Codio Activity

Using Linters to Achieve Python Code Quality

```
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**

```
************* 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: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
```

- **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
```

- **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
```

- **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__
```

Comparison of linters

Linter	Analysis
pylint	• The most comprehensive feedback of all linters addresses docstrings, imports, and code styles. • Valuable feedback such as "`unreachable code`", and "`unused imports`" make developers aware of not writing code that is not used or importing definitions that are not required. ***Overall:*** recommended for all stages of Python development, especially regular commits into repositories. This linter covers a wide range of issues that help improve the quality and reliability of code when addressed early in development.
pyflakes	• Outputs messages related to unreferenced variables, bad imports or poor assignments. • The feedback is valuable to ensure developers only commit *working, necessary* code with no unused code. ***Overall:*** useful during Python code development and at the CI/CD level.
pycodestyle	• Focuses mainly on the *tokenization/parsing* aspects of Python code, for instance, "`statement ends with a semicolon`" or references to whitespace.

Linter	Analysis
	• Useful is notifying the user about a comparison to None and how it *should be* written. So, instead of "`if x != None`", the linter shows the correct form of "`if x is not None`" ***Overall:*** useful linter to be run as part of CI/CD scenarios to ensure high-quality code before final commit into repositories.
Pydocstyle	• Concerned with document comments and correct format of such statements. • Though, the notifying users that a sentence should end with "." It is not tremendously valuable. • I think the reference to "imperative mood" is helpful to cause developers to rethink the *phrasing* of the document comment. • Notifying the lack of public documentation comments is very useful to ensure all public methods are described clearly. ***Overall:*** *useful as a last stage linter before publishing code to public repositories.*