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
```

Now 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?

Copying the code into Visual Studio Code on my local machine:

```
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
```

The output from 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:44:0: C0304: Final newline missing (missing-final-newline)

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:9:13: C0103: Argument name "x" doesn't conform to snake_case naming style (invalid-name)

code_with_lint.py:9:16: C0103: Argument name "y" doesn't conform to snake_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: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

The output from 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

The output from 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:44:15: W292 no newline at end of file

The output from 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__
```

Conclusion

Both pylint and pyflakes focus on syntactic and stylistic errors (Sweigart, 2022), whereas pycodestyle and pydocstyle focus purely on stylistic features. Pylint is definitely the most comprehensive linter in terms of code quality aspects that effect the immediate requirements of a program. Pycodestyle and pydocstyle would be more beneficial in terms of maintaining the code.

References:

Sweigart, A. (2022) Python Linter Comparison 2022: Pylint vs Pyflakes vs Flake8 vs autopep8 vs Bandit vs Prospector vs Pylama vs Pyroma vs Black vs Mypy vs Radon vs mccabe. Available from https://inventwithpython.com/blog/2022/11/19/python-linter-comparison-2022-pylint-vs-pyflakes-vs-flake8-vs-autopep8-vs-bandit-vs-prospector-vs-pylama-vs-pyroma-vs-black-vs-mypy-vs-radon-vs-mccabe/ [Accessed 4 July 2023].