Introduction

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

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 `pycodestyle.py` file for this purpose.
- Comes with a comprehensive test suite.

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 `pycodestyle` 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 pydocstyle project.

- **automatic fixing**: see the section *PEP8 Fixers* in the related tools page.

Installation

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

```
$ pip install pycodestyle
$ pip install --upgrade pycodestyle
$ pip uninstall pycodestyle
```

Example usage and output

```
$ pycodestyle --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 `pycodestyle.py` show the source code for each error, and even the relevant text from PEP 8:

```
$ pycodestyle --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:

```
$ pycodestyle --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
```

You can also make `pycodestyle.py` show the error text in different formats by using `--format` having options default/pylint/custom:

```
$ pycodestyle testsuite/E40.py --format=default
testsuite/E40.py:2:10: E401 multiple imports on one line

$ pycodestyle testsuite/E40.py --format=pylint
testsuite/E40.py:2: [E401] multiple imports on one line

$ pycodestyle testsuite/E40.py --format='%(path)s|%(row)d|%(col)d| %(code)s %(text)s'
testsuite/E40.py|2|10| E401 multiple imports on one line
```

Variables in the `custom` format option

Variable	Significance
`path`	File name
`row`	Row number
`col`	Column number
`code`	Error code
`text`	Error text

Quick help is available on the command line:

```
$ pycodestyle -h
Usage: pycodestyle [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)
  --max-doc-length=n   set maximum allowed doc line length and perform these
                       checks (unchecked if not set)
  --indent-size=n      set how many spaces make up an indent (default: 4)
  --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 [pycodestyle] 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, max-doc-length, hang-closing, count,
    format, quiet, show-pep8, show-source, statistics, verbose.

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

Configuration

The behaviour may be configured at two levels, the user and project levels.

At the user level, settings are read from the following locations:

If on Windows:

> `~\.pycodestyle`

Otherwise, if the `XDG_CONFIG_HOME` environment variable is defined:

> `XDG_CONFIG_HOME/pycodestyle`

Else if `XDG_CONFIG_HOME` is not defined:

`~/.config/pycodestyle`

Example:

```
[pycodestyle]
count = False
ignore = E226,E302,E41
max-line-length = 160
statistics = True
```

At the project level, a `setup.cfg` file or a `tox.ini` file is read if present. If none of these files have a `[pycodestyle]` section, no project specific configuration is loaded.

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
E114	indentation is not a multiple of four (comment)
E115	expected an indented block (comment)
E116	unexpected indentation (comment)
E117	over-indented
E121 (*^)	continuation line under-indented for hanging indent
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 with same indent as 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

code	sample message
E129 (^)	visually indented line with same indent as next logical line
E131 (^)	continuation line unaligned for hanging indent
E133 (*)	closing bracket is missing indentation
E2	*Whitespace*
E201	whitespace after '('
E202	whitespace before ')'
E203	whitespace before ',', ';', or ':'
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 ',', ';', or ':'
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 '# '

code	sample message
E265	block comment should start with ‘# ‘
E266	too many leading ‘#’ for block comment
E271	multiple spaces after keyword
E272	multiple spaces before keyword
E273	tab after keyword
E274	tab before keyword
E275	missing whitespace after keyword
E3	*Blank line*
E301	expected 1 blank line, found 0
E302	expected 2 blank lines, found 0
E303	too many blank lines (3)
E304	blank lines found after function decorator
E305	expected 2 blank lines after end of function or class
E306	expected 1 blank line before a nested definition
E4	*Import*
E401	multiple imports on one line
E402	module level import not at top of file
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

code	sample message
E704 (*)	multiple statements on one line (def)
E711 (^)	comparison to None should be ‘if cond is None:’
E712 (^)	comparison to True should be ‘if cond is True:’ or ‘if cond:’
E713	test for membership should be ‘not in’
E714	test for object identity should be ‘is not’
E721 (^)	do not compare types, use ‘isinstance()’
E722	do not use bare except, specify exception instead
E731	do not assign a lambda expression, use a def
E741	do not use variables named ‘l’, ‘O’, or ‘I’
E742	do not define classes named ‘l’, ‘O’, or ‘I’
E743	do not define functions named ‘l’, ‘O’, or ‘I’
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
W5	*Line break warning*
W503 (*)	line break before binary operator

code	sample message
W504 (*)	line break after binary operator
W505 (*^)	doc line too long (82 > 79 characters)
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()'
W605	invalid escape sequence 'x'
W606	'async' and 'await' are reserved keywords starting with Python 3.7

(*) In the default configuration, the checks **E121**, **E123**, **E126**, **E133**, **E226**, **E241**, **E242**, **E704**, **W503**, **W504** and **W505** are ignored because they are not rules unanimously accepted, and PEP 8 does not enforce them. Please note that if the option `--ignore=errors` is used, the default configuration will be overridden and ignore only the check(s) you skip. The check **W503** is mutually exclusive with check **W504**. The check **E133** is mutually exclusive with check **E123**. Use switch `--hang-closing` to report **E133** instead of **E123**. Use switch `--max-doc-length=n` to report **W505**.

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

Related tools

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

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