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

**Answer:**

Based on the different outputs, we can say that using the PyLint gives the most detailed description. It also includes the best practices of naming conventions for variables/functions, comparing to other tools. On the other hand, it appears that Pyflakes, PyCodestyle and PyDocStyle have more concise outputs. From that perspective, I would prefer to use these instead of PyLint however, the PyLint seems to be the most popular nowadays. Beside that it also assesses the code quality and displays the rating. Another interesting feature is that different linter tools have their own internal error code naming conventions which is somehow expected.

**Appendices:**

Using PyLint

codio@adamexotic-neonsalute:~/workspace$ 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:2:0: W0622: Redefining built-in 'pow' (redefined-builtin)

code\_with\_lint.py:1:0: C0114: Missing module docstring (missing-module-docstring)

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: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:16:4: W0101: Unreachable code (unreachable)

code\_with\_lint.py:13:4: W0612: Unused variable 'some\_global\_var' (unused-variable)

code\_with\_lint.py:19:17: C0103: Argument name "x" doesn't conform to snake\_case naming style (invalid-name)

code\_with\_lint.py:19:20: C0103: Argument name "y" doesn't conform to snake\_case naming style (invalid-name)

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, ceil, copysign, cos, cosh, degrees, e, erf, erfc, exp, expm1, fabs, factorial, floor, fmod, frexp, fsum, gamma, gcd, hypot, inf, isclose, isfinite, isinf, isnan, ldexp, lgamma, log, log10, log1p, log2, modf, nan, pow, radians, sin, sinh, sqrt, tan, tanh, tau and trunc from wildcard import of math (unused-wildcard-import)

-----------------------------------

Your code has been rated at 0.00/10

Using PyFlakes

codio@adamexotic-neonsalute:~/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 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

Using PyCodestyle

codio@adamexotic-neonsalute:~/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:31:23: 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)

Using PyDocStyle

codio@adamexotic-neonsalute:~/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`:

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\_\_

**Code:**

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