COMP1531

🐶 Software Engineering
3.2 - Testing - linting

In this lecture

Why?

- You can't manually fix your style forever - we need automated approaches

What?

- Linting
- Pylint

What is good style?

Programs must be written for people to read, and only incidentally for machines to execute – *Abelson & Sussman, "Structure and Interpretation of Computer Programs"*

Why care about style?

- Good style means that:
  - It's easier to follow the flow of code with consistent whitespace
  - It's easier to visually glance at code with similar patterns
  - It can be easier to detect bugs
    - E.G. If you force constants to be uppercase named, it's easy to spot a mutated uppercase variable

What is good style?

- Which of the following are good style?
  - Snake case vs camel case?
  - Tabs vs spaces?
  - More?

The only real tenet of good style is **consistency within a project**. The most important thing you can do is follow the conventions set out that have been already established.

A good software engineer can be agnostic to particular style decisions

Well who decides on style??

Typically when a group picks a style, they:

1. **Choose a style guide set out by a large organisation** (E.G. Google, Facebook, Microsoft, etc).
2. **(Optionally) modify specific style rules to satisfy any clear subjective opinions** (e.g. if most of an organisation strongly believes in spaces over tabs).

In COMP1531, the staff use standard style guides and provides them to students.

Linters: Enforcing style

In early computing courses you're given a style guide and told to manually make sure your code complies.

In proper software engineering projects we tend to "lint" code by using software that **statically analyses your code**

We call programs that lint code "linters"

What is "static analysis"?

Static analysis is the processing of analysing as much of your program as you can **before running it.**

E.G. C compilation contains elements of static analysis (e.g. type checking , unused variables, etc)

Linting python code consists of:

- Style issues (whitespace, indentation)
- Semantic issues (bad logic, potential bugs)

Because C is interpreted (no compile step) linting helps bridge the gap of some things missed out by compilers.

pylint

- A popular external tool for statically analysing python code
- Can detect errors, warn of potential errors, check against conventions, and give possible refactoring suggestions
- By default, it is **very** strict
- Can be configured to be more lenient through configuration

```python
name="Hayden"
scores=[99, 11,12]
for i in range ( len ( scores)):
  print(scores[i])
```

```
************* Module hello
hello.py:4:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)
hello.py:1:0: C0114: Missing module docstring (missing-module-docstring)
hello.py:1:0: C0103: Constant name "name" doesn't conform to UPPER_CASE naming style (invalid-name)
hello.py:3:0: C0200: Consider using enumerate instead of iterating with range and len (consider-using-enumerate)

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

fixing pylint issues

- Semantic issues often need to be fixed manually
- Style issues can be fixed either via:
  - Using IDE addons, such as auto-formatters for VSCode
  - Using tools like **autopep8** to parse your code

Let's install autopep8 with pip and try and clean the following:

```
name="Hayden"
scores=[99, 11,12]
for i in range ( len ( scores)):
  print(scores[i])
```

Controlling Messages

- Disable messages via the command line

```
$ pylint --disable=<checks> <files_to_check>
$ pylint --disable=missing-docstring <files>
```

- Disable messages in code; e.g.

```
if year % 4 != 0: #pylint: disable=no-else-return
```

- Disable messages via a config file
    - If a .pylintrc file is in the current directory it will be used
    - Can generate one with:

```
pylint <options> --generate-rcfile > .pylintrc
```

Usage in gitlab

- Pylint will be part of your labs from week 3 onwards, including usage in your project from iteration 2 onwards
- It will also be added to gitlab pipelines

Feedback