Activity: Using Linters to Achieve Python Code Quality

Linters for Python Code Quality. Remember to save your work to your GitHub repository.

Copy the following Python code into a file named code_with_lint.py:

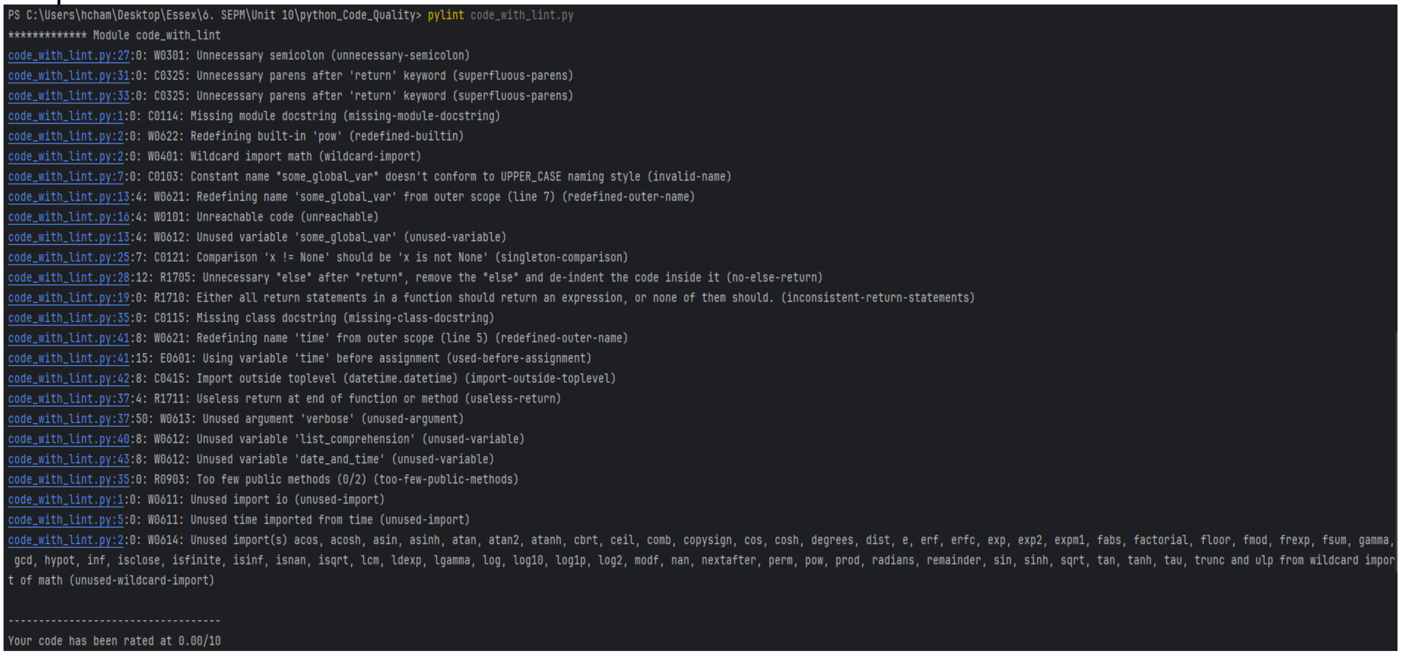
```
import io
from math import *


from time import time

some_global_var = 'GLOBAL VAR NAMES SHOULD BE IN ALL_CAPS_WITH_UNDERSCOES'

def multiply(x, y):
    """
    This returns the result of a multiplation of the inputs
    """
    some_global_var = 'this is actually a local variable...'
    result = x* y
    return result
    if result == 777:
        print("jackpot!")

def is_sum_lucky(x, y):
    """This returns a string describing whether or not the sum of input is lucky
    This function first makes sure the inputs are valid and then calculates the
    sum. Then, it will determine a message to return based on whether or not
    that sum should be considered "lucky"
    """
    if x != None:
        if y is not None:
            result = x+y;
            if result == 7:
                return 'a lucky number!'
            else:
                return( 'an unlucky number!')

            return ('just a normal number')

class SomeClass:

    def __init__(self, some_arg,  some_other_arg, verbose = False):
        self.some_other_arg  =  some_other_arg
        self.some_arg        =  some_arg
        list_comprehension = [((100/value)*pi) for value in some_arg if value != 0]
        time = time()
        from datetime import datetime
        date_and_time = datetime.now()
        return
```

Code source: https://realpython.com/python-code-quality/

- **Now run the code against a variety of linters to test the code quality:**

1. pylint code_with_lint.py

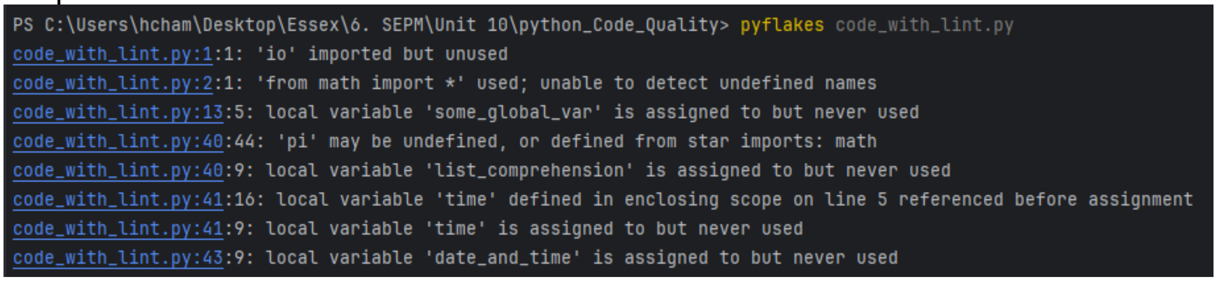
Output:



```
PS C:\Users\hcham\Desktop\Essex\6. SEPM\Unit 10\python_Code_Quality> pylint code_with_lint.py
************* 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: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 impor
t of math (unused-wildcard-import)

-----------------------------------
Your code has been rated at 0.00/10
```

2. pyflakes code_with_lint.py

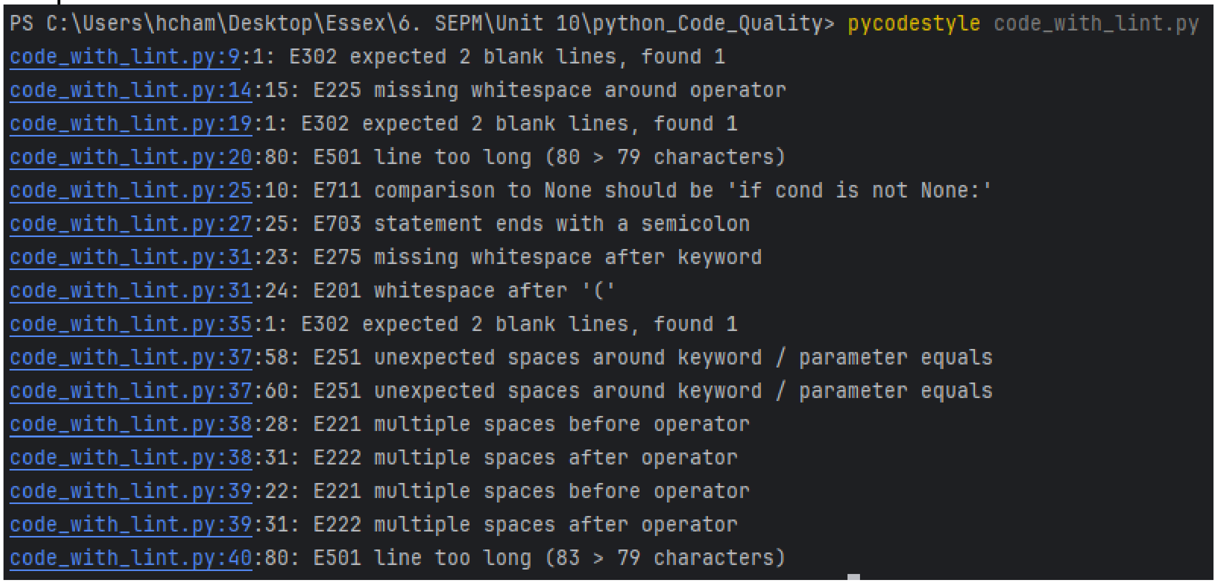
Output:



```
PS C:\Users\hcham\Desktop\Essex\6. SEPM\Unit 10\python_Code_Quality> pyflakes code_with_lint.py
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
```

3. pycodestyle code_with_lint.py

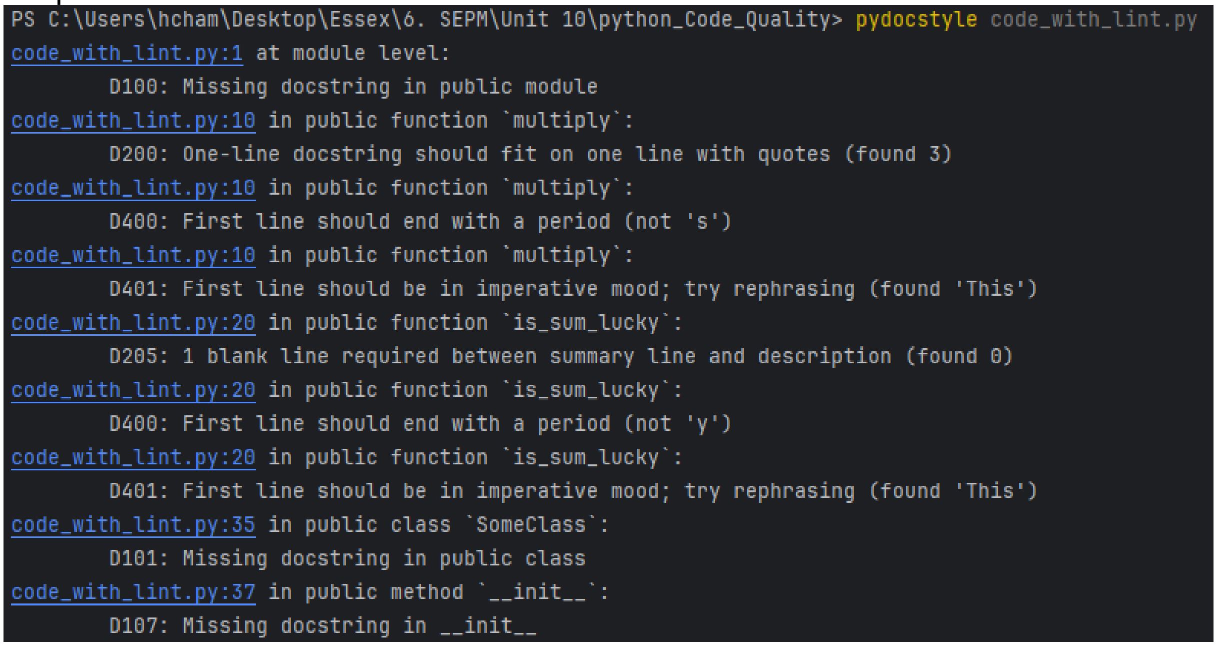
Output:



```
PS C:\Users\hcham\Desktop\Essex\6. SEPM\Unit 10\python_Code_Quality> pycodestyle code_with_lint.py
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)
```

4. pydocstyle code_with_lint.py

Output:



```
PS C:\Users\hcham\Desktop\Essex\6. SEPM\Unit 10\python_Code_Quality> pydocstyle code_with_lint.py
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__
```

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

Coding best practices improve programming skills, collaboration, code structure, efficiency, and error detection. They foster consistency, readability, maintainability, and code reuse while reducing complexity and development time (GeeksforGeeks, 2024). According to Simmons et al. (2020), coding standards, like inline documentation and structural construct organisation, improve readability, identify incorrect code, and follow best practices. Python coding standards, like PEP 8, cover code layout, naming, programming recommendations, package organisation, and virtual environments (Van Rossum et al. (2001). For instance, each linter focuses on different aspects of code quality:

- **Pylint** is a static code analysis tool under the Logical & Stylistic checks category (VanTol, 2018). It checks for errors, enforces coding standards, and detects lousy code smells. It offers features like checking line-code length and ensuring variable names are well-formed according to coding standards (Andreev, 2024).

- **PyFlakes** is a static analysis tool that examines programs and identifies different types of errors (VanTol, 2018). It is a Python source file error checker, similar to PyLint, but focuses on style, making it faster and less intrusive than other tools (Andreev, 2024).
- **Pycodestyle** falls under the Stylistic category and checks against some style conventions in PEP 8 (VanTol, 2018). PEP8 (pycodestyle) is a tool that verifies Python code against certain style conventions in Python's PEP 8 style guidelines, including indentation, spacing, naming conventions, and more.
- **Pydocstyle** falls under the category of Stylistic and checks compliance with Python docstring conventions (VanTol, 2018). It is a Python module that generates documentation for modules, scripts, and classes, allowing users to access built-in strings and view them in a user-friendly format (Andreev, 2024).

Code quality is essential in software development, and understanding developers' perceptions could enhance the effectiveness of software quality efforts (Börstler et al., 2023). Pylint is a comprehensive linter that can detect numerous issues, making it an excellent option for thorough code reviews. Pyflakes identifies unused imports and undefined variables, while pycodestyle is essential for maintaining consistent and readable code styles. Additionally, Pydocstyle ensures proper documentation, which contributes to improved code clarity and maintainability. The effectiveness of each tool in defining and identifying code quality depends on specific needs, and combining multiple tools is often recommended for a comprehensive assessment of code quality, as VanTol (2018) explains.

References:

Andreev, M. (2024). *Enhance Your Project Quality with These Top Python Libraries*. Available from: https://dev.to/mrkandreev/enhance-your-project-quality-with-these-top-python-libraries-2ln5 [Accessed 30 May 2024].

Börstler, J., Bennin, K. E., Hooshangi, S., Jeuring, J., Keuning, H., Kleiner, C., MacKellar, B., Duran, R., Störrle, H., Toll, D. & Van Assema, J. (2023). Developers talking about code quality. *Empirical Software Engineering*, 28(6). Available from: https://doi.org/10.1007/s10664-023-10381-0.

GeeksforGeeks (2024) *Coding Standards and Guidelines*. Available from: https://www.geeksforgeeks.org/coding-standards-and-guidelines/.

Van Rossum, G., Warsaw, B. & Coghlan, N. (2001) *PEP 8 – Style Guide for Python Code*. Available from: https://peps.python.org/pep-0008/ [Accessed 30 May 2024].

Simmons, A.J., Barnett, S., Rivera-Villicana, J., Bajaj, A. & Vasa, R. (2020). A large-scale comparative analysis of Coding Standard conformance in Open-Source Data Science projects. *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Available from: https://doi.org/10.1145/3382494.3410680.

VanTol, A. (2018) *Python Code Quality: Tools & Best Practices*. [Online] [online]. Available from: https://realpython.com/python-code-quality/ [Accessed 30 May 2024].