Intermediate Research Software Development in Python

15 April 2024

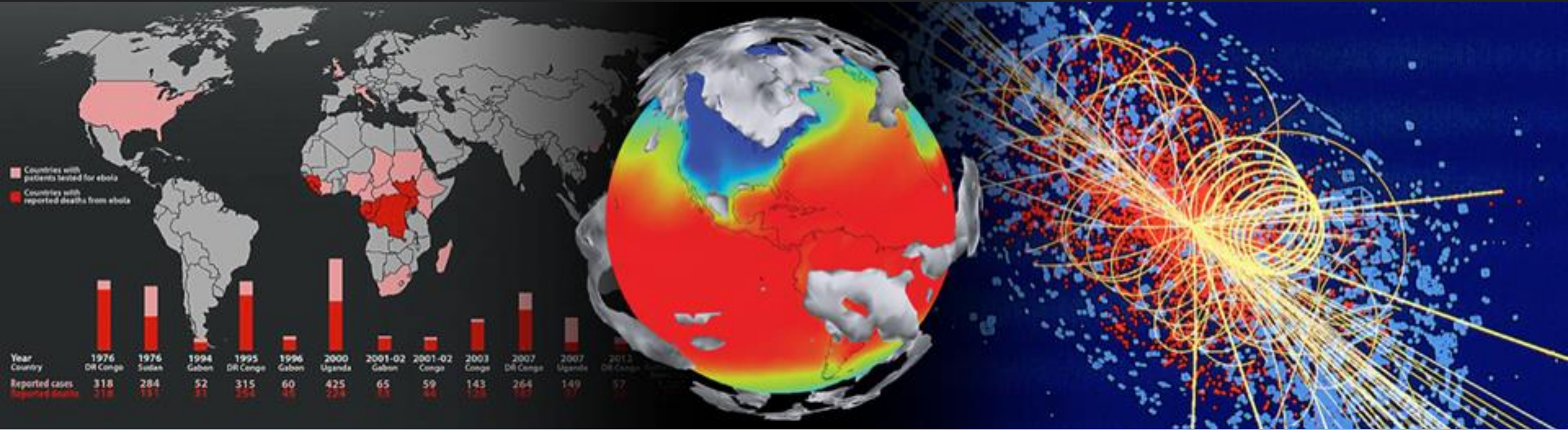
Slides borrowed from Aleksandra Nenadic,
Software Sustainability Institute

netherlands
eScience center
Digital Skills

Introduction

- What is this course about?
- Why should I consider how I engineer my projects?
- How is this course organized?

Modern research is impossible without software

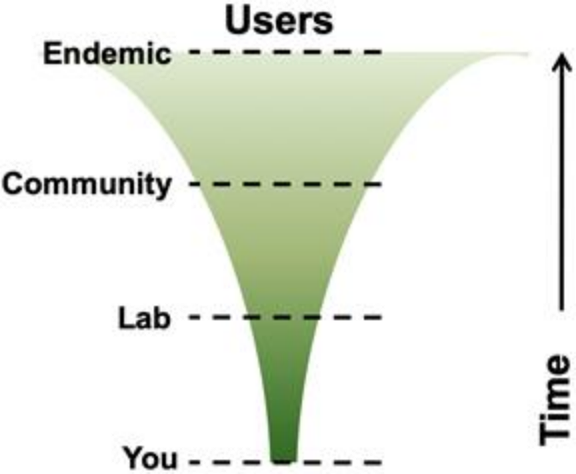


From thrown-together scripts, through an abundance of complex spreadsheets, to the millions of lines of code behind large-scale infrastructure, there are few areas where software does not play a fundamental part in research

The software you write is important!

- Software inherently contains *value,* not a throw-away artifact

- Difficult to gauge to what extent it might be used in the future
  - By who?
  - Which parts?
  - Which projects?
  - Reproducibility of results – from publications!



Can it/should it be reusable by others?
...including yourself?

When should you have considered how to engineer your software?

"The best time to plant a tree is 20 years ago. The second best time is now."

Programming vs engineering

Programming / Coding

- Focus is on one aspect of software development (implementation)
- Writes software for themselves
- Mostly an individual activity
- Writes software to fulfil research goals (ideally from a design)

Engineering

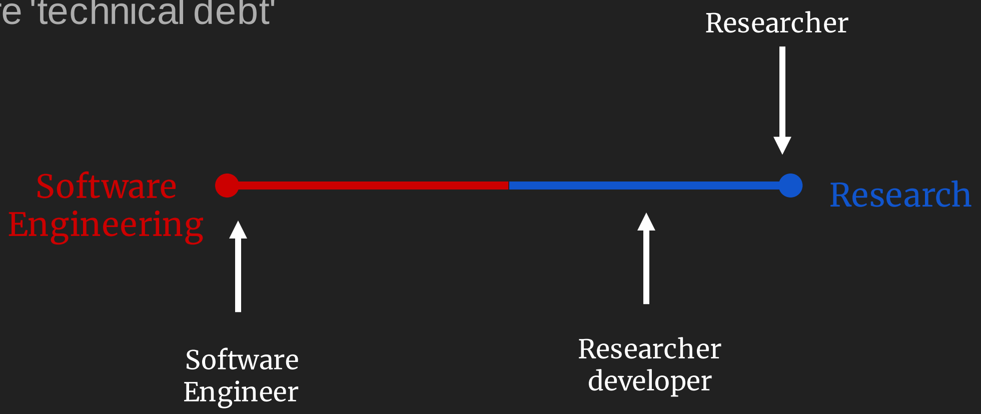
- Considers the *lifecycle* of software
- Writes software for *stakeholders*
- Takes *team ethic* into account
- Applies a *process* to understanding, designing, building, releasing, and maintaining software

"Programmers tend to start coding right away. Sometimes this works."

- Eric Larsen, 2018

Target audience

- **A researcher developer, with foundational coding skills, now facing new challenges**
    - Software projects becoming more complex
    - Scale of development increasing
    - More collaborative development and use, internal or external
    - Higher expectations on software
    - Increase in requests for change, more 'technical debt'



The story...

You have just joined a team that works on an existing software project. The projects is written in Python and studies and analyses inflammation in a group of patients that have been given a new treatment for arthritis in a medical trial. The software project is not finished and contains some bugs and tests that are failing. Over the course - you will work either individually or with your colleagues to develop more code and fix these errors.

Course outline

1. **Environment for Collaborative Code Development**
   - ○ Setting up our software project, development environments and tools, virtual environments, Git & GitHub, coding conventions
2. **Ensuring Correctness of Software at Scale**
   - ○ Automatically testing your software, unit tests, continuous integration, debugging, improving code robustness
3. **Software Development as a Process**
   - ○ Software requirements, design, architecture and implementation (via programming paradigms) as typical stages of a software development process
4. **Collaborative Software Development for Reuse**
   - ○ Code review, documenting and preparing/packaging software for release and reuse
5. **Managing and Improving Software Over Its Lifetime**
   - ○ Tracking issues, managing team communications and software project planning and management
6. **Apply learned skills to own project**

How will the workshop run

- **Delivery is helper-supported self-learning (no live coding)**
  - Learners go through material individually (at their own pace)
  - You work together in groups of 4-5 people
  - Trainers will be available to assist with problems and answer questions
  - Initially exercises are performed individually, more group/team work in later stages when we will need to keep more in sync with everyone in a group
  - Feel free to ask questions and start discussions with people in your group or instructors. This is a good opportunity to ask related questions about your daily work as well.
  - Take regular breaks as you need

A final note …

- **About working through the material**
  - Start from the actual section introduction - bigger picture
  - Exercises - try to do them on your own (before looking at the solution)
  - Exercises - don't skip them! Some later materials depend on this
    - Optional exercises are clearly marked - all other exercises should be completed
  - If you fall behind, ask a group member or instructor to walk you through so you are in sync with the rest

- **Please ask for help - do not suffer in silence!**
  - Ask for help if needed
  - We know from experience that some people don't like to read, especially not at the end of the day when you feel like everyone else is moving faster than you. Ask an instructor to explain instead of reading the texts!

What questions do you have?

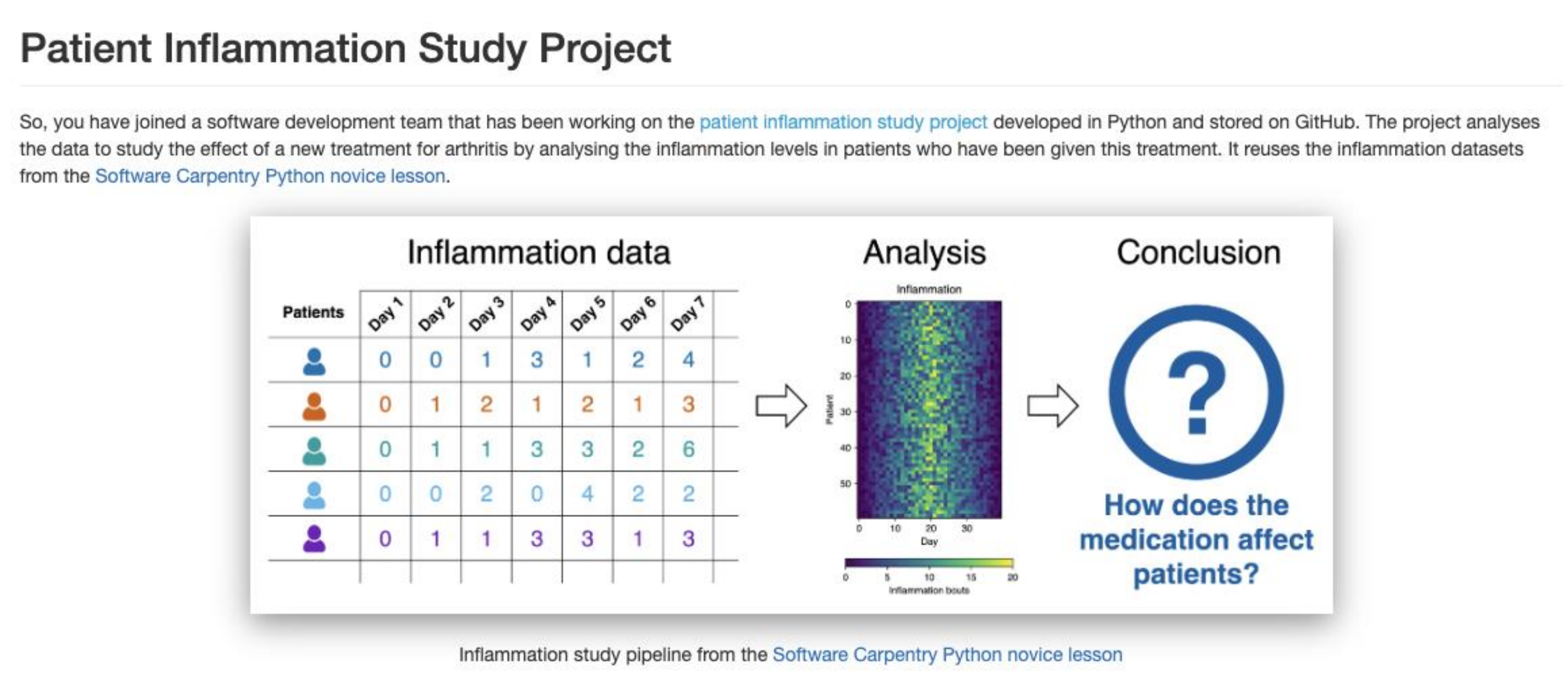
Let’s start!

Section 1: Setting Up Environment For Collaborative Code Development & Intro to Our Software Project

- **In order to develop (write, run, test, debug, backup) code efficiently, you need to use a number of different tools interchangeably**

- Introduction to software project and software development tools (refresher)
  - Command line, PyCharm IDE, Git + GitHub
- Setting up virtual development environment to isolate code (pip, venv)
- Writing and checking for well-formatted and readable code - PEP8 coding style convention & linting

"Anyone can write code that a computer can understand. Good programmers write code that humans can understand." *- Martin Fowler, a British software engineer, author and international speaker on software development*



Patient Inflammation Study Project

So, you have joined a software development team that has been working on the patient inflammation study project developed in Python and stored on GitHub. The project analyses the data to study the effect of a new treatment for arthritis by analysing the inflammation levels in patients who have been given this treatment. It reuses the inflammation datasets from the Software Carpentry Python novice lesson.

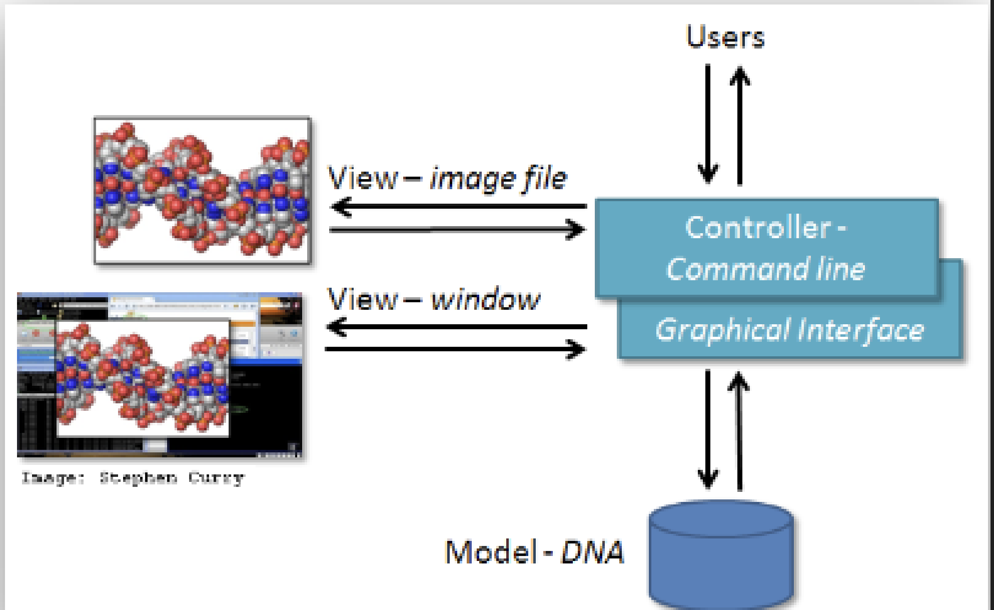
Patients	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
👤	0	0	1	3	1	2	4
👤	0	1	2	1	2	1	3
👤	0	1	1	3	3	2	6
👤	0	0	2	0	4	2	2
👤	0	1	1	3	3	1	3

How does the medication affect patients?

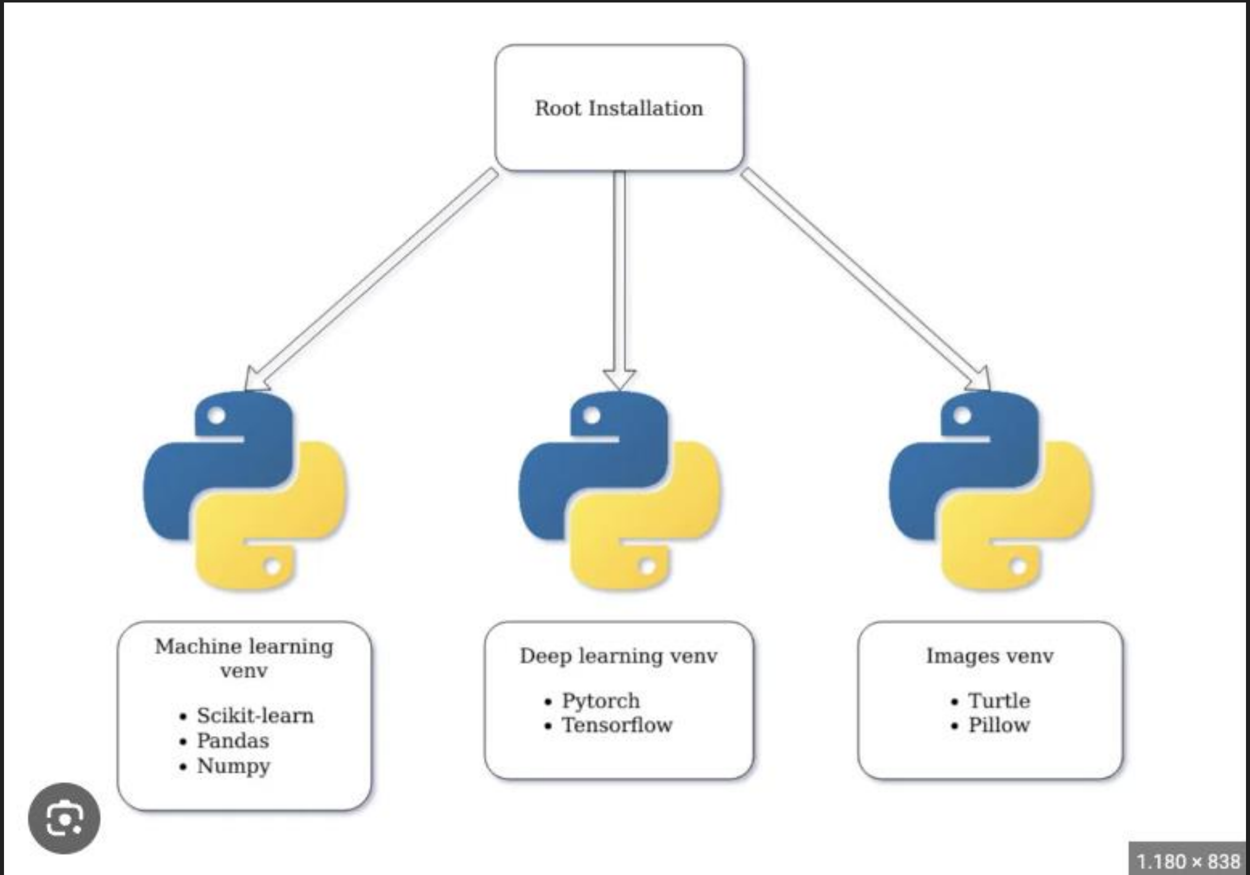
Inflammation study pipeline from the Software Carpentry Python novice lesson

Model – View – Controller architecture

- Model (data)
- View (client interface), and
- Controller (processes that handle input/output and manipulate the data).



Python virtual environments



Root Installation

Machine learning venv

- Scikit-learn
- Pandas
- Numpy

Deep learning venv

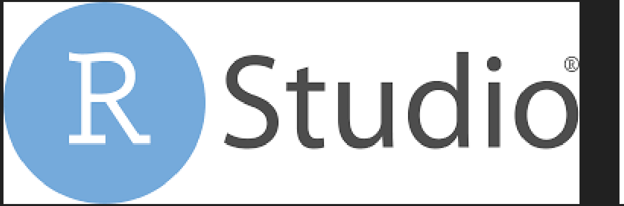
- Pytorch
- Tensorflow

Images venv

- Turtle
- Pillow

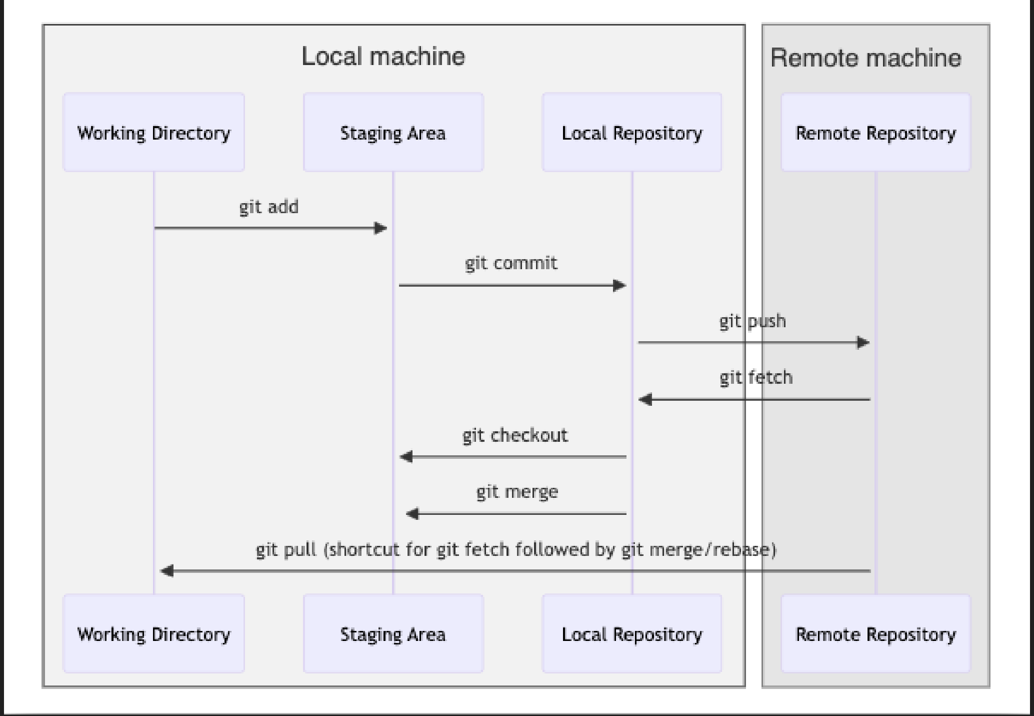
Integrated Development Environment (IDE)

- An Integrated Development Environment (IDE) is a very powerful editor that has a lot of useful extra features.
- If you do not want to you do not have to leave the environment while programming.
- What do you use? And why?

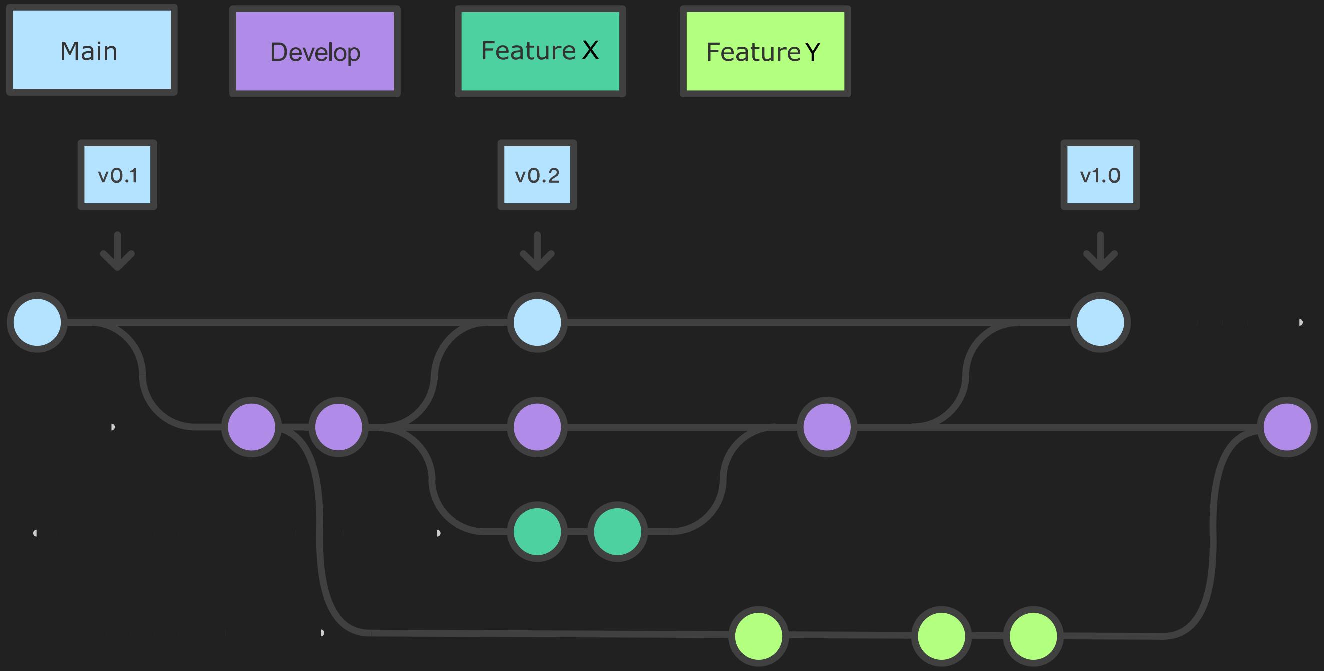


Visual Studio Code

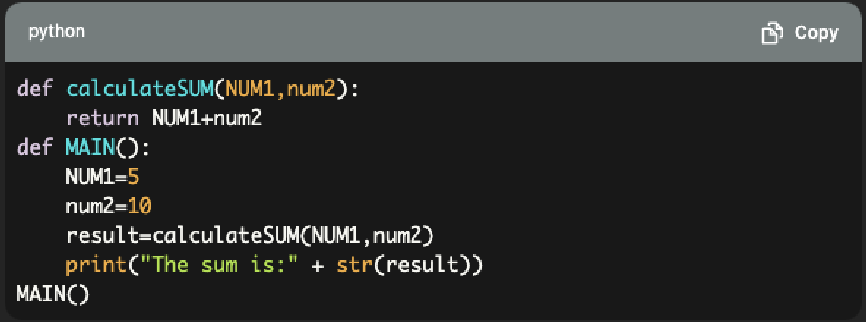
Software development lifecycle with git



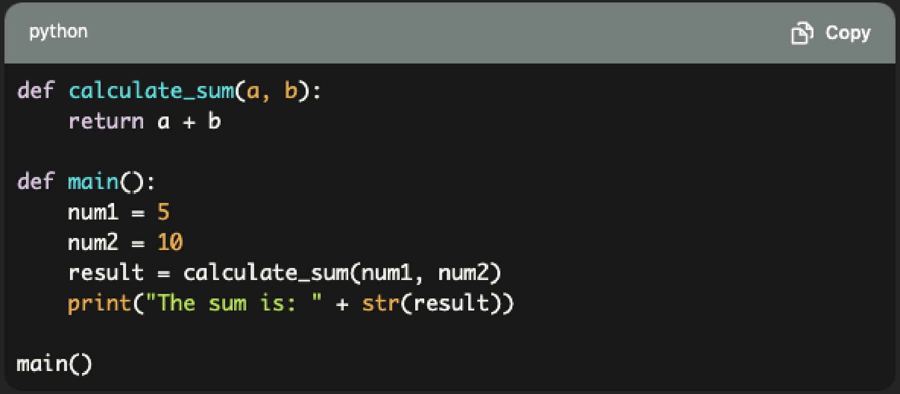
Git feature branches



Python code style conventions

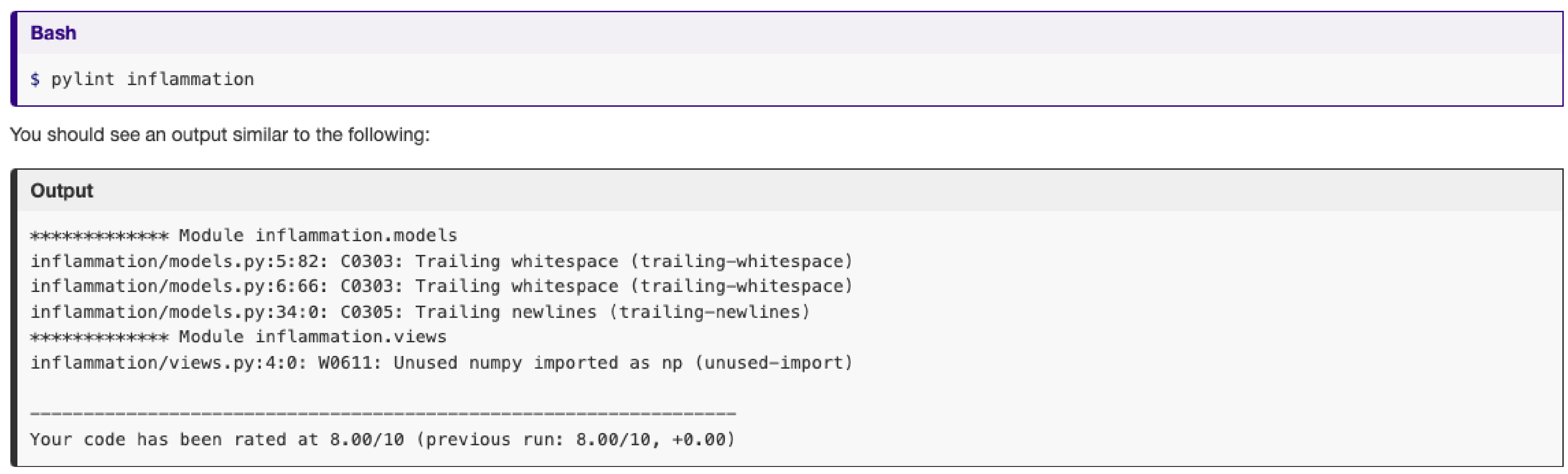


```python
def calculateSUM(NUM1,num2):
    return NUM1+num2
def MAIN():
    NUM1=5
    num2=10
    result=calculateSUM(NUM1,num2)
    print("The sum is:" + str(result))
MAIN()
```



```python
def calculate_sum(a, b):
    return a + b

def main():
    num1 = 5
    num2 = 10
    result = calculate_sum(num1, num2)
    print("The sum is: " + str(result))

main()
```

Verifying Code Style Using Linters



Bash

```
$ pylint inflammation
```

You should see an output similar to the following:

Output

```
************* Module inflammation.models
inflammation/models.py:5:82: C0303: Trailing whitespace (trailing-whitespace)
inflammation/models.py:6:66: C0303: Trailing whitespace (trailing-whitespace)
inflammation/models.py:34:0: C0305: Trailing newlines (trailing-newlines)
************* Module inflammation.views
inflammation/views.py:4:0: W0611: Unused numpy imported as np (unused-import)

-------------------------------------------------------------------
Your code has been rated at 8.00/10 (previous run: 8.00/10, +0.00)
```

Section 2: Ensuring Correctness of Software at Scale

- Introduction to software testing and unit testing
  - An example dataset and application to test for correctness
  - Writing and running tests using the Pytest unit testing framework
  - Using parameterisation to reuse unit tests with different test data

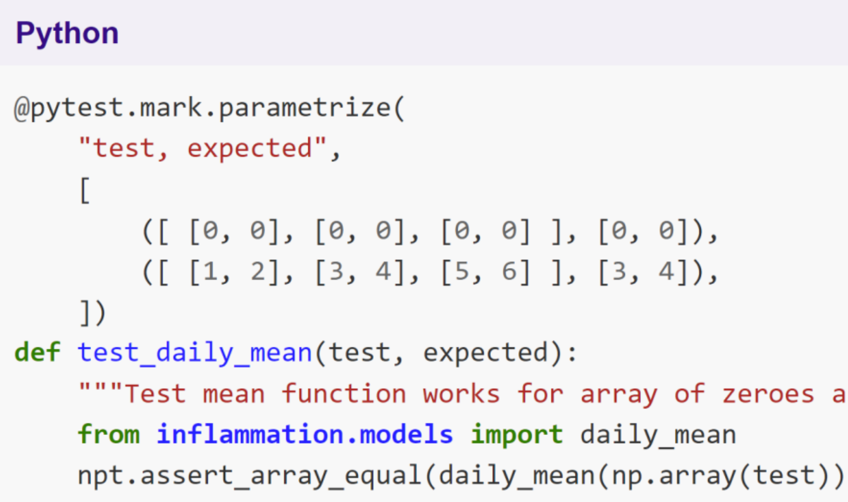
- Introduction to Continuous Integration (CI)
  - Using GitHub Actions to automate running of unit tests
  - Build matrices for cross-platform testing using CI
- Debugging issues and improving robustness
  - Using PyCharm's debugger to diagnose and robustly fix a problem

As researchers we need to be able to verify and demonstrate that our code produces correct results, and unit testing - when done in the right way - can help, as well as save us development time in the long run

Automatically Testing Software

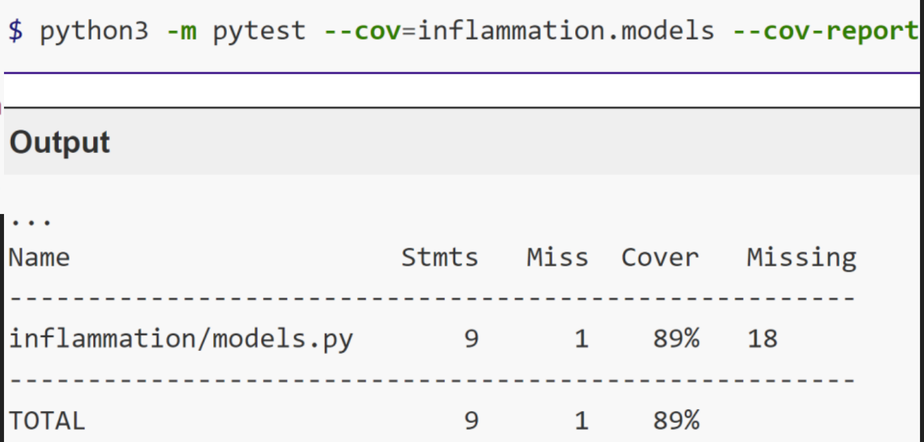
```
 mf2  NLeSC-S
=================                  ==================
platform linux --
rootdir: /home/sa
plugins: datadir-
collected 58 item

tests/multi_fidel                                [ 13%]
tests/property_te                                [ 70%]
tests/regression_                                [100%]
```

Scaling Up Unit Testing



Python

```python
@pytest.mark.parametrize(
    "test, expected",
    [
        ([ [0, 0], [0, 0], [0, 0] ], [0, 0]),
        ([ [1, 2], [3, 4], [5, 6] ], [3, 4]),
    ])
def test_daily_mean(test, expected):
    """Test mean function works for array of zeroes a
    from inflammation.models import daily_mean
    npt.assert_array_equal(daily_mean(np.array(test))
```

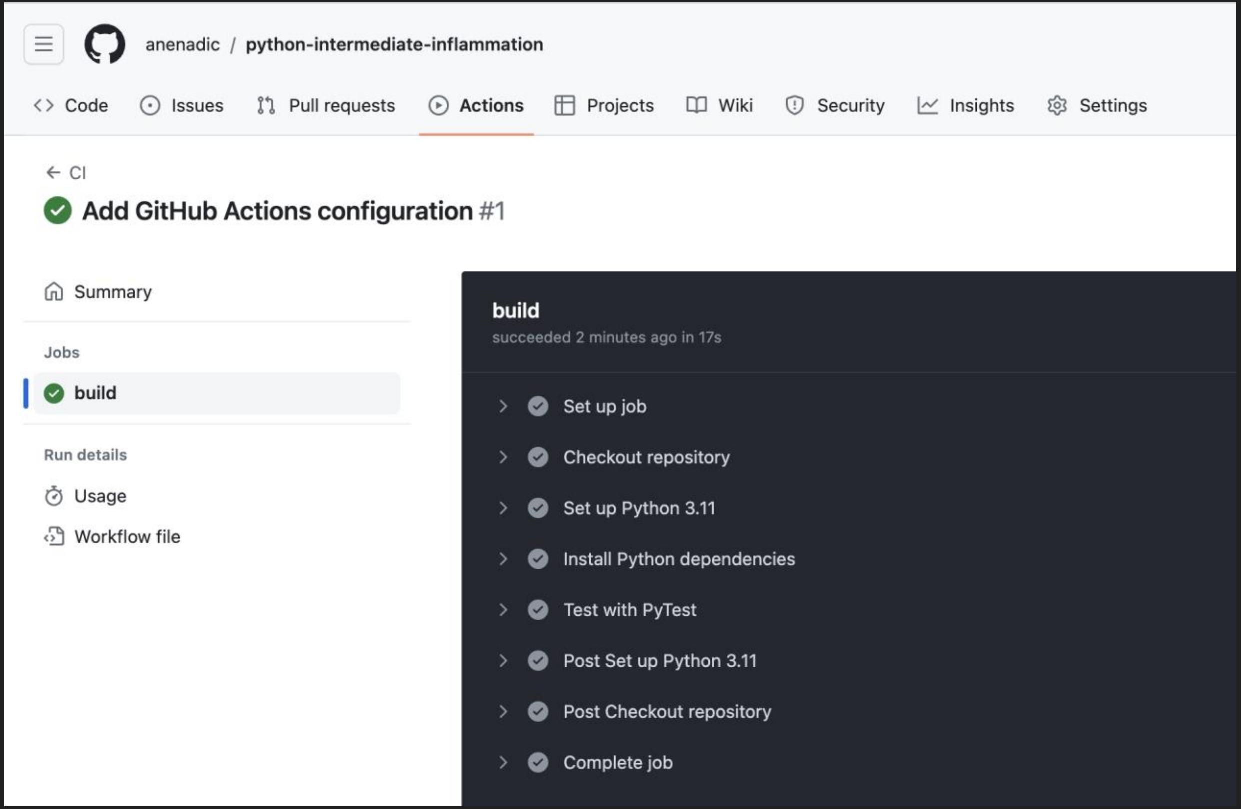


```
$ python3 -m pytest --cov=inflammation.models --cov-report
```

Output

```
...
Name                        Stmts   Miss  Cover   Missing
---------------------------------------------------------
inflammation/models.py          9      1    89%   18
---------------------------------------------------------
TOTAL                           9      1    89%
```

Continuous Integration for Automated Testing



anenadic / python-intermediate-inflammation

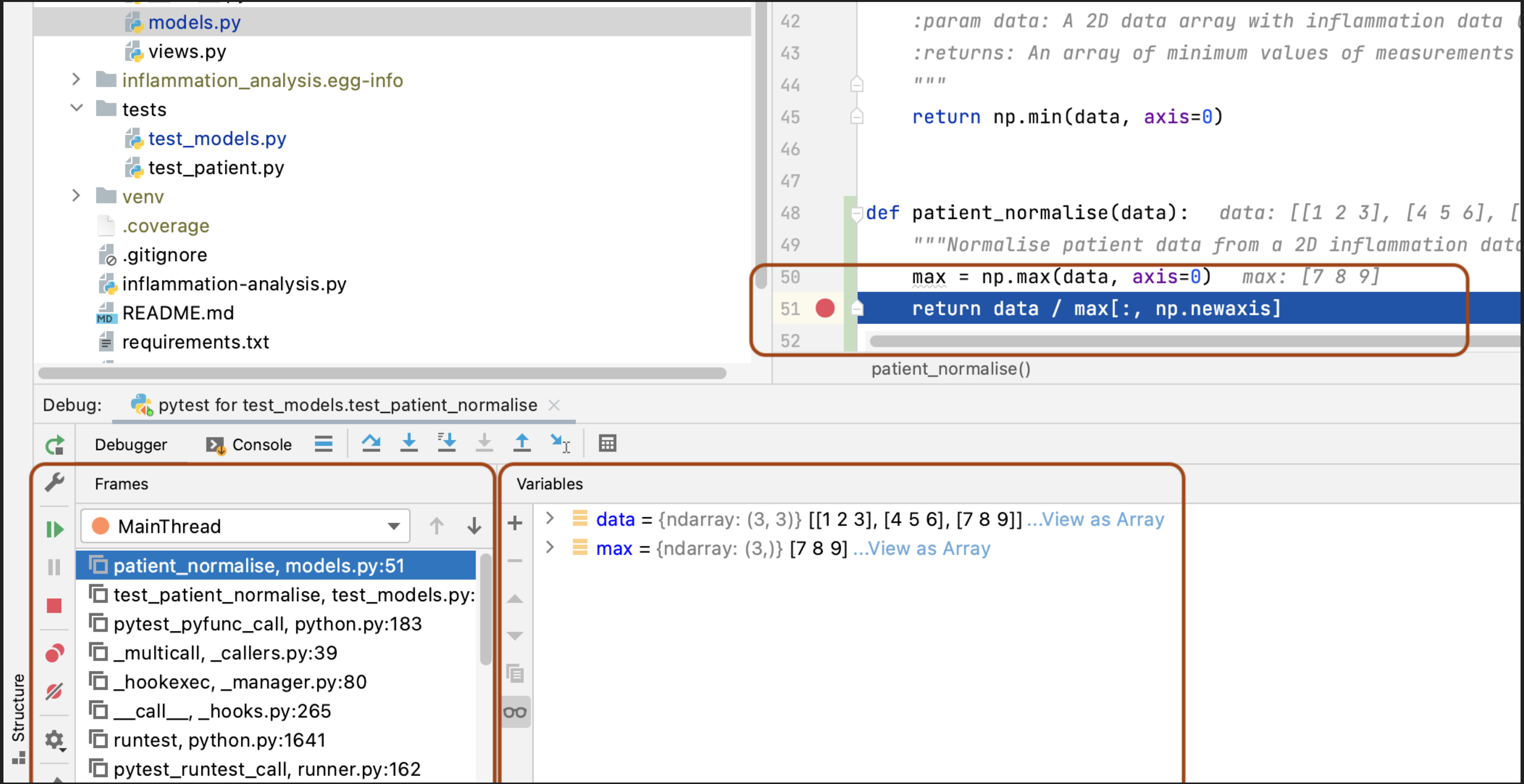
Add GitHub Actions configuration #1

build

succeeded 2 minutes ago in 17s

- Set up job
- Checkout repository
- Set up Python 3.11
- Install Python dependencies
- Test with PyTest
- Post Set up Python 3.11
- Post Checkout repository
- Complete job

Diagnosing Issues and Improving Robustness



```
    :param data: A 2D data array with inflammation data
    :returns: An array of minimum values of measurements
    """
    return np.min(data, axis=0)


def patient_normalise(data):   data: [[1 2 3], [4 5 6], [
    """Normalise patient data from a 2D inflammation dat
    max = np.max(data, axis=0)   max: [7 8 9]
    return data / max[:, np.newaxis]
```

```
patient_normalise, models.py:51
test_patient_normalise, test_models.py:
pytest_pyfunc_call, python.py:183
_multicall, _callers.py:39
_hookexec, _manager.py:80
__call__, _hooks.py:265
runtest, python.py:1641
pytest_runtest_call, runner.py:162
```

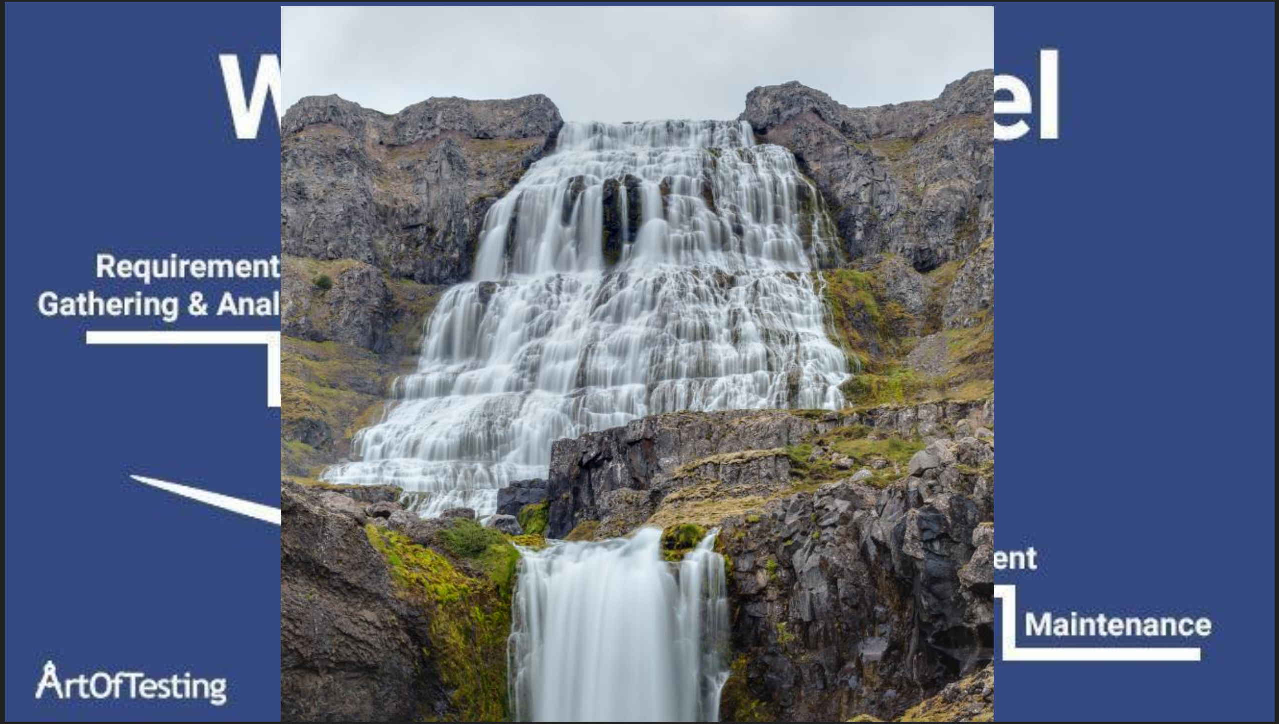
```
data = {ndarray: (3, 3)} [[1 2 3], [4 5 6], [7 8 9]] ...View as Array
max = {ndarray: (3,)} [7 8 9] ...View as Array
```

Section 3: Software Development as a Process

- **Programming vs software engineering**
- Software requirements, design, architecture and implementation (via programming paradigms) as typical stages of a software development process
    - Design requirements and constraints - what affects our design choices?
    - Layered architecture
    - Programming paradigms
        - Procedural, Functional, Object Oriented

Software designed deliberately will be much easier to work with in the long term than software we build without a plan - time spent planning in advance will save you a lot more time later

Software development process



Software requirements

Without clear requirements you will find out later that you built software that no one actually needed.

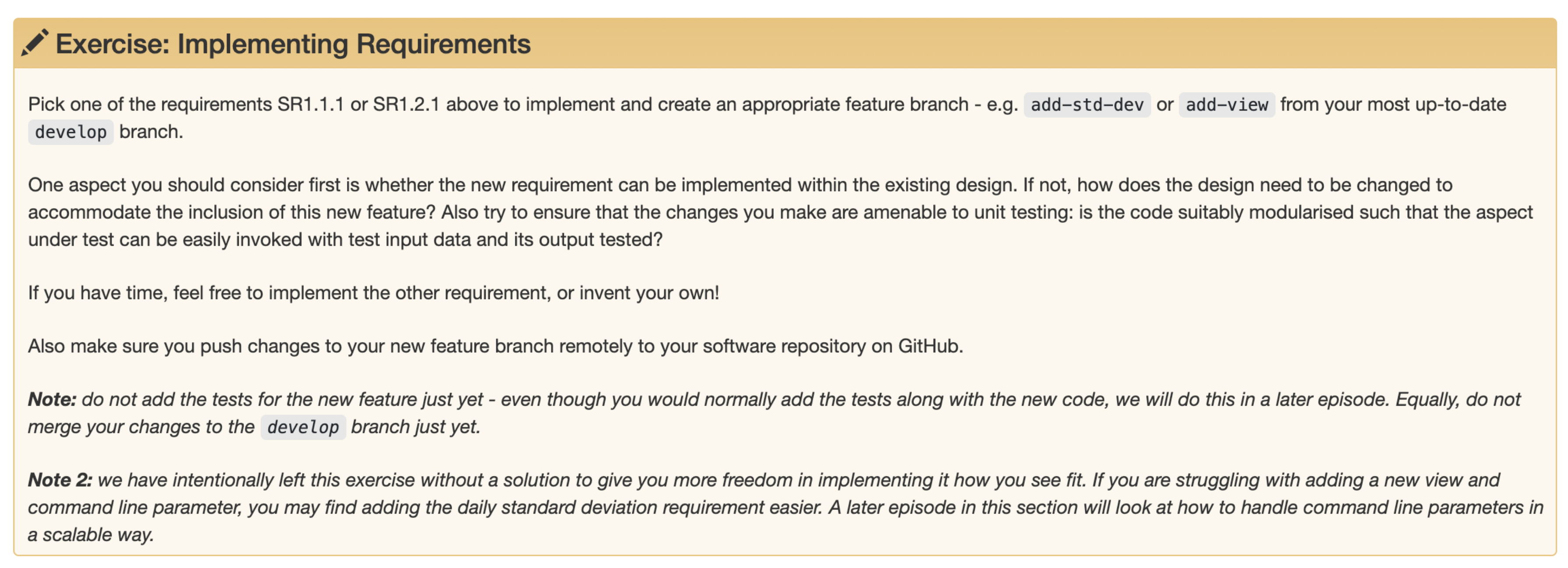
Requirements will constantly change!

Software requirements

Without clear requirements you will find out later that you built software that no one actually needed.

Requirements will almost always change!

Warning: Explicitly describing requirements can feel like a lot of overhead.



Exercise: Implementing Requirements

Pick one of the requirements SR1.1.1 or SR1.2.1 above to implement and create an appropriate feature branch - e.g. `add-std-dev` or `add-view` from your most up-to-date `develop` branch.

One aspect you should consider first is whether the new requirement can be implemented within the existing design. If not, how does the design need to be changed to accommodate the inclusion of this new feature? Also try to ensure that the changes you make are amenable to unit testing: is the code suitably modularised such that the aspect under test can be easily invoked with test input data and its output tested?

If you have time, feel free to implement the other requirement, or invent your own!

Also make sure you push changes to your new feature branch remotely to your software repository on GitHub.

Note: *do not add the tests for the new feature just yet - even though you would normally add the tests along with the new code, we will do this in a later episode. Equally, do not merge your changes to the* `develop` *branch just yet.*

Note 2: *we have intentionally left this exercise without a solution to give you more freedom in implementing it how you see fit. If you are struggling with adding a new view and command line parameter, you may find adding the daily standard deviation requirement easier. A later episode in this section will look at how to handle command line parameters in a scalable way.*

Read the notes!

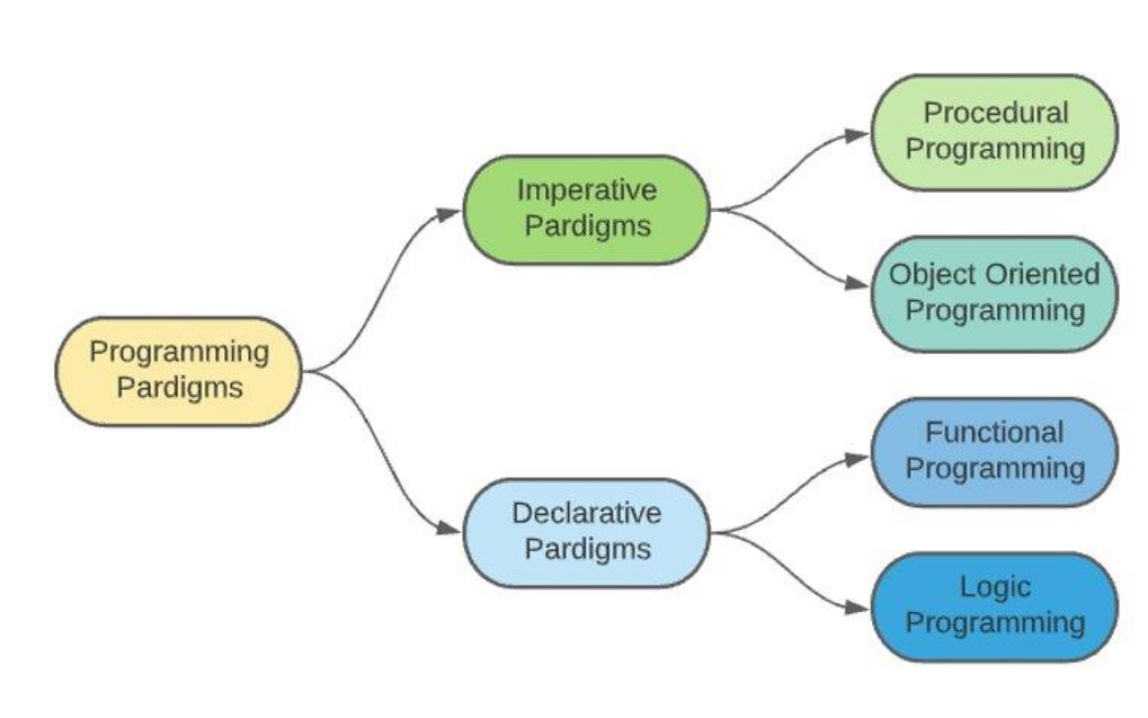
Software design

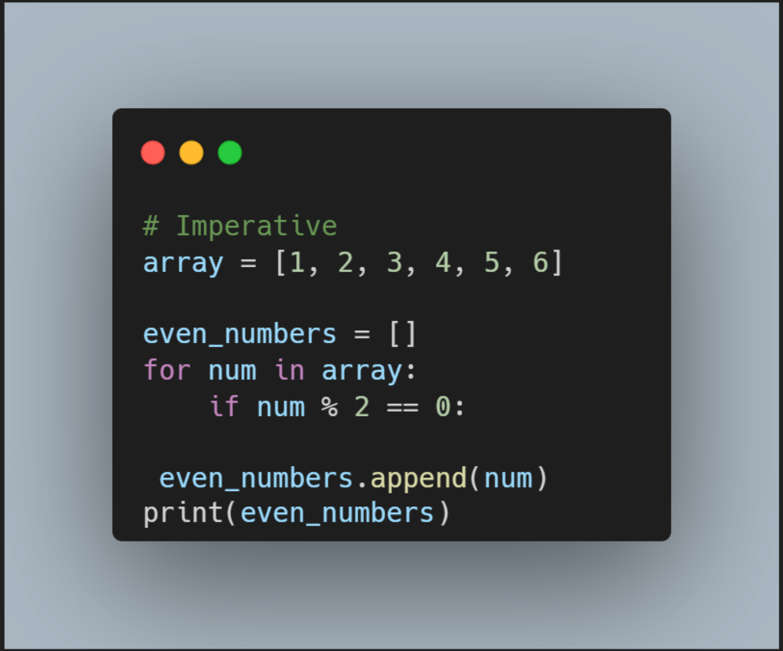
- Under-design
- Technical debt

- Over-design
- Overhead

It is your job as engineer to find a balance in the tools that you will learn.

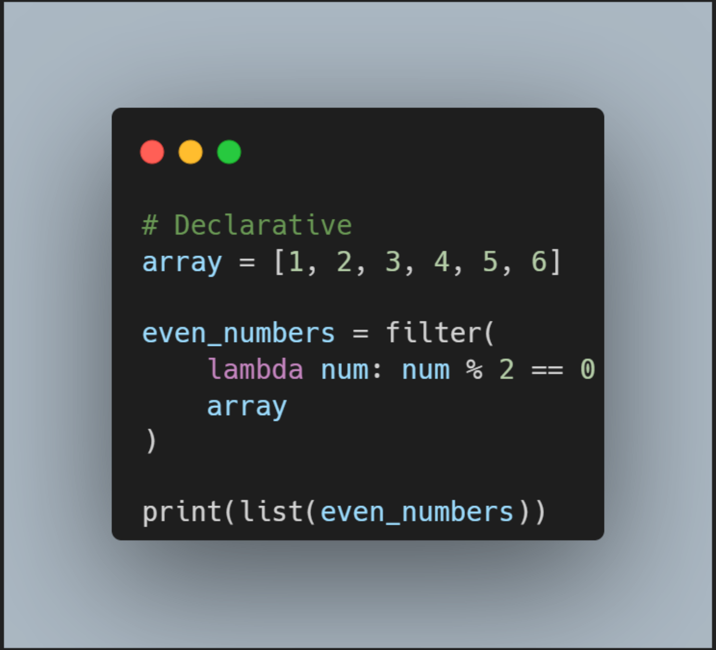
Programming paradigms





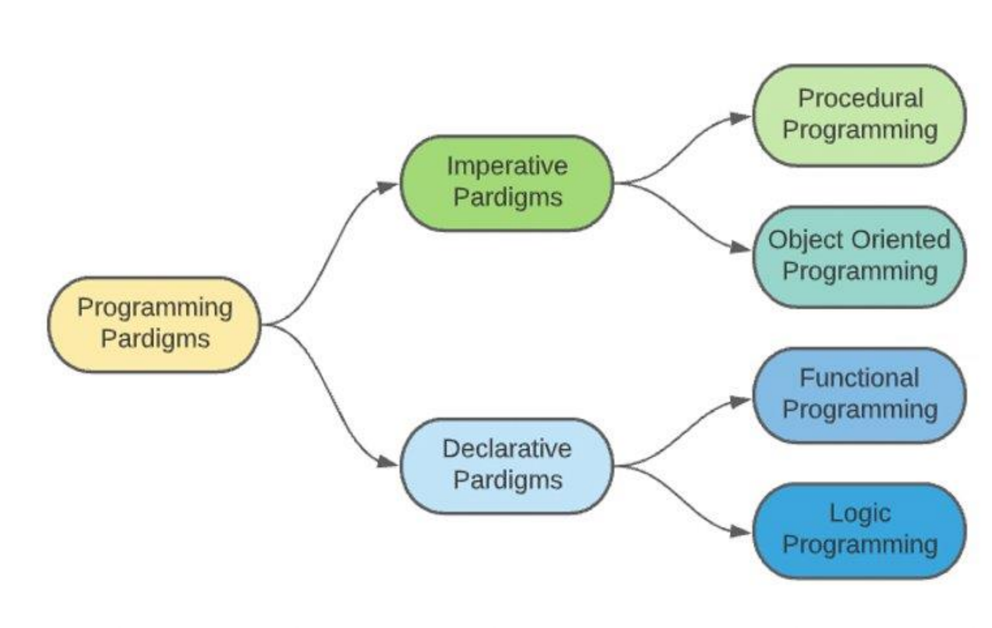
```
# Imperative
array = [1, 2, 3, 4, 5, 6]

even_numbers = []
for num in array:
    if num % 2 == 0:

 even_numbers.append(num)
print(even_numbers)
```

How to do things

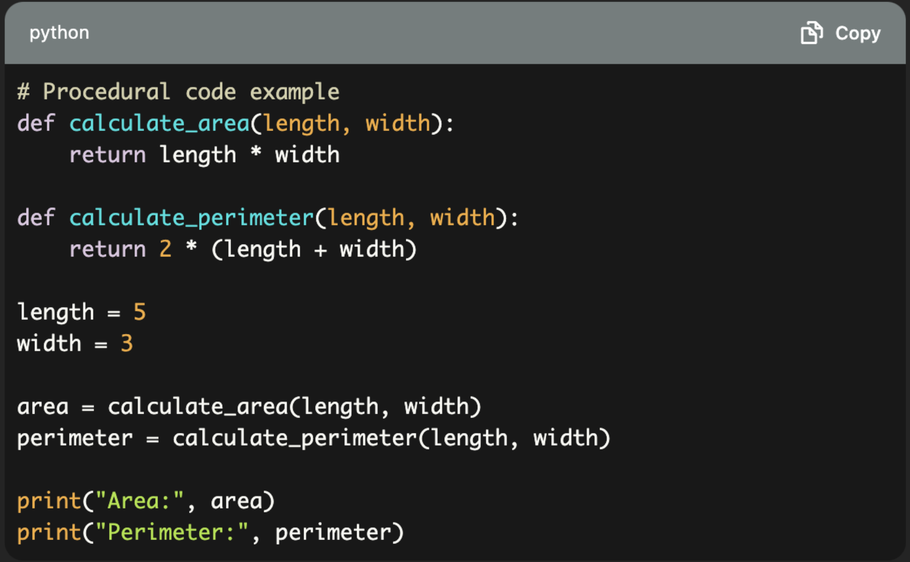


```
# Declarative
array = [1, 2, 3, 4, 5, 6]

even_numbers = filter(
    lambda num: num % 2 == 0
    array
)

print(list(even_numbers))
```

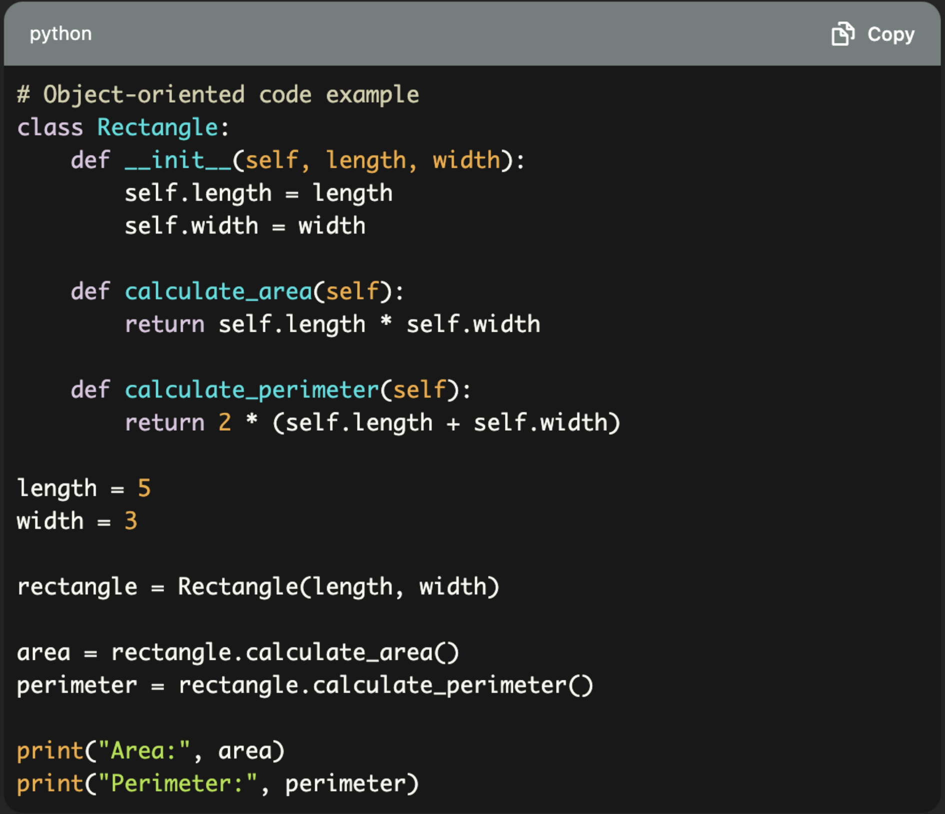
What to do


Programming Pardigms
Imperative Pardigms
Declarative Pardigms
Procedural Programming
Object Oriented Programming
Functional Programming
Logic Programming

Procedural versus object-oriented programming



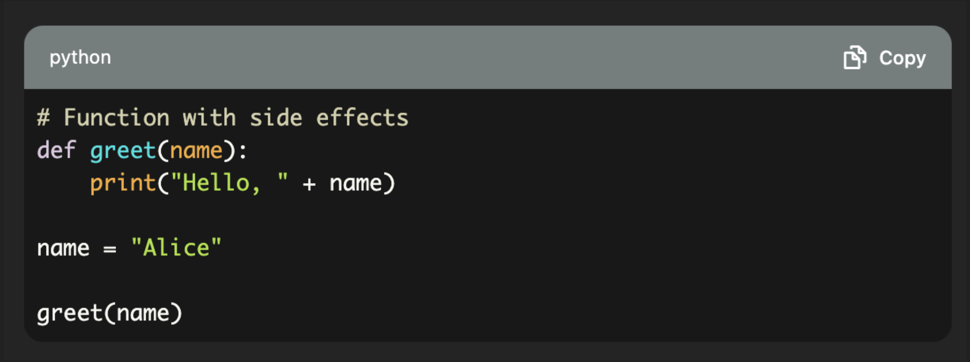
```python
# Procedural code example
def calculate_area(length, width):
    return length * width

def calculate_perimeter(length, width):
    return 2 * (length + width)

length = 5
width = 3

area = calculate_area(length, width)
perimeter = calculate_perimeter(length, width)

print("Area:", area)
print("Perimeter:", perimeter)
```



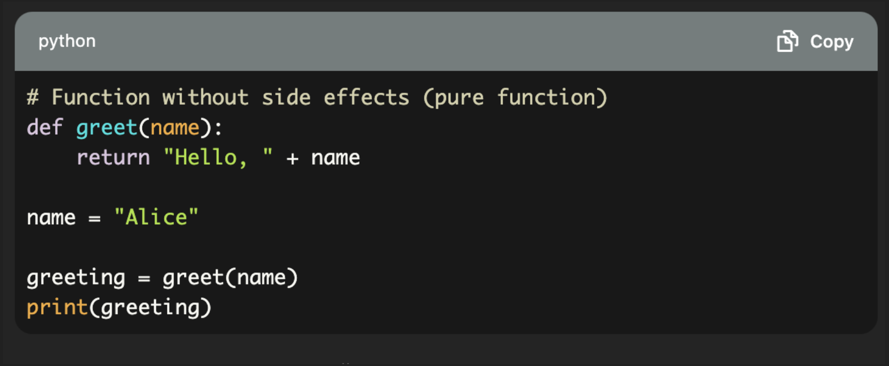
```python
# Object-oriented code example
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

    def calculate_perimeter(self):
        return 2 * (self.length + self.width)

length = 5
width = 3

rectangle = Rectangle(length, width)

area = rectangle.calculate_area()
perimeter = rectangle.calculate_perimeter()

print("Area:", area)
print("Perimeter:", perimeter)
```

Functional programming

- Functions are first-class citizens
- Pure functions. No side effects!



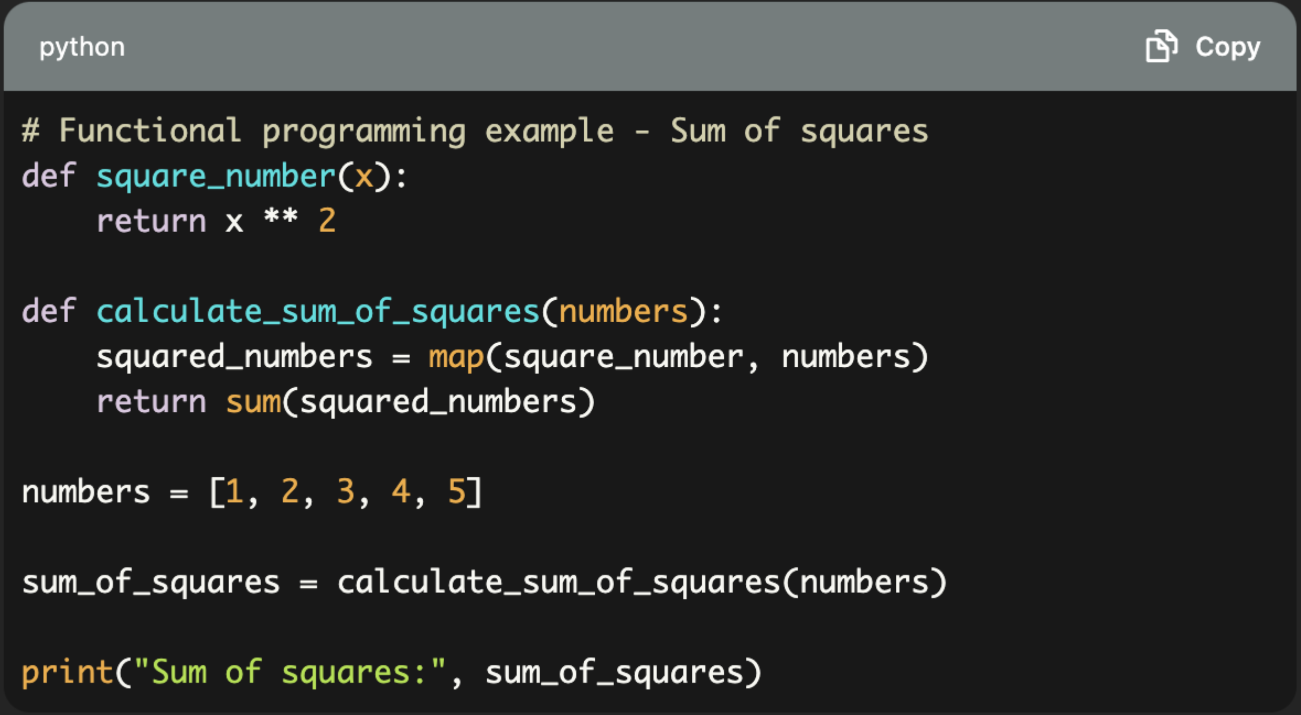
```python
# Function with side effects
def greet(name):
    print("Hello, " + name)

name = "Alice"

greet(name)
```



```python
# Function without side effects (pure function)
def greet(name):
    return "Hello, " + name

name = "Alice"

greeting = greet(name)
print(greeting)
```

A typical functional programming example

When would you use this paradigm?



```python
# Functional programming example - Sum of squares
def square_number(x):
    return x ** 2

def calculate_sum_of_squares(numbers):
    squared_numbers = map(square_number, numbers)
    return sum(squared_numbers)

numbers = [1, 2, 3, 4, 5]

sum_of_squares = calculate_sum_of_squares(numbers)

print("Sum of squares:", sum_of_squares)
```

Use the right tool for the right job

tinyurl.com/2024-04-15-ds-int-py-day2

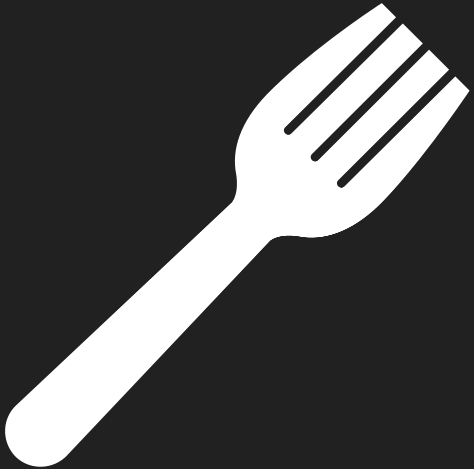
tinyurl.com/2024-04-15-ds-int-py-day2

Section 4: Collaborative Software Development for Reuse

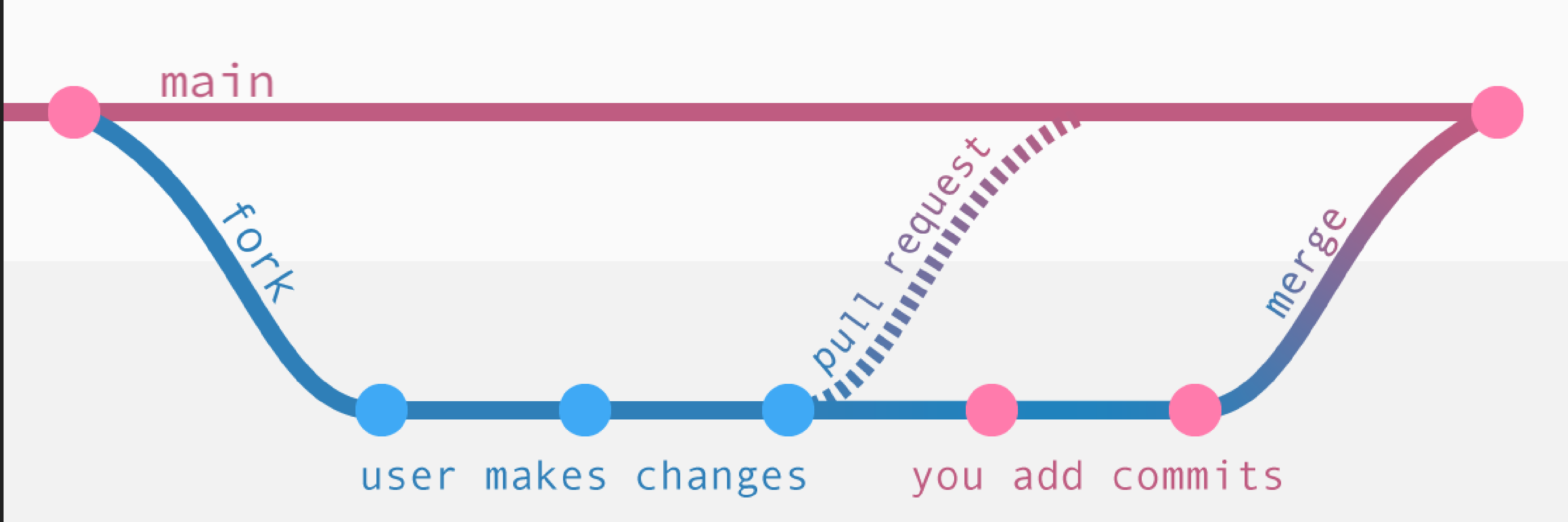
- Practices for developing software collaboratively that will make it easier for us and others to further develop and reuse
- Code review
  - Improving the quality of code
  - Detecting bugs and code defects early
  - Increasing knowledge of the software within the team
- Preparing software for reuse
  - Levels of reusability
  - Documenting code for reuse
  - Choosing an open source license
  - Packaging software for release and reuse

When changes, particularly big changes, are made to a codebase, how can a team ensure that these changes are well considered and represent good solutions?

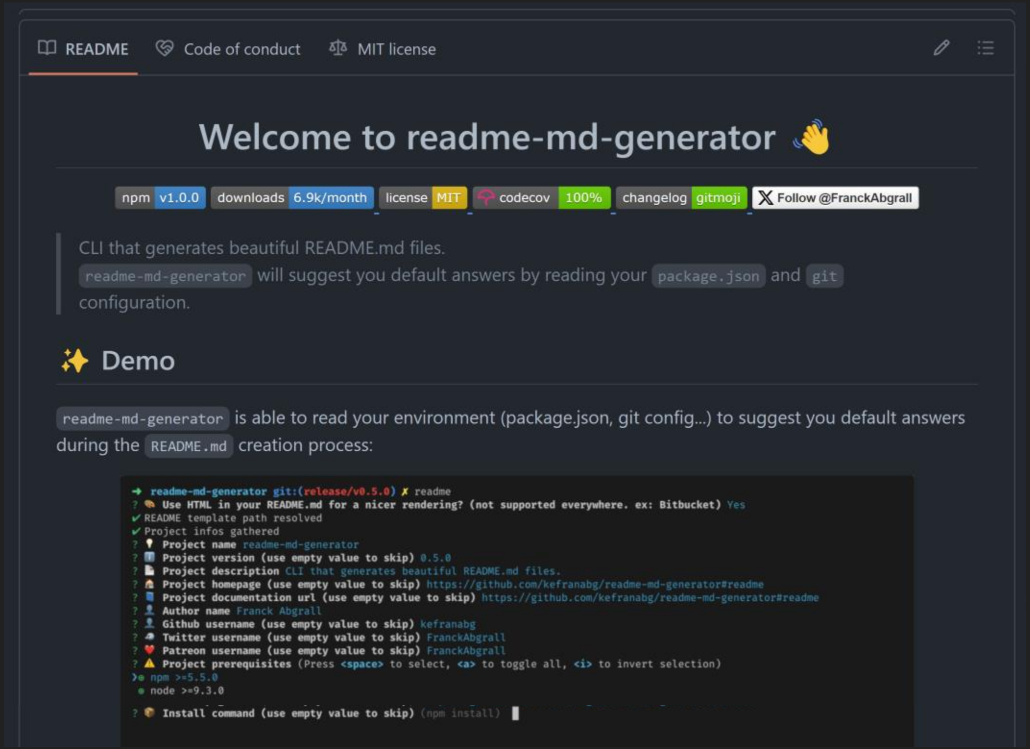
Developing Software in a Team: Code Review



Developing Software in a Team: Code Review



Preparing Software for Reuse and Release



Welcome to readme-md-generator 👋

> CLI that generates beautiful README.md files.
> `readme-md-generator` will suggest you default answers by reading your `package.json` and `git` configuration.

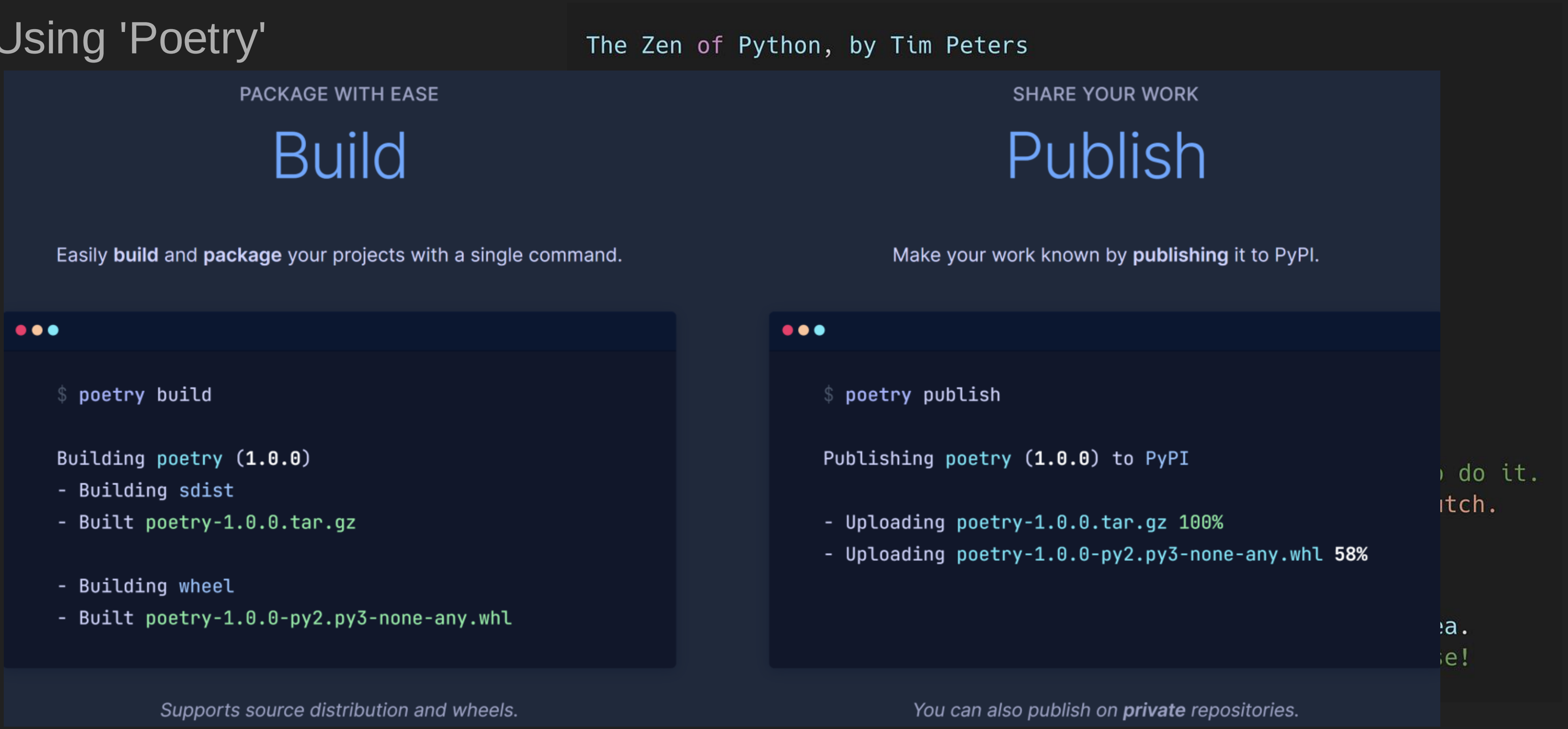
✨ Demo

`readme-md-generator` is able to read your environment (package.json, git config...) to suggest you default answers during the `README.md` creation process:

```
➜ readme-md-generator git:(release/v0.5.0) ✗ readme
? Use HTML in your README.md for a nicer rendering? (not supported everywhere. ex: Bitbucket) Yes
✔ README template path resolved
✔ Project infos gathered
? Project name readme-md-generator
? Project version (use empty value to skip) 0.5.0
? Project description CLI that generates beautiful README.md files.
? Project homepage (use empty value to skip) https://github.com/kefranabg/readme-md-generator#readme
? Project documentation url (use empty value to skip) https://github.com/kefranabg/readme-md-generator#readme
? Author name Franck Abgrall
? Github username (use empty value to skip) kefranabg
? Twitter username (use empty value to skip) FranckAbgrall
? Patreon username (use empty value to skip) FranckAbgrall
? Project prerequisites (Press <space> to select, <a> to toggle all, <i> to invert selection)
❯◉ npm >=5.5.0
 ◉ node >=9.3.0

? Install command (use empty value to skip) (npm install) █
```

Packaging Software for Release and Distribution

- Using 'Poetry'



```
The Zen of Python, by Tim Peters
```

```
do it.
tch.
a.
e!
```

PACKAGE WITH EASE

Build

Easily **build** and **package** your projects with a single command.

```
$ poetry build

Building poetry (1.0.0)
- Building sdist
- Built poetry-1.0.0.tar.gz

- Building wheel
- Built poetry-1.0.0-py2.py3-none-any.whl
```

Supports source distribution and wheels.

SHARE YOUR WORK

Publish

Make your work known by **publishing** it to PyPI.

```
$ poetry publish

Publishing poetry (1.0.0) to PyPI

- Uploading poetry-1.0.0.tar.gz 100%
- Uploading poetry-1.0.0-py2.py3-none-any.whl 58%
```

You can also publish on ***private*** *repositories.*

tinyurl.com/2024-04-15-ds-int-py-day2

Section 5: Managing and Improving Software

- **Software not only needs to be developed, but also managed**
- Collaborative platforms are a means of communication and documentation
- Estimating the required effort for work items is useful to set priorities

tinyurl.com/2024-04-15-ds-int-py-day3

Managing a Collaborative Software Project

- Platforms like GitHub provide tools to manage software

- Manage bug reports, feature requests and future work with issues
  - Use labels to categorize issues: "bug", "enhancement", "help wanted", ..
  - Example

- Notifications & referencing make it easy to reference
  - Collaborators with mentions: @f-hafner
  - Issues, pull requests and comments: #14 Some issue

- Manage software projects with project boards
  - Overview of open issues, current work, planned work, etc. Example

Assessing Software for Suitability and Improvement

- This is about judging other people's software...
  - "Should we include package X as a dependency in our software?"

- … but also important for developing our own software
  - Provide contact information
  - Manage support, for instance with an issue tracker
  - Manage expectations: which platforms? What kind of support? How long?

- Software management plans
  - 10.5281/zenodo.7038280

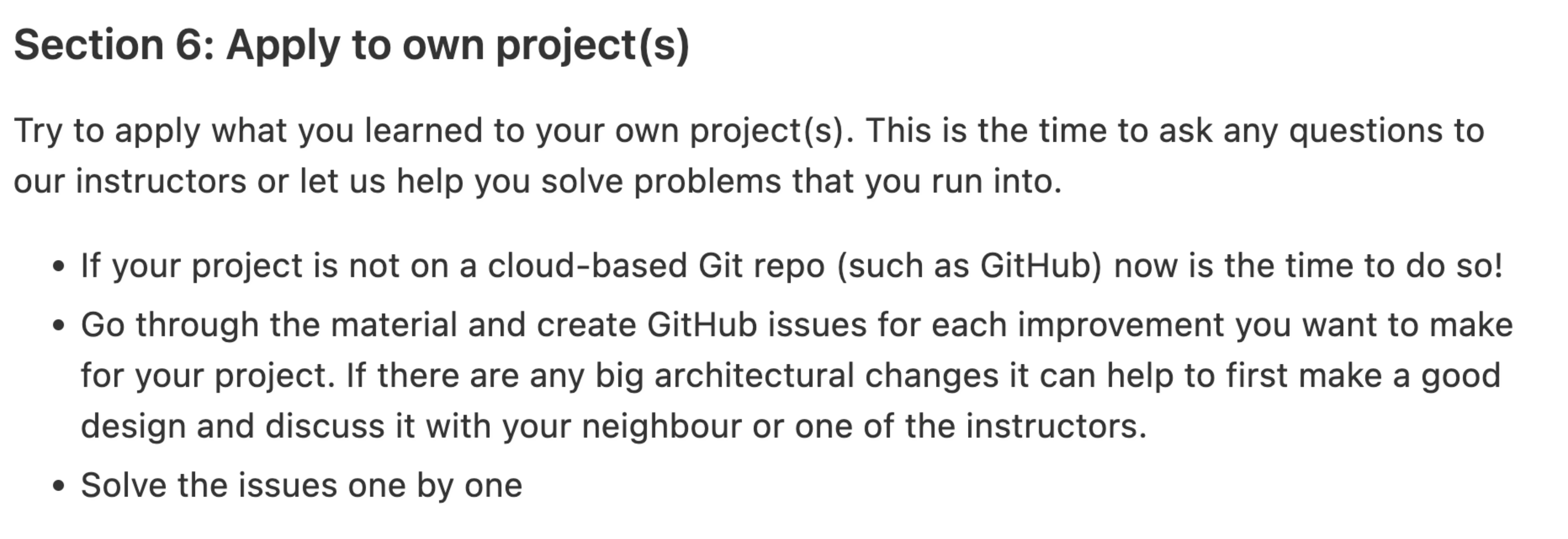
Software Improvement Through Feedback

1. Estimate time
    - How much time do we have to resolve requirements (=issues)?
    - How much overall effort do we have available?
    - How long does each requirement need to resolve?
    - -> done by people doing the actual work; keep learning; do it as a team

2. Prioritize our work
    - Collaborators, researchers, Principle Investigators are all affected by project and may have different interests
    - MoSCoW method: must have, should have, could have, won't have

3. Sprints tie these things together
    - Work on agreed prioritized things for a given period (1w-1mo)
    - This can be a continuous process

Section 6: Apply learned skills to your own projects



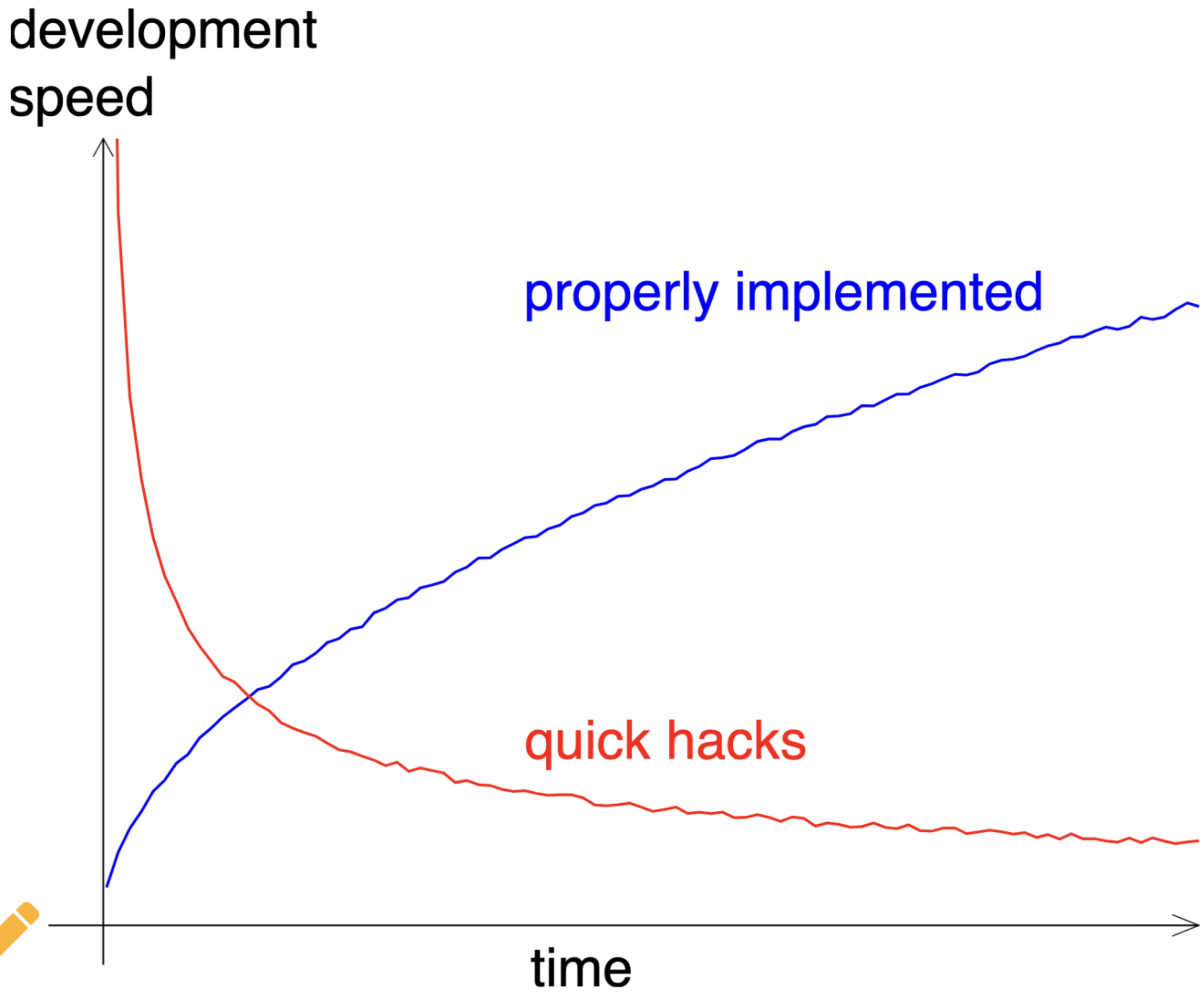
Section 6: Apply to own project(s)

Try to apply what you learned to your own project(s). This is the time to ask any questions to our instructors or let us help you solve problems that you run into.

- If your project is not on a cloud-based Git repo (such as GitHub) now is the time to do so!
- Go through the material and create GitHub issues for each improvement you want to make for your project. If there are any big architectural changes it can help to first make a good design and discuss it with your neighbour or one of the instructors.
- Solve the issues one by one

For software to succeed it needs to be managed as well as developed

Congratulations! You have evolved from a simple programmer to a full-grown software engineer!



- Design
- Requirements
- Automated testing
- Code quality
- Linter
- Continuous Integration
- Packaging
- Documentation
- Peer review
- Project organisation
- Feature branching

We can teach you some useful skills, but unfortunately in software engineering experience matters…

Ferdi Rizkiyanto

Next steps

- Apply what you learned in your own projects.
    - Measure code coverage and code quality, slowly build this up over time.
- Self-learning (see collaborative document day 3 for resources)
    - Read a book
    - Watch youtube
    - Online courses and tutorials

What did you learn that is maybe more important than experience?

What did you learn that is maybe more important than experience:

ATTITUDE