

Pylint Report

What is Pylint?

It is a Python static code analysis tool that looks for programming errors, helps to enforce a coding standard, sniffs for code smells and offers simple refactoring suggestions.

It's highly configurable, having special pragmas to control its errors and warnings from within your code, as well as from an extensive configuration file. It is also possible to write your plugins for adding your checks or for extending pylint in one way or the other.

How to install pylint?

1. For python 2.7.9 and below:
 - a. Download the source distribution version 1.9.5 from the given link <https://github.com/PyCQA/pylint/releases/tag/pylint-1.9.5>
 - b. Extract all the files and open the terminal/ command prompt in the extracted folder
 - c. Run the command `python setup.py install`
2. For python 2.7.9 and above :
 - a. Open the command prompt/ terminal
 - b. Run the command `pip install pylint`
 - c. The user can also download the latest version from the source distribution (link: <https://github.com/PyCQA/pylint>) and follow the installation steps in method 1.

How to run Pylint?

1. Pylint is meant to be called from the command line. It can be used as `pylint [options] modules_or_packages` (make sure the path is proper)
2. We can also use pylint as `pylint directory/mymodule.py`
3. We can also achieve parallel execution by using the following command `pylint -j 4 mymodule1.py mymodule2.py mymodule3.py mymodule4.py`
4. To run the pylint for the entire package use command `find . -type f -name "*.py" | xargs pylint`.
5. `pylint directory/mymodule.py` can be also used if the directory is a python package (i.e. has an `__init__.py` file or it is an implicit namespace package) or if the “directory” is in the python path.

How is the output?

```
C:\Users\Amanjot\Desktop\test>pylint a.py
***** Module a
a.py:1:17: C0326: Exactly one space required after comma
brace_open = ["[","{","("]
               ^ (bad-whitespace)
a.py:1:21: C0326: Exactly one space required after comma
brace_open = ["[","{","("]
               ^ (bad-whitespace)
a.py:2:18: C0326: Exactly one space required after comma
brace_close = ["]","}","")
               ^ (bad-whitespace)
a.py:2:22: C0326: Exactly one space required after comma
brace_close = ["]","}","")
               ^ (bad-whitespace)
a.py:5:10: C0303: Trailing whitespace (trailing-whitespace)
a.py:5:7: C0326: Exactly one space required around assignment
    lst=[]
         ^ (bad-whitespace)
a.py:6:15: C0303: Trailing whitespace (trailing-whitespace)
a.py:10:13: C0326: Exactly one space required around assignment
    p=brace_close.index(i)
         ^ (bad-whitespace)
a.py:11:25: C0326: Exactly one space required around comparison
    if ((len(lst)>0) and (brace_open[p]==lst[len(lst)-1])):
        ^ (bad-whitespace)
a.py:11:47: C0326: Exactly one space required around comparison
    if ((len(lst)>0) and (brace_open[p]==lst[len(lst)-1])):
        ^^ (bad-whitespace)
a.py:21:6: C0326: Exactly one space required around assignment
string='[{({})}]'
        ^ (bad-whitespace)
a.py:22:3: C0326: Exactly one space required around assignment
sol=check(string)
    ^ (bad-whitespace)
a.py:23:0: C0304: Final newline missing (missing-final-newline)
a.py:1:0: C0114: Missing module docstring (missing-module-docstring)
a.py:4:0: C0103: Argument name "s" doesn't conform to snake_case naming style (invalid-name)
a.py:4:0: C0116: Missing function or method docstring (missing-function-docstring)
a.py:10:12: C0103: Variable name "p" doesn't conform to snake_case naming style (invalid-name)
a.py:16:8: R1705: Unnecessary "else" after "return" (no-else-return)
a.py:21:0: C0103: Constant name "string" doesn't conform to UPPER_CASE naming style (invalid-name)
a.py:22:0: C0103: Constant name "sol" doesn't conform to UPPER_CASE naming style (invalid-name)

-----
Your code has been rated at -1.11/10 (previous run: -1.11/10, +0.00)
```

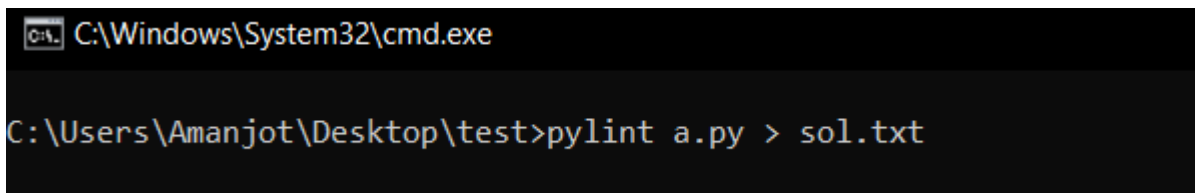
Fig: 1

The above image is the output on running a pylint on a single module.

We can get the output in the same format of various commands mentioned in the “How to Run” section.

Note: To save the output in a different file we can use the redirection operator “>” common for both linux based os and windows.

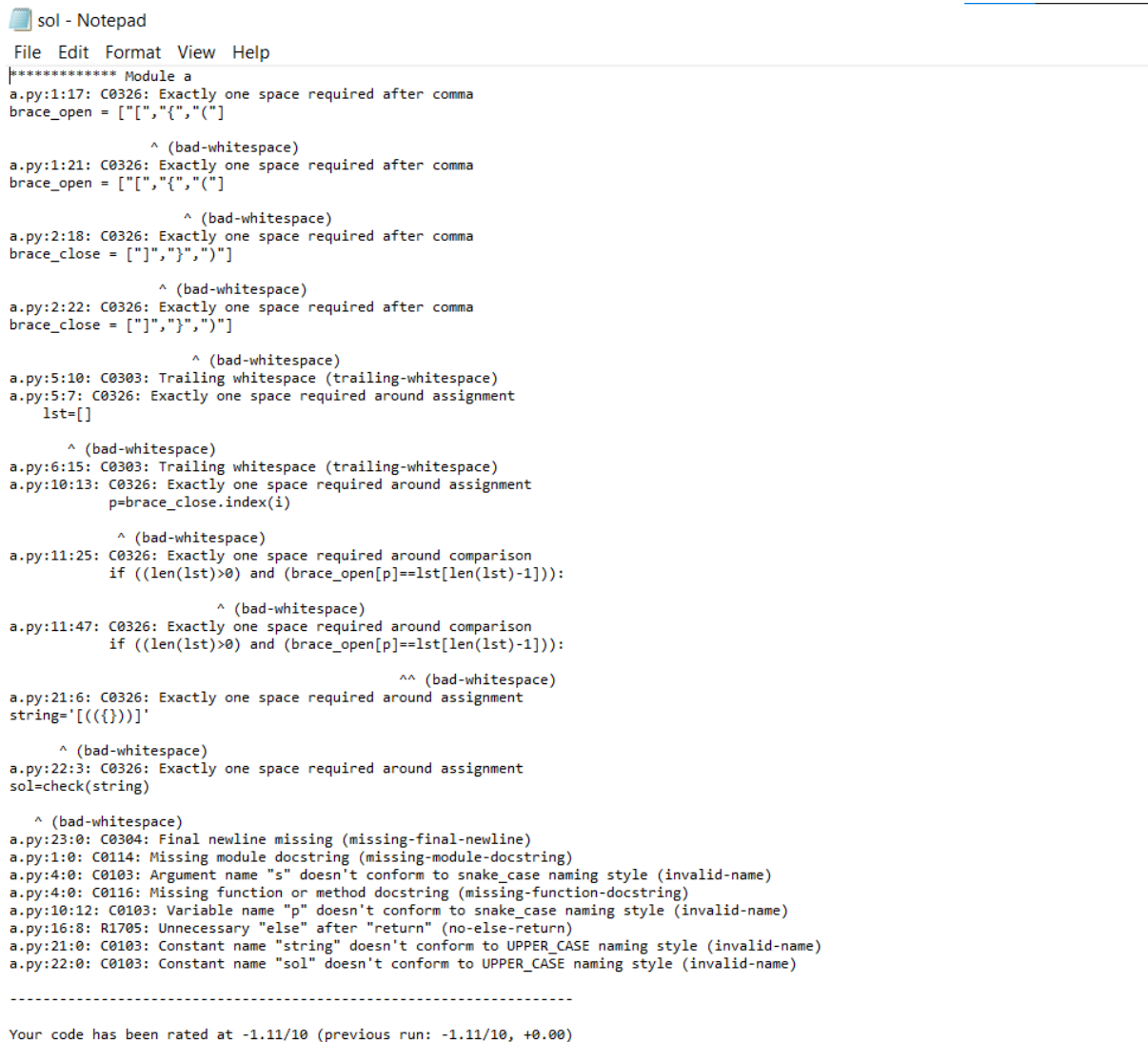
For example:



```
C:\Windows\System32\cmd.exe

C:\Users\Amanjot\Desktop\test>pylint a.py > sol.txt
```

Fig: 2



```
sol - Notepad
File Edit Format View Help
|***** Module a
a.py:1:17: C0326: Exactly one space required after comma
brace_open = ["{", "{", "("]
                ^ (bad-whitespace)
a.py:1:21: C0326: Exactly one space required after comma
brace_open = ["{", "{", "("]
                ^ (bad-whitespace)
a.py:2:18: C0326: Exactly one space required after comma
brace_close = ["]", "}", ")"]
                ^ (bad-whitespace)
a.py:2:22: C0326: Exactly one space required after comma
brace_close = ["]", "}", ")"]
                ^ (bad-whitespace)
a.py:5:10: C0303: Trailing whitespace (trailing-whitespace)
a.py:5:7: C0326: Exactly one space required around assignment
lst=[]
            ^ (bad-whitespace)
a.py:6:15: C0303: Trailing whitespace (trailing-whitespace)
a.py:10:13: C0326: Exactly one space required around assignment
p=brace_close.index(i)
            ^ (bad-whitespace)
a.py:11:25: C0326: Exactly one space required around comparison
if ((len(lst)>0) and (brace_open[p]==lst[len(lst)-1])):
            ^ (bad-whitespace)
a.py:11:47: C0326: Exactly one space required around comparison
if ((len(lst)>0) and (brace_open[p]==lst[len(lst)-1])):
            ^ (bad-whitespace)
a.py:21:6: C0326: Exactly one space required around assignment
string='[{({})}]'
            ^^ (bad-whitespace)
a.py:22:3: C0326: Exactly one space required around assignment
sol=check(string)
            ^ (bad-whitespace)
a.py:23:0: C0304: Final newline missing (missing-final-newline)
a.py:1:0: C0114: Missing module docstring (missing-module-docstring)
a.py:4:0: C0103: Argument name "s" doesn't conform to snake_case naming style (invalid-name)
a.py:4:0: C0116: Missing function or method docstring (missing-function-docstring)
a.py:10:12: C0103: Variable name "p" doesn't conform to snake_case naming style (invalid-name)
a.py:16:8: R1705: Unnecessary "else" after "return" (no-else-return)
a.py:21:0: C0103: Constant name "string" doesn't conform to UPPER_CASE naming style (invalid-name)
a.py:22:0: C0103: Constant name "sol" doesn't conform to UPPER_CASE naming style (invalid-name)

-----
Your code has been rated at -1.11/10 (previous run: -1.11/10, +0.00)
```

Fig: 3

- We see that after running the command in figure 2 we get a file saved (figure 3) in the same package in the format that we need it in (.json, .html, .txt, etc.).
- The output format and view can also be changed into the format that we need it in.
- More information on the same can be seen over here https://pylint.pycqa.org/en/latest/user_guide/output.html
- Integrating pylint with various editors and Ide can also be done.

Output With report:

<http://dontpad.com/pylintKhu>

PyLint vs Flake8

Flake8:

It's a wrapper around code style checker, error checker and complexity checker

- ★ **pycodestyle** - style checker for PEP8 compliance. PEP8 is a set of rules for how to format your Python code to maximize its readability. It acts as a base for a company/project/team-specific code style. Error codes are E/W/N8**
- ★ **pyflakes** - checks for errors or violation of common non-stylistic practices. For example, things like “module imported but unused”. See full list [here](#)
- ★ **McCabe complexity checker**. Complexity is a quantitative measure of the number of linearly independent paths through a program's source code. It was developed by Thomas J. McCabe, Sr. in 1976. So basically nested if statements inside for loops will blow up this number when the threshold for a good code is less than 10.

Pylint:

- ★ From the authors of flake8 - pylint, checks for PEP8-like code style, some code smells and type errors. It intersects with flake8 in many regards, but doesn't check for complexity (can be enabled through the plugin `pylint.extensions.mccabe`).
- ★ Pylint doesn't just have the best range of features, but it's also constantly maintained, making it a must-have tool for Python developers. It's been around for 13 years, and over that time it has included features like coding standards, error detection and refactoring by detecting duplicated code.
- ★ It is easy to set up, requiring a minimal amount of configuration, but it's fully customizable if you want it to be — by editing a file you can select which errors and conventions are most relevant to you.