Linters for Python Code Quality

Run the code against a variety of linters to test the code quality:

- **pylint code_with_lint.py**
- **pyflakes code_with_lint.py**
- **pycodestyle code_with_lint.py**
- **pydocstyle code_with_lint.py**

Compare the effectiveness of each tool in defining and identifying code quality. What can you conclude about the effectiveness of each approach?

Linter output

pylint

********** Module code_with_lint

code_with_lint.py:27:0: W0301: Unnecessary semicolon (unnecessary-semicolon)

code_with_lint.py:31:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)

code_with_lint.py:33:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)

code_with_lint.py:46:0: C0305: Trailing newlines (trailing-newlines)

code_with_lint.py:1:0: C0114: Missing module docstring (missing-module-docstring)

code_with_lint.py:2:0: W0622: Redefining built-in 'pow' (redefined-builtin)

code_with_lint.py:2:0: W0401: Wildcard import math (wildcard-import)

code_with_lint.py:7:0: C0103: Constant name "some_global_var" doesn't conform to UPPER_CASE naming style (invalid-name)

code_with_lint.py:13:4: W0621: Redefining name 'some_global_var' from outer scope (line 7) (redefined-outer-name)

code_with_lint.py:16:4: W0101: Unreachable code (unreachable)

code_with_lint.py:13:4: W0612: Unused variable 'some_global_var' (unused-variable)

code_with_lint.py:25:7: C0121: Comparison 'x != None' should be 'x is not None' (singleton-comparison)

code_with_lint.py:28:12: R1705: Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (no-else-return)

code_with_lint.py:19:0: R1710: Either all return statements in a function should return an expression, or none of them should. (inconsistent-return-statements)

code_with_lint.py:35:0: C0115: Missing class docstring (missing-class-docstring)

code_with_lint.py:41:8: W0621: Redefining name 'time' from outer scope (line 5) (redefined-outer-name)

code_with_lint.py:41:15: E0601: Using variable 'time' before assignment (used-before-assignment)

code_with_lint.py:42:8: C0415: Import outside toplevel (datetime.datetime) (import-outside-toplevel)

code_with_lint.py:37:4: R1711: Useless return at end of function or method (useless-return)

code_with_lint.py:37:50: W0613: Unused argument 'verbose' (unused-argument)

code_with_lint.py:40:8: W0612: Unused variable 'list_comprehension' (unused-variable)

code_with_lint.py:43:8: W0612: Unused variable 'date_and_time' (unused-variable)

code_with_lint.py:35:0: R0903: Too few public methods (0/2) (too-few-public-methods)

code_with_lint.py:1:0: W0611: Unused import io (unused-import)

code_with_lint.py:5:0: W0611: Unused time imported from time (unused-import)

code_with_lint.py:2:0: W0614: Unused import(s) acos, acosh, asin, asinh, atan, atan2, atanh, cbrt, ceil, comb, copysign, cos, cosh, degrees, dist, e, erf, erfc, exp, exp2, expm1, fabs, factorial, floor, fmod, frexp, fsum, gamma, gcd, hypot, inf, isclose, isfinite, isinf, isnan, isqrt, lcm, ldexp, lgamma, log, log10, log1p, log2, modf, nan, nextafter, perm, pow, prod, radians, remainder, sin, sinh, sqrt, tan, tanh, tau, trunc and ulp from wildcard import of math (unused-wildcard-import)

Your code has been rated at 0.00/10

pyflakes

code_with_lint.py:1:1: 'io' imported but unused

code_with_lint.py:2:1: 'from math import *' used; unable to detect undefined names

code_with_lint.py:13:5: local variable 'some_global_var' is assigned to but never used

code_with_lint.py:40:44: 'pi' may be undefined, or defined from star imports: math

code_with_lint.py:40:9: local variable 'list_comprehension' is assigned to but never used

code_with_lint.py:41:16: local variable 'time' defined in enclosing scope on line 5 referenced before assignment

code_with_lint.py:41:9: local variable 'time' is assigned to but never used

code_with_lint.py:43:9: local variable 'date_and_time' is assigned to but never used

pycodestyle

code_with_lint.py:9:1: E302 expected 2 blank lines, found 1

code_with_lint.py:14:15: E225 missing whitespace around operator

code_with_lint.py:19:1: E302 expected 2 blank lines, found 1

code_with_lint.py:20:80: E501 line too long (80 > 79 characters)

code_with_lint.py:25:10: E711 comparison to None should be 'if cond is not None:'

code_with_lint.py:27:25: E703 statement ends with a semicolon

code_with_lint.py:31:23: E275 missing whitespace after keyword

code_with_lint.py:31:24: E201 whitespace after '('

code_with_lint.py:35:1: E302 expected 2 blank lines, found 1

code_with_lint.py:37:58: E251 unexpected spaces around keyword / parameter equals

code_with_lint.py:37:60: E251 unexpected spaces around keyword / parameter equals

code_with_lint.py:38:28: E221 multiple spaces before operator

code_with_lint.py:38:31: E222 multiple spaces after operator

code_with_lint.py:39:22: E221 multiple spaces before operator

code_with_lint.py:39:31: E222 multiple spaces after operator

code_with_lint.py:40:80: E501 line too long (83 > 79 characters)

code_with_lint.py:46:1: W391 blank line at end of file

pydocstyle

code_with_lint.py:1 at module level:

    D100: Missing docstring in public module

code_with_lint.py:10 in public function `multiply`:

    D200: One-line docstring should fit on one line with quotes (found 3)

code_with_lint.py:10 in public function `multiply`:

    D400: First line should end with a period (not 's')

code_with_lint.py:10 in public function `multiply`:

    D401: First line should be in imperative mood; try rephrasing (found 'This')

code_with_lint.py:20 in public function `is_sum_lucky`:

    D205: 1 blank line required between summary line and description (found 0)

code_with_lint.py:20 in public function `is_sum_lucky`:

    D400: First line should end with a period (not 'y')

code_with_lint.py:20 in public function `is_sum_lucky`:

D401: First line should be in imperative mood; try rephrasing (found 'This')

code_with_lint.py:35 in public class `SomeClass`:

D101: Missing docstring in public class

code_with_lint.py:37 in public method `__init__`:

D107: Missing docstring in __init__

Comparisons

I used Chat GPT (OpenAI, 2025) to find the below bulleted crossover statements. I analysed these to see which linter was the best all-rounder linter.

Unused imports

2 linters picked up on these, with pylint providing a more comprehensive breakdown of what was unused from the math import (where all or '*' was imported). Pyflakes merely hinted that 'math import *' was unadvisable.

- **pylint**:
  - W0611: Unused import io (unused-import)
  - W0611: Unused time imported from time (unused-import)
  - W0614: Unused import(s) acos, acosh, asin, asinh, atan, atan2, atanh, cbrt, ceil, comb, copysign, cos, cosh, degrees, dist, e, erf, erfc, exp, exp2, expm1, fabs, factorial, floor, fmod, frexp, fsum, gamma, gcd, hypot, inf, isclose, isfinite, isinf, isnan, isqrt, lcm, ldexp, lgamma, log, log10, log1p, log2, modf, nan, nextafter, perm, pow, prod, radians, remainder, sin, sinh, sqrt, tan, tanh, tau, trunc and ulp from wildcard import of math (unused-wildcard-import)
- **pyflakes**:
  - 'io' imported but unused
  - 'from math import *' used; unable to detect undefined names

Unused variables

Again, pylint and pyflakes were the only linters to pick up on these, with 3 similar findings but with pyflakes finding one more unused variable 'time'.

- **pylint:**
  - W0612: Unused variable 'some_global_var' (unused-variable)
  - W0612: Unused variable 'list_comprehension' (unused-variable)
  - W0612: Unused variable 'date_and_time' (unused-variable)
- **pyflakes:**
  - local variable 'some_global_var' is assigned to but never used
  - local variable 'list_comprehension' is assigned to but never used
  - local variable 'time' is assigned to but never used

- local variable 'date_and_time' is assigned to but never used

Comparison to None

- **pylint:** C0121: Comparison 'x != None' should be 'x is not None'
- **pycodestyle:** E711 comparison to None should be 'if cond is not None:'

pylint and pycodestyle picked up on the same comparison to None, however pylint quoted the specific variable when suggesting a change - 'x is not None' – whereas pycodestyle was more generic in its suggestion: ‘should be 'if cond is not None:'’

Trailing Semicolons

- **pylint**: W0301: Unnecessary semicolon (unnecessary-semicolon)
- **pycodestyle**: E703 statement ends with a semicolon

Again, pylint and pycodestyle found the same issue, this time with both providing similar responses.

Missing Docstrings

- **pylint:**
  - C0114: Missing module docstring (missing-module-docstring)
  - C0115: Missing class docstring (missing-class-docstring)
- **pydocstyle:**
  - D100: Missing docstring in public module
  - D101: Missing docstring in public class
  - D107: Missing docstring in __init__

Once again pylint and pycodestyle found similar issues. Pydocstyle gave more detail regarding the exact location of the docstring.

Trailing Newlines / Blank Line at End of File:

- **pylint:** C0305: Trailing newlines (trailing-newlines)
- **pycodestyle:** W391 blank line at end of file

Pylint and pycodestyle found the same trailing newline/blank line.

Variable Redefinition / Used Before Assignment:

- **pylint:**
  - W0622: Redefining built-in 'pow' (redefined-builtin)
  - W0621: Redefining name 'some_global_var' from outer scope (line 7) (redefined-outer-name)

    - W0621: Redefining name 'time' from outer scope (line 5) (redefined-outer-name)
    - E0601: Using variable 'time' before assignment (used-before-assignment)
- **pyflakes:**
    - local variable 'time' defined in enclosing scope on line 5 referenced before assignment

Pylint found more variable redefinition issues than pyflakes.

Conclusion

Pylint appears to be a more comprehensive linter, picking up on more coding quality issues than any of the other linters. It seems to be a good all-rounder, as it was ever-present where there was crossover with other linters, picking up on **Unused imports, Comparison to None, Trailing Semicolons, Missing Docstrings, Trailing Newlines** and **Variable Redefinition before assignment.**

Pyflakes appears to be a less comprehensive all-rounder, while pydocstyle – as the name suggests - appears to pick up on purely doc-string related quality issues. Pycodestyle seems to exclusively look out for irregularities with spaces, blank lines and line length.

Having used pylint before and after seeing these results, I will continue to use pylint for a comprehensive quality assessment, while employing pydocstyle to comprehensively analyse docstrings, and pycodestyle to comprehensively find issues relating to unneeded spaces.

References

OpenAI (2025) ChatGPT response to Todd Edge. Available at https://g.co/gemini/share/04081afca433 (Accessed: 3 July 2025).