Codio Activity: Using Linters to Achieve Python Code Quality

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:

- 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?

pylint code_with_lint.py

```
codio@ritualscarlet-drinkgrand:~/workspace$ pylint code_with_lint.py
No config file found, using default configuration
************* Module code_with_lint
C: 17, 0: Unnecessary parens after 'print' keyword (superfluous-parens)
W: 27, 0: Unnecessary semicolon (unnecessary-semicolon)
C: 31, 0: Unnecessary parens after 'return' keyword (superfluous-parens)
C: 31, 0: No space allowed after bracket
               return( 'an unlucky number!')
                     ^ (bad-whitespace)
C: 33, 0: Unnecessary parens after 'return' keyword (superfluous-parens)
C: 37, 0: Exactly one space required after comma
    def __init__(self, some_arg,  some_other_arg, verbose = False):
                               ^ (bad-whitespace)
C: 37, 0: No space allowed around keyword argument assignment
    def __init__(self, some_arg,  some_other_arg, verbose = False):
                                                          ^ (bad-whitespace)
C: 38, 0: Exactly one space required around assignment
        self.some_other_arg  =  some_other_arg
                             ^ (bad-whitespace)
C: 39, 0: Exactly one space required around assignment
        self.some_arg        =  some_arg
                             ^ (bad-whitespace)
C: 44, 0: Final newline missing (missing-final-newline)
W:  2, 0: Redefining built-in 'pow' (redefined-builtin)
C:  1, 0: Missing module docstring (missing-docstring)
W:  2, 0: Wildcard import math (wildcard-import)
C:  7, 0: Constant name "some_global_var" doesn't conform to UPPER_CASE naming style (invalid-
W: 13, 4: Redefining name 'some_global_var' from outer scope (line 7) (redefined-outer-name)
C:  9, 0: Argument name "x" doesn't conform to snake_case naming style (invalid-name)
C:  9, 0: Argument name "y" doesn't conform to snake_case naming style (invalid-name)
W: 16, 4: Unreachable code (unreachable)
W: 13, 4: Unused variable 'some_global_var' (unused-variable)
C: 19, 0: Argument name "x" doesn't conform to snake_case naming style (invalid-name)
C: 19, 0: Argument name "y" doesn't conform to snake_case naming style (invalid-name)
R: 28,12: Unnecessary "else" after "return" (no-else-return)
R: 19, 0: Either all return statements in a function should return an expression, or none of t
. (inconsistent-return-statements)
C: 35, 0: Missing class docstring (missing-docstring)
C: 35, 0: Old-style class defined. (old-style-class)
W: 41, 8: Redefining name 'time' from outer scope (line 5) (redefined-outer-name)
E: 41,15: Using variable 'time' before assignment (used-before-assignment)
```

.

```
W:  5, 0: Unused time imported from time (unused-import)


-----------------------------------
Your code has been rated at -18.89/10
```

Pylint detects many syntactic and stylistic errors in the code, including unused arguments, extra semicolons, clauses, and parenthesis, as well as wildcard imports, missing docstrings, naming convention violations, redundant returns, and more. Overall, it generates a highly detailed report of the problems with the code's quality..

pyflakes code_with_lint.py

```
codio@ritualscarlet-drinkgrand:~/workspace$ 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 ass
ignment
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
codio@ritualscarlet-drinkgrand:~/workspace$
```

pyflakes repors some of the issues reported by pylint and seems to be providing only a lighter overview of the issues in the code in question.

pycodestyle code_with_lint.py

```
codio@ritualscarlet-drinkgrand:~/workspace$ 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:31:24: 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
codio@ritualscarlet-drinkgrand:~/workspace$
```

Pycodestyle checks the code against the style conventions defined in the official PEP8 style guide. It does not check for coding issues or errors. This tool is useful to make sure the styling conventions are followed as expected.

pydocstyle code_with_lint.py

```
codio@ritualscarlet-drinkgrand:~/workspace$ 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`:
        D401: First line should be in imperative mood; try rephrasing (found 'This')
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`:
        D200: One-line docstring should fit on one line with quotes (found 3)
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: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`:
        D205: 1 blank line required between summary line and description (found 0)
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__
codio@ritualscarlet-drinkgrand:~/workspace$
```

PyDocStyle focuses on checking compliance with the Python docstring conventions ensuring we are following Python docstring conventions.

Overall.

In comparison to the other tolls described above, pylint appears to provide greater reporting and functionality, however each has its own special advantages. If I had to pick just one linter, pylint would be my first pick