Codio Activity

Using Linters to Achieve Python Code Quality

Running Linters

```
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
```

Using pylint

Unfortunately, the Codio environment presented the following error attempting to install pylint:

```
    File "/tmp/easy_install-ZFsU97/setuptools_scm-6.4.2/setup.py", line 60, in <module>
      with io.open(
    File "/tmp/easy_install-ZFsU97/setuptools_scm-6.4.2/setup.py", line 31, in scm_version
      CFLAGS = ''
  RuntimeError: support for python < 3.6 has been removed in setuptools_scm>=6.0.0
```

Using 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:12:5 local variable 'some_global_var' is assigned to but never used
code_with_lint.py:39:44 'pi' may be undefined, or defined from star imports: math
code_with_lint.py:39:9 local variable 'list_comprehension' is assigned to but never used
code_with_lint.py:40:16 local variable 'time' defined in enclosing scope on line 4 referenced before assignment
code_with_lint.py:40:9 local variable 'time' is assigned to but never used
code_with_lint.py:42:9 local variable 'date_and_time' is assigned to but never used
codio@abrahamrachel-alamoengine:~/workspace$
```

The most useful output from this linter is reference to undefined variables (such as “pi”, line 39), unused variables (line 39, 12, 40 and 42). The error referring to “time defined in enclosing scope…before assignment” is the one that caused me to research and discover the cause of the message.

Using pycodestyle

```
code_with_lint.py:8:1: E302 expected 2 blank lines, found 1
code_with_lint.py:13:15: E225 missing whitespace around operator
code_with_lint.py:18:1: E302 expected 2 blank lines, found 1
code_with_lint.py:19:80: E501 line too long (80 > 79 characters)
code_with_lint.py:24:10: E711 comparison to None should be 'if cond is not None:'
code_with_lint.py:26:25: E703 statement ends with a semicolon
code_with_lint.py:30:24: E201 whitespace after '('
code_with_lint.py:34:1: E302 expected 2 blank lines, found 1
code_with_lint.py:36:58: E251 unexpected spaces around keyword / parameter equals
code_with_lint.py:36:60: E251 unexpected spaces around keyword / parameter equals
code_with_lint.py:37:28: E221 multiple spaces before operator
code_with_lint.py:37:31: E222 multiple spaces after operator
code_with_lint.py:38:22: E221 multiple spaces before operator
code_with_lint.py:38:31: E222 multiple spaces after operator
code_with_lint.py:39:80: E501 line too long (83 > 79 characters)
code_with_lint.py:43:15: W292 no newline at end of file
codio@abrahamrachel-alamoengine:~/workspace$
```

The results from this linter reference a lot of whitespace issues, which in my opinion, do not affect the *functionality* of the program and I am unlikely to spend time worrying about output like “whitespace after ‘(‘”. However, useful outputs are “comparison to None should be …” and “statement ends with a semicolon” because they are valuable actions I can take to improve code quality.

Using pydocstyle

```
code_with_lint.py:1 at module level:
        D100: Missing docstring in public module
code_with_lint.py:9 in public function `multiply`:
        D401: First line should be in imperative mood; try rephrasing (found 'This')
code_with_lint.py:9 in public function `multiply`:
        D400: First line should end with a period (not 's')
code_with_lint.py:9 in public function `multiply`:
        D200: One-line docstring should fit on one line with quotes (found 3)
code_with_lint.py:19 in public function `is_sum_lucky`:
        D401: First line should be in imperative mood; try rephrasing (found 'This')
code_with_lint.py:19 in public function `is_sum_lucky`:
        D400: First line should end with a period (not 'y')
code_with_lint.py:19 in public function `is_sum_lucky`:
        D205: 1 blank line required between summary line and description (found 0)
code_with_lint.py:34 in public class `SomeClass`:
        D101: Missing docstring in public class
code_with_lint.py:36 in public method `__init__`:
        D107: Missing docstring in __init__
codio@abrahamrachel-alamoengine:~/workspace$
```

The output from this tool is useful and something I will leverage when preparing the *final* version of code to fellow developers. The reference to “One-line docstring should fit on one line with quotes” is handy to know as well as the consideration that the first line of document strings should end with a period.