the exit code to 1 when the `--doctest` or the `--testsuite` fails.
- Correctly detect the encoding in Python 3. (Issue #69)
- Drop support for Python 2.3, 2.4 and 3.0. (Issue #78)

4.3.13 1.2 (2012-06-01)

- Add E121 through E128 for continuation line indentation. These checks are disabled by default. If you want to force all checks, use switch `--select=E,W`. Patch by Sam Vilain. (Issue #64)
- Add E721 for direct type comparisons. (Issue #47)
- Add E711 and E712 for comparisons to singletons. (Issue #46)
- Fix spurious E225 and E701 for function annotations. (Issue #29)
- Add E502 for explicit line join between brackets.
- Fix E901 when printing source with `--show-source`.
- Report all errors for each checker, instead of reporting only the first occurence for each line.

- Option `--show-pep8` implies `--first`.

4.3.14 1.1 (2012-05-24)

- Add E901 for syntax errors. (Issues #63 and #30)
- Add E271, E272, E273 and E274 for extraneous whitespace around keywords. (Issue #57)
- Add `tox.ini` configuration file for tests. (Issue #61)
- Add `.travis.yml` configuration file for continuous integration. (Issue #62)

4.3.15 1.0.1 (2012-04-06)

- Fix inconsistent version numbers.

4.3.16 1.0 (2012-04-04)

- Fix W602 `raise` to handle multi-char names. (Issue #53)

4.3.17 0.7.0 (2012-03-26)

- Now `--first` prints only the first occurence of each error. The `--repeat` flag becomes obsolete because it is the default behaviour. (Issue #6)
- Allow to specify `--max-line-length`. (Issue #36)
- Make the shebang more flexible. (Issue #26)
- Add testsuite to the bundle. (Issue #25)
- Fixes for Jython. (Issue #49)
- Add PyPI classifiers. (Issue #43)
- Fix the `--exclude` option. (Issue #48)
- Fix W602, accept `raise` with 3 arguments. (Issue #34)
- Correctly select all tests if `DEFAULT_IGNORE == ''`.

4.3.18 0.6.1 (2010-10-03)

- Fix inconsistent version numbers. (Issue #21)

4.3.19 0.6.0 (2010-09-19)

- Test suite reorganized and enhanced in order to check more failures with fewer test files. Read the `run_tests` docstring for details about the syntax.
- Fix E225: accept `print >>sys.stderr, "..."` syntax.
- Fix E501 for lines containing multibyte encoded characters. (Issue #7)
- Fix E221, E222, E223, E224 not detected in some cases. (Issue #16)

- Fix E211 to reject `v = dic['a'] ['b']`. (Issue #17)
- Exit code is always 1 if any error or warning is found. (Issue #10)
- `--ignore` checks are now really ignored, especially in conjunction with `--count`. (Issue #8)
- Blank lines with spaces yield W293 instead of W291: some developers want to ignore this warning and indent the blank lines to paste their code easily in the Python interpreter.
- Fix E301: do not require a blank line before an indented block. (Issue #14)
- Fix E203 to accept NumPy slice notation `a[0, :]`. (Issue #13)
- Performance improvements.
- Fix decoding and checking non-UTF8 files in Python 3.
- Fix E225: reject `True+False` when running on Python 3.
- Fix an exception when the line starts with an operator.
- Allow a new line before closing `)`, `}` or `]`. (Issue #5)

4.3.20 0.5.0 (2010-02-17)

- Changed the `--count` switch to print to sys.stderr and set exit code to 1 if any error or warning is found.
- E241 and E242 are removed from the standard checks. If you want to include these checks, use switch `--select=E,W`. (Issue #4)
- Blank line is not mandatory before the first class method or nested function definition, even if there's a docstring. (Issue #1)
- Add the switch `--version`.
- Fix decoding errors with Python 3. (Issue #13 [1])
- Add `--select` option which is mirror of `--ignore`.
- Add checks E261 and E262 for spaces before inline comments.
- New check W604 warns about deprecated usage of backticks.
- New check W603 warns about the deprecated operator <>.
- Performance improvement, due to rewriting of E225.
- E225 now accepts:
  - no whitespace after unary operator or similar. (Issue #9 [1])
  - lambda function with argument unpacking or keyword defaults.
- Reserve "2 blank lines" for module-level logical blocks. (E303)
- Allow multi-line comments. (E302, issue #10 [1])

4.3.21 0.4.2 (2009-10-22)

- Decorators on classes and class methods are OK now.

[1] These issues refer to the previous issue tracker.

4.3.22 0.4 (2009-10-20)

- Support for all versions of Python from 2.3 to 3.1.
- New and greatly expanded self tests.
- Added `--count` option to print the total number of errors and warnings.
- Further improvements to the handling of comments and blank lines. (Issue #1 [1] and others changes.)
- Check all py files in directory when passed a directory (Issue #2 [1]). This also prevents an exception when traversing directories with non `*.py` files.
- E231 should allow commas to be followed by `)`. (Issue #3 [1])
- Spaces are no longer required around the equals sign for keyword arguments or default parameter values.

4.3.23 0.3.1 (2009-09-14)

- Fixes for comments: do not count them when checking for blank lines between items.
- Added setup.py for pypi upload and easy_installability.

4.3.24 0.2 (2007-10-16)

- Loads of fixes and improvements.

4.3.25 0.1 (2006-10-01)

- First release.
- Online documentation: http://pep8.readthedocs.org/
- Source code and issue tracker: https://github.com/jcrocholl/pep8

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *search*

CHAPTER

SIX

CREDITS

Created by Johann C. Rocholl.

Maintained by Florent Xicluna.

CHAPTER

SEVEN

LICENSE

The `pep8` library is provided under the terms and conditions of the Expat license:

```
# Permission is hereby granted, free of charge, to any person
# obtaining a copy of this software and associated documentation files
# (the "Software"), to deal in the Software without restriction,
# including without limitation the rights to use, copy, modify, merge,
# publish, distribute, sublicense, and/or sell copies of the Software,
# and to permit persons to whom the Software is furnished to do so,
# subject to the following conditions:
#
# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
# BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
# ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
# CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.
```

PYTHON MODULE INDEX