

# Intermediate Research Software Development in Python

## Setting the Scene

Why this course?

Have you ever thought:

- "there must be a better way to do this"
- "this software is getting in the way of my research"
- "why is it so difficult to get this program to run?"
- "this code is incomprehensible and really difficult to modify"
- "I screwed up my Python installation again and need to reinstall my OS"

## Intermediate Research Software Development

What you will be able to do at the end that should help your work:

- restructure existing code and design more robust software from scratch
- automate the process of testing and verifying software correctness
- support collaborations with others in a way that mimics a typical collaborative software development process
- get you ready to distribute your code for use by others

PSA: This is a Collaborative Learning Session

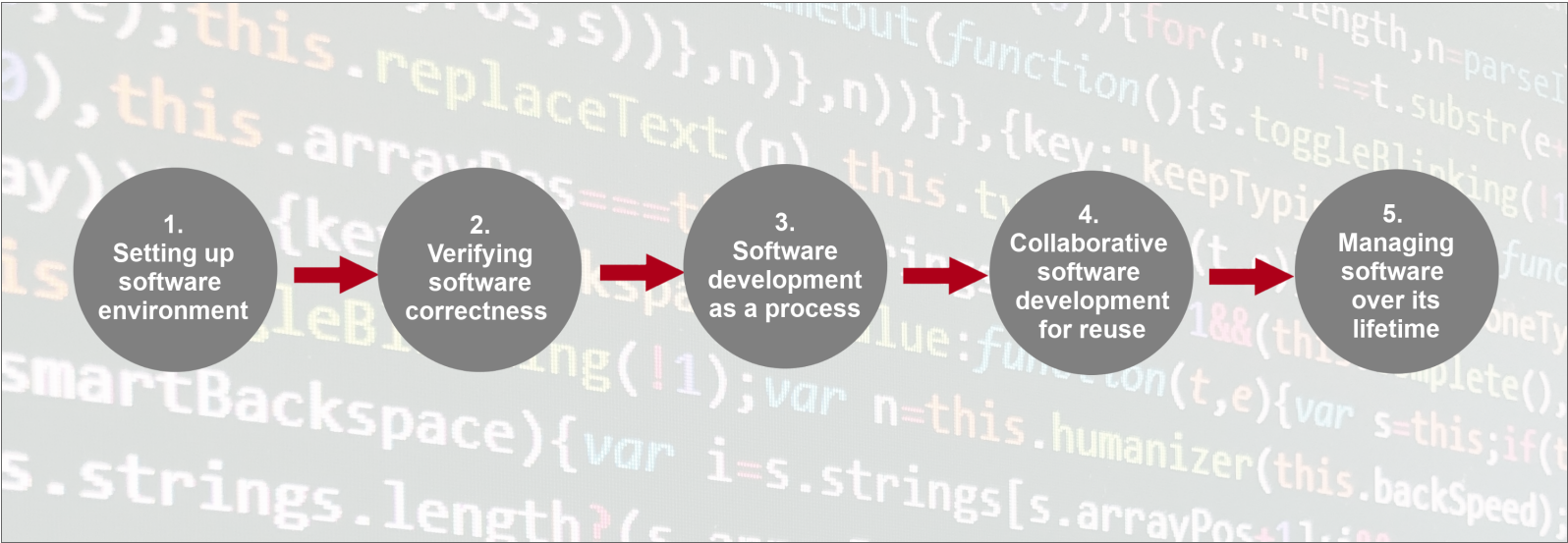
## Tools

- Python
- Integrated Development Environment: PyCharm or VS Code
- `pip` and `venv`
- GitHub

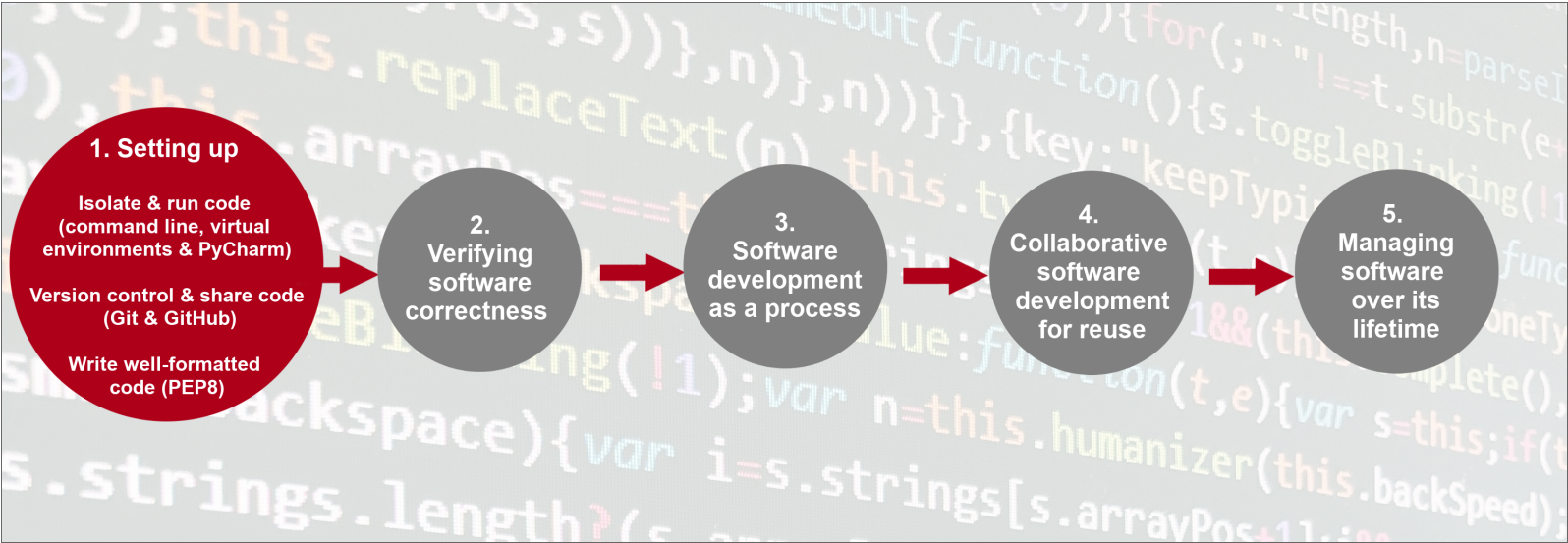
## Rules of Engagement

- Monitoring your status
  - self reporting
  - sporadic polls
- Questions at any time by raising hand 🙋
- Some lecturing by instructor combined with independent study, exercises, and group activities
- Take a break whenever you need it ☕

# Content Overview



# Section 1: Environment For Collaborative Code Development








Exercise:  Obtain the Software Project Locally

# Project Structure

```
.
├── data/
│   └── inflammation-*.csv
├── inflammation/
│   ├── models.py
│   └── views.py
├── inflammation-analysis.py
├── README.md
├── tests/
│   ├── test_models.py
│   └── test_patient.py
```

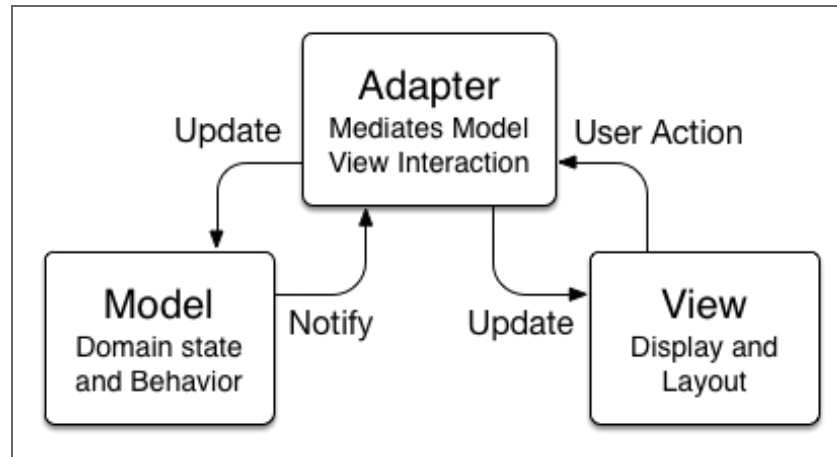
Exercise:  Have a Peak at the Data

Please post your answers in the shared document.

Software Architecture

**Theory covered later in Section 3: Software Design**

## Model-View-Adapter (MVA)



By Soroush Khanlou, <https://khanlou.com/2014/03/model-view-whatever/>

 5 Minute Break 

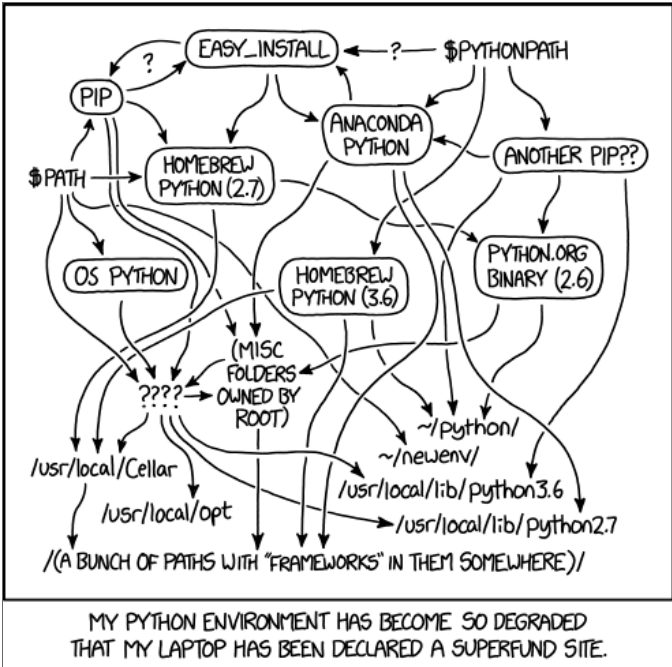
# Virtual Environments For Software Development



## Tools for Dependency Management

- For creating and managing virtual environments: `venv`
- For installing dependencies in those environments: `pip`

Lots of other tools...



Breakout Exercise: Creating a `venv` Environment

Read through and follow along until the end of the episode page.

Need to recreate your virtual environment?

```
rm -r venv/  
python3 -m venv venv  
source venv/bin/activate  
pip install <your_dependencies>  
# or  
pip install -r requirements.txt # great reason to have this file
```

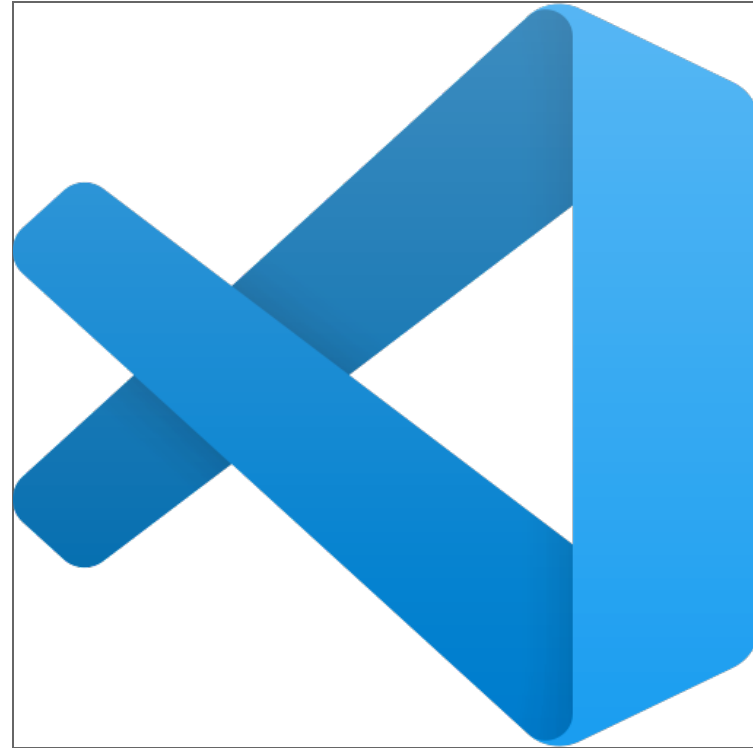
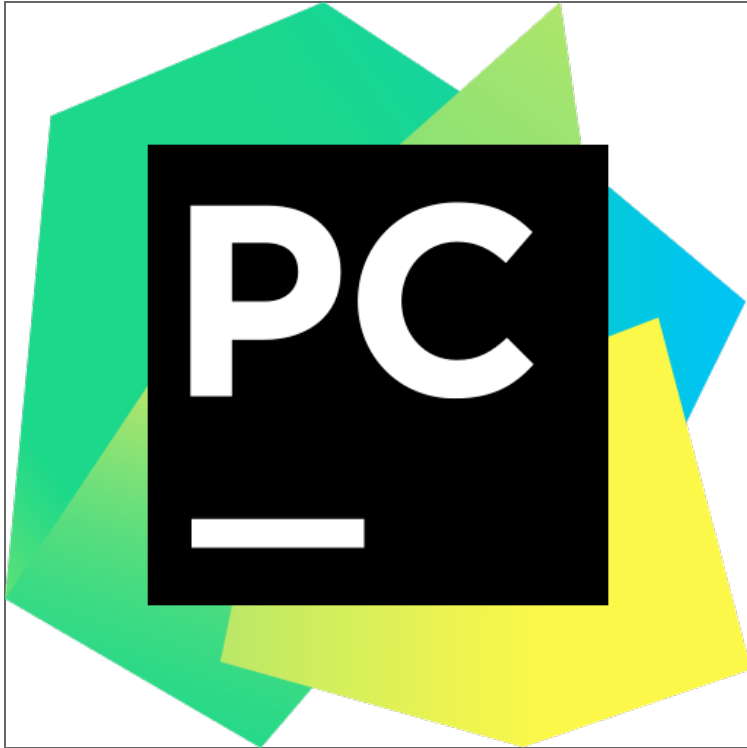
## Dependency Management in Other Languages

- Each will have its own way of handling this, and it will also depend on *where* you are doing your development
- The *coverall* option these days is to develop in a Docker container (or relevant analogue)
  - The `Dockerfile` codifies the dependencies and setup for your project
- If you are on a cluster, then you might be familiar with the `module` command
  - This allows you to get different versions of libraries without installing them yourself (and indeed, because you don't have permission to install them)
  - Spack and Easy Build are also quite popular package

management tools for HPC; Spack has virtual environments!

- C++
  - CMake is an ubiquitous build tool and overlaps with dependency management
  - Conan is a specific package manager for C++
  - Spack also a good option
- Fortran
  - Very nascent creation of the Fortran package manager (fpm) and probably more for modern Fortran
  - Spack again

# Integrated Development Environments



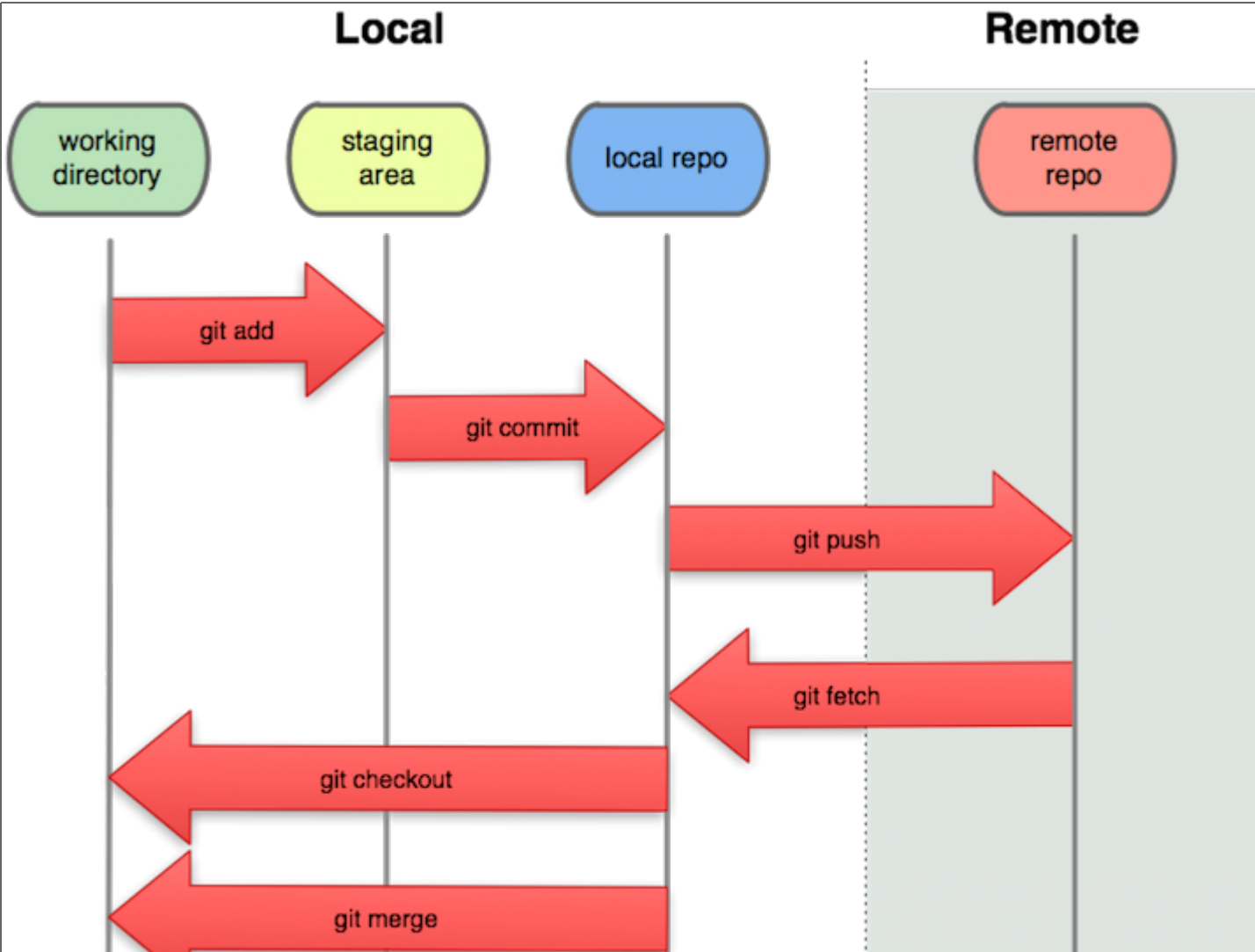
## Breakout Exercise: Using the PyCharm IDE

Start from this heading and continue to the end of the page.



 15 Minute Break 

# Collaborative Software Development Using Git and GitHub

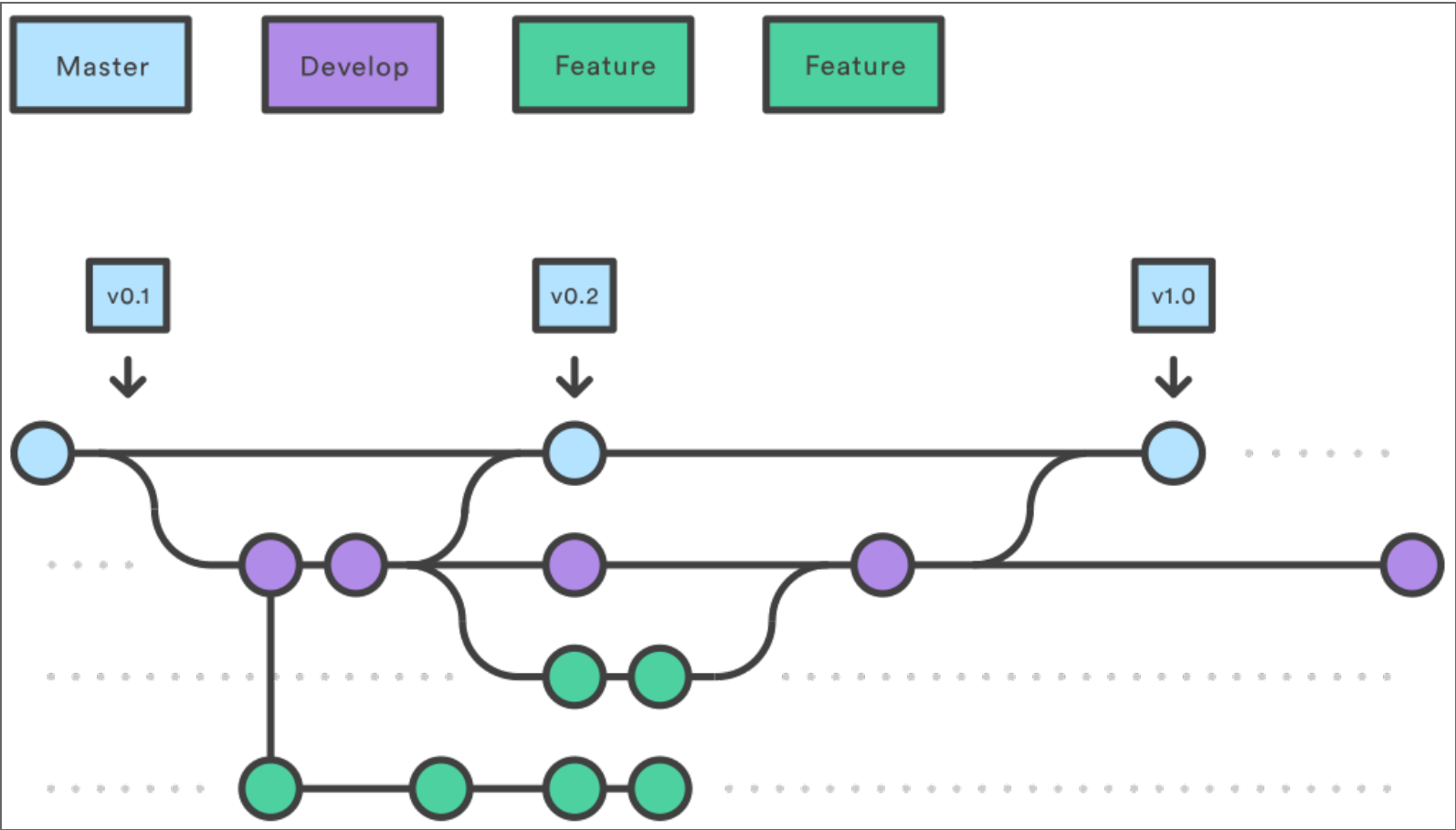




Breakout Exercise: Checking in Changes to Our Project

Start from this heading and go until the "Git Branches" heading.

# Git Branches



Breakout Exercise:  Creating Branches

Continue from this heading to the end of the page.

 5 Minute Break 

## Python Coding Conventions

*"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." — Martin Fowler*

- Coding *style* is one factor that makes our code more understandable
- Consistency is key



## Style in Different Languages and Tools

- Python: PEP8
  - `black`, `flake8`, `pylint`, etc...
- C++: no language-wide consensus
  - `clang-format` is widely used for enforcing formatting, and there are built-in presets for existing conventions followed by Google, LLVM, etc. Project specific settings made in a `.clang-format` file.
  - `cpplint` is another option
- Fortran: no language-wide consensus
  - some tools for VSCode
  - recent revival and there is a push towards modernising (best practices on new website)

## Breakout Exercise: Indentation

Start from this section heading and go to the end of the page.

```
"""  
Functions:  
    load_csv - Load a Numpy array from a CSV file  
    daily_mean - Calculate the daily mean of a 2D inflammation data array  
    daily_max - Calculate the daily max of a 2D inflammation data array  
    daily_min - Calculate the daily min of a 2D inflammation data array  
"""
```

 End of Section 1 

Please fill out the end-of-section survey!