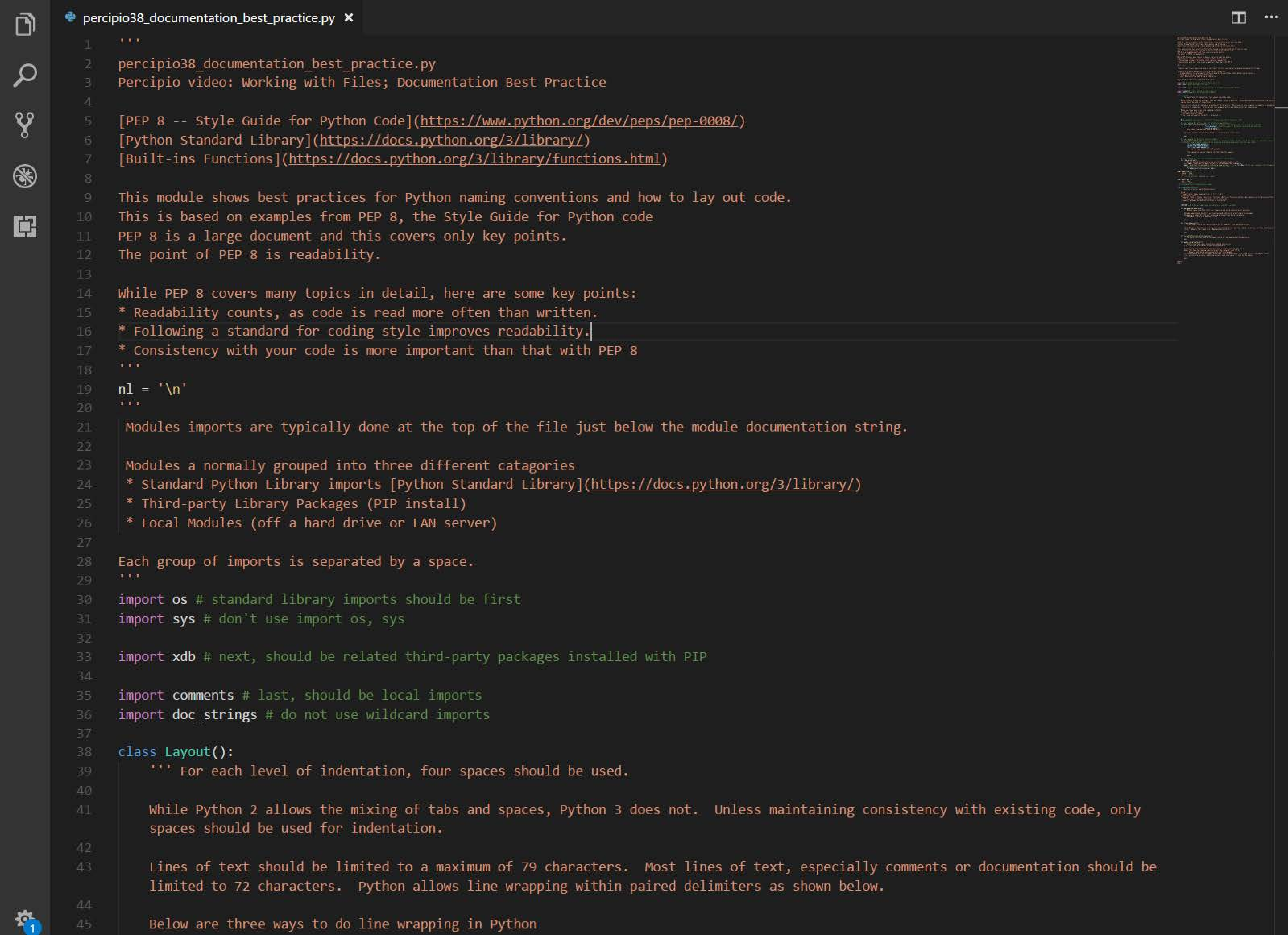
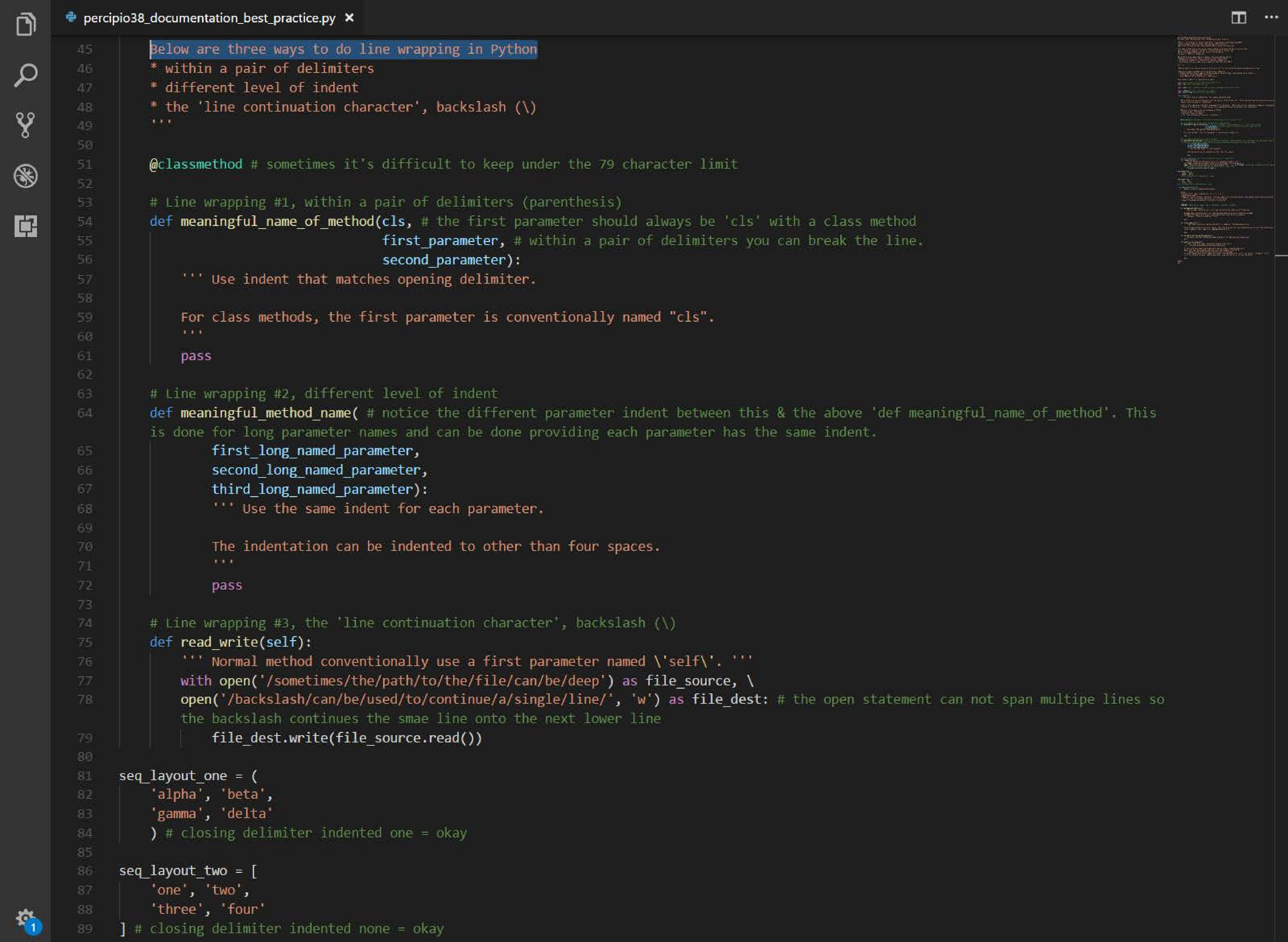
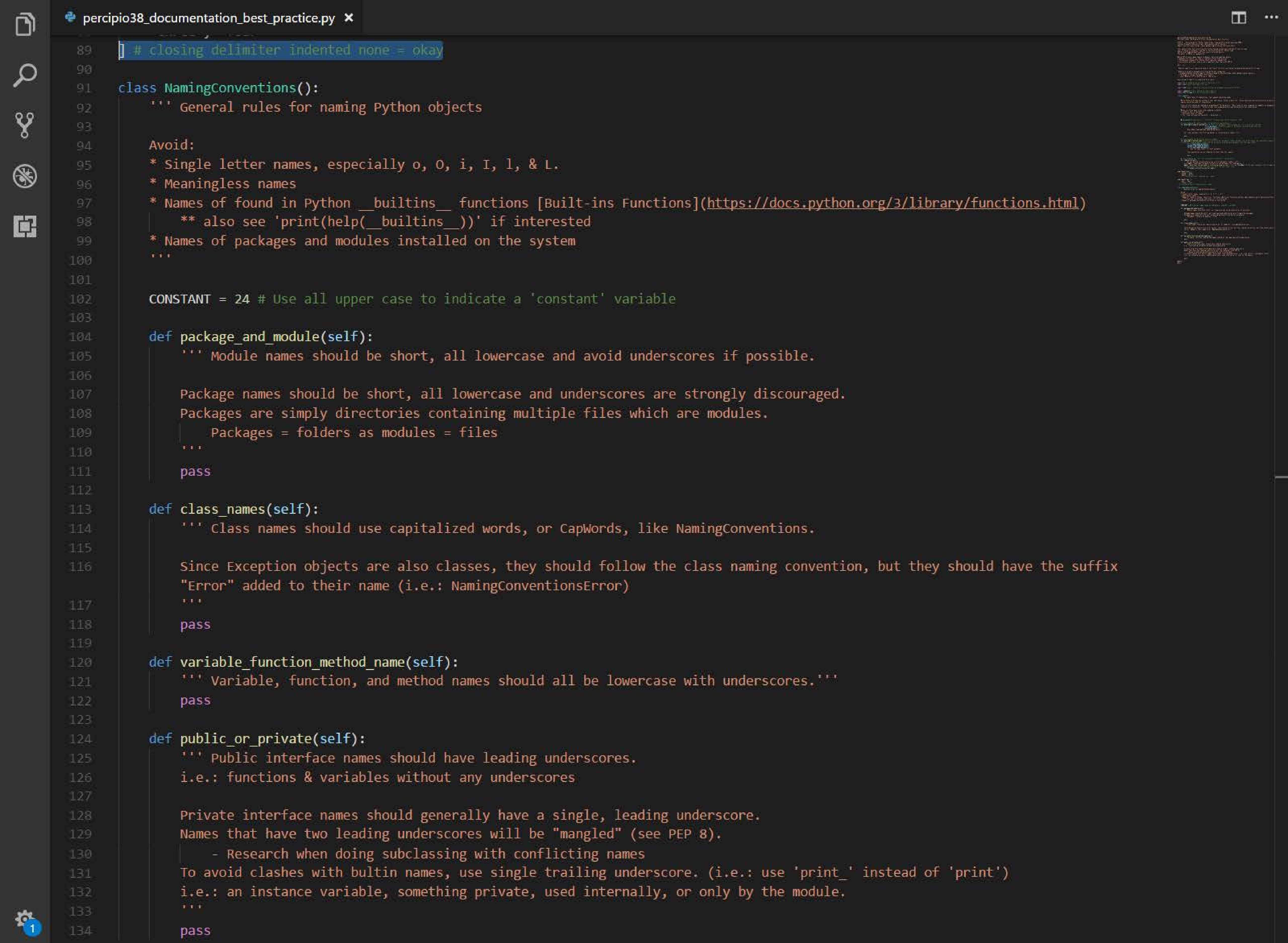


```python
'''
percipio38_documentation_best_practice.py
Percipio video: Working with Files; Documentation Best Practice

[PEP 8 -- Style Guide for Python Code](https://www.python.org/dev/peps/pep-0008/)
[Python Standard Library](https://docs.python.org/3/library/)
[Built-ins Functions](https://docs.python.org/3/library/functions.html)

This module shows best practices for Python naming conventions and how to lay out code.
This is based on examples from PEP 8, the Style Guide for Python code
PEP 8 is a large document and this covers only key points.
The point of PEP 8 is readability.

While PEP 8 covers many topics in detail, here are some key points:
* Readability counts, as code is read more often than written.
* Following a standard for coding style improves readability.
* Consistency with your code is more important than that with PEP 8
'''
nl = '\n'
'''
 Modules imports are typically done at the top of the file just below the module documentation string.

 Modules a normally grouped into three different catagories
 * Standard Python Library imports [Python Standard Library](https://docs.python.org/3/library/)
 * Third-party Library Packages (PIP install)
 * Local Modules (off a hard drive or LAN server)

Each group of imports is separated by a space.
'''
import os # standard library imports should be first
import sys # don't use import os, sys

import xdb # next, should be related third-party packages installed with PIP

import comments # last, should be local imports
import doc_strings # do not use wildcard imports

class Layout():
    ''' For each level of indentation, four spaces should be used.

    While Python 2 allows the mixing of tabs and spaces, Python 3 does not.  Unless maintaining consistency with existing code, only spaces should be used for indentation.

    Lines of text should be limited to a maximum of 79 characters.  Most lines of text, especially comments or documentation should be limited to 72 characters.  Python allows line wrapping within paired delimiters as shown below.

    Below are three ways to do line wrapping in Python
```



```python
    Below are three ways to do line wrapping in Python
    * within a pair of delimiters
    * different level of indent
    * the 'line continuation character', backslash (\)
    '''

    @classmethod # sometimes it's difficult to keep under the 79 character limit

    # Line wrapping #1, within a pair of delimiters (parenthesis)
    def meaningful_name_of_method(cls, # the first parameter should always be 'cls' with a class method
                                  first_parameter, # within a pair of delimiters you can break the line.
                                  second_parameter):
        ''' Use indent that matches opening delimiter.

        For class methods, the first parameter is conventionally named "cls".
        '''
        pass

    # Line wrapping #2, different level of indent
    def meaningful_method_name( # notice the different parameter indent between this & the above 'def meaningful_name_of_method'. This is done for long parameter names and can be done providing each parameter has the same indent.
            first_long_named_parameter,
            second_long_named_parameter,
            third_long_named_parameter):
            ''' Use the same indent for each parameter.

            The indentation can be indented to other than four spaces.
            '''
            pass

    # Line wrapping #3, the 'line continuation character', backslash (\)
    def read_write(self):
        ''' Normal method conventionally use a first parameter named \'self\'. '''
        with open('/sometimes/the/path/to/the/file/can/be/deep') as file_source, \
        open('/backslash/can/be/used/to/continue/a/single/line/', 'w') as file_dest: # the open statement can not span multipe lines so the backslash continues the smae line onto the next lower line
            file_dest.write(file_source.read())

seq_layout_one = (
    'alpha', 'beta',
    'gamma', 'delta'
    ) # closing delimiter indented one = okay

seq_layout_two = [
    'one', 'two',
    'three', 'four'
] # closing delimiter indented none = okay
```



```python
] # closing delimiter indented none = okay

class NamingConventions():
    ''' General rules for naming Python objects

    Avoid:
    * Single letter names, especially o, O, i, I, l, & L.
    * Meaningless names
    * Names of found in Python __builtins__ functions [Built-ins Functions](https://docs.python.org/3/library/functions.html)
        ** also see 'print(help(__builtins__))' if interested
    * Names of packages and modules installed on the system
    '''

    CONSTANT = 24 # Use all upper case to indicate a 'constant' variable

    def package_and_module(self):
        ''' Module names should be short, all lowercase and avoid underscores if possible.

        Package names should be short, all lowercase and underscores are strongly discouraged.
        Packages are simply directories containing multiple files which are modules.
            Packages = folders as modules = files
        '''
        pass

    def class_names(self):
        ''' Class names should use capitalized words, or CapWords, like NamingConventions.

        Since Exception objects are also classes, they should follow the class naming convention, but they should have the suffix
        "Error" added to their name (i.e.: NamingConventionsError)
        '''
        pass

    def variable_function_method_name(self):
        ''' Variable, function, and method names should all be lowercase with underscores.'''
        pass

    def public_or_private(self):
        ''' Public interface names should have leading underscores.
        i.e.: functions & variables without any underscores

        Private interface names should generally have a single, leading underscore.
        Names that have two leading underscores will be "mangled" (see PEP 8).
            - Research when doing subclassing with conflicting names
        To avoid clashes with bultin names, use single trailing underscore. (i.e.: use 'print_' instead of 'print')
        i.e.: an instance variable, something private, used internally, or only by the module.
        '''
        pass
```