Guide to Pylint Errors

Help Center

Pylint is a source code bug and quality checker for the Python programming language. The machine-graders for this class will use a customized version of Pylint to check the style of your code. Pylint errors due to violations of the style guidelines will result in deductions from the final score for your mini-project.
If you would like to experiment with Pylint, you are welcome to use this Owltest page to submit programs directly to Pylint for style checking. Here are explanations of some common warning message that Pylint will display.

- Missing docstring for program: Pylint warns you if your program does not begin with a docstring (which violate the style guidelines). This program generates the warning `"[line 1] Missing docstring"`. Adding a docstring at the top of the program will remedy this error.
- Non-compliant variables names: Pylint warns you if your variable names do not conform to the style guidelines. In particular, all variable names should consist of three or more lower case letters or underscores. For example, this program will generate the warning `"Invalid name "n" (should match [a-z_][a-z0-9_]{2,30}$)"` which signals that the parameter name `n` does not meet the style guidelines. Renaming `n` to `number` in the program avoids the warning.
- Use of global variables: Pylint warns you if you are using global variables (which violate the style guidelines). Since global constants are allowed, Pylint will warn you that they should be named using all capital letters. For example, this program generates the warning `" Invalid name "canvas_width" (should match (([A-Z_][A-Z0-9_]*)|(__.*__))$)"`. Capitalizing the names of the two global constants yields a program free of warnings.
- Missing docstring for a function or class definition: Pylint will warn you if your function or class definition does not contain a docstring (which violate the style guidelines). For example, this program generates the warning `"[line 5] Missing docstring"`. Adding a docstring to the function definition yields a program that avoids this warning.
- Code refactoring warnings: Pylint analyzes your code and check for long, complex blocks of code. If the complexity of these blocks exceeds Pylint's built-in limits, Pylint will issue a warning asking you to refactor (rewrite) your code in smaller pieces using several functions or methods. For example, this program generates the warning `"Too many return statements (10/6)"`. In this case, rewriting this program to use the remainder operator avoids the warning.
- Redefining built-in function names: Pylint warns you if you attempt to redefine the name of a built-in Python function. For example, this program will generate the warning `"Redefining built-in 'format'"` that signals that you are trying to redefine the built-in `format` function in Python. Renaming `format` to `template` in the program avoids the warning. (Yes, we did this in the template for Stopwatch for IIPP. We also write bad code sometimes.)
- Unused variables: Pylint warns you if a variable in your code is unused (which may indicate a problem in your code). A common situation where this issues arises is when using list comprehensions to create lists with constant values. This example will cause Pylint to report the error `"[line 10] Unused variable 'index'"`. Renaming the variable with the prefix `dummy_...` will inform Pylint that this variable is intentionally being used as a dummy variable. This program

will pass Pylint's style check.

- Access to a protected member: Pylint warns you if you attempt to access a class field outside the class definition. This example will cause Pylint to report the error `"[line 16] Access to a protected member _name of a client class"`. (Pylint also gives a refactoring warning about having no public methods for the class.) Adding the method `get_name` (and a second public method) yields this program that will pass Pylint's style check.
- Invalid name for class field: Pylint warns you if you fail to use a leading underscore on class fields to indicate that they are private. This example will cause Pylint to report the error `"[line 12] Invalid name "name" (should match _[a-z0-9_]{3,30}$) function "Widget.__init__", line 12"`. Adding a leading underscore to the field `name` yields this program that will pass Pylint's style check.

Created Tue 22 Apr 2014 3:07 PM PDT

Last Modified Wed 18 Jun 2014 1:10 PM PDT