Python / `pylint`

For Python, Pylint is used, which is a static code analysis tool and linter for Python that ensures code is idiomatic and follows Python best practices such as PEP-8. It is also used to check code for errors and can help prevent difficult-to-find runtime bugs. In this class, **we have disabled certain pylint checks to make the checks less strict**. If desired, these checks can be re-enabled to check for all pylint categories (*see **Running with all checks enabled***). There are a few different ways of running pylint, each of which are detailed in the below sections.

Documentation is available on the Python style `pylint` checks for:

- Pylint docs - each individual warning/error/refactor
- PEP-8 Style guide - Python best practices for code style

In addition, there are some examples of compliant/noncompliant example code provided:

- `object.py` (non-compiliant) - example Flask `Resource` object with bad formatting
- `object_formatted` (compliant) - same as above, plus it passes all pylint checks

Running `pylint` with Script *(Recommended)*

A utility script was developed by the CS 2340 TAs to make running pylint on your projects easier.

Prerequisites

- Python **3** installed and on the `PATH` (tutorial on Canvas... complete the "Installing Python" section)
- Pylint script downloaded
- `pylint` installed using `pip`:
  - `python -m pip install pylint`

Running

The `run_pylint.py` script supports running pylint over every python file in a directory. For example, if my project structure is as follows:

```
project/
├── src/
│   ├── models/
│   │   └── object.py
│   ├── api.py
│   └── app.py
└── run_pylint.py
```

then I can run `pylint` on every `.py` file in the project directory (except for the script):

```
python run_pylint.py
```

This will output something like the following:

```
~/project/ $ python run_pylint.py

Running pylint on 3 files:

************* Module object
 src/models/object.py:9: convention (C0326, bad-whitespace, ) Exactly one space required after :
        if name in objects.keys():    return objects[name], 200
                                  ^
 src/models/object.py:18: convention (C0326, bad-whitespace, ) Exactly one space required around
 assignment
                objects[name]=args.get('value')
                             ^
 src/models/object.py:5: convention (C0103, invalid-name, formatMessage) Function name
 "formatMessage" doesn't conform to snake_case naming style
 src/models/object.py:9: convention (C0321, multiple-statements, Object.get) More than one statement
 on a single line

--------------------------------------------------------------------------------------
Your code has been rated at 8.10/10 [raw score: 8.10/10] (previous run: 8.10/10, +0.00)
```

The score given at the bottom is a metric of overall code quality, and is computed using the following formula where `e` is the number of pylint errors, `w` is the number of pylint warnings/conventions/refactors, and `n` is the number of Python statements in the scanned code:

$$\mathcal{S}(e, w, n) = 10 \left(1 - \frac{5e + w}{n} \right)$$

Running with all checks enabled

To run with all checks, add `--all` to the end of the command used to run pylint:

```
python run_pylint.py --all
```

This will likely result in a lower score than without all checks enabled, but **this score will not be used when grading. Only the score that is a result of the standard run will be used**.

Running over another directory

To run the script on a directory other than the current working directory, specify a relative or absolute path using `--root path/to/folder`:

```
python run_pylint.py --root path/to/folder
```

Running `pylint` Directly

Prerequisites

- Python installed and on the `PATH` (tutorial on Canvas... complete the "Installing Python" section)
- `pylint` installed using `pip`:
  - `python -m pip install pylint`

Running

Once the required files are present, run the following command, adding each file to the end as necessary:

```
python -m pylint --disable=C0111,R1705,E0401,R0201,E1101 --const-naming-style=any file1.py file2.py
```

The program should output something similar to the following, where each pylint error is listed, along with its filename, line number, and column number (where applicable).:

```
$ python -m pylint --disable=C0111,R1705,E0401,R0201,E1101 --const-naming-style=any api.py app.py \
    models/object.py
 ************* Module object
 models/object.py:9: convention (C0326, bad-whitespace, ) Exactly one space required after :
         if name in objects.keys():    return objects[name], 200
                                    ^
 models/object.py:18: convention (C0326, bad-whitespace, ) Exactly one space required around
 assignment
                 objects[name]=args.get('value')
                              ^
 models/object.py:5: convention (C0103, invalid-name, formatMessage) Function name "formatMessage"
 doesn't conform to snake_case naming style
 models/object.py:9: convention (C0321, multiple-statements, Object.get) More than one statement
 on a single line

--------------------------------------------------------------------------------------
Your code has been rated at 8.10/10 [raw score: 8.10/10] (previous run: 8.10/10, +0.00)
```

Overall, this method is more complex and requires using platform-specific ways of finding every python file in a directory to run pylint on. For those reasons, we recommend using the `run_pylint.py` script as detailed above.

Running with all checks enabled

To run with all checks, remove the `--disable=...` and `--const-naming-style=any` flags from the command:

```
python -m pylint file1.py file2.py
```

Running via IDE/Editor Plugins

Plugins are available for `pylint` for a variety of different editors/IDEs. Some of the more popular ones are listed below:

- PyCharm - pylint plugin that creates editor inspections for pylint checks
- Visual Studio Code - vscode plugin that runs pylint automatically
  - Make sure to set the following setting to replicate the specific checks used for this class:
  - `"python.linting.pylintArgs": ["--disable=C0111,R1705,E0401,R0201,E1101"]`
- Atom - pylint plugin that leverages the Atom linters API to visualize errors in code
- Emacs - integration via `flymake`

> **Note**: it is recommended to run `pylint` via the script/directly at least once before submitting each milestone to make sure all errors are caught.