UNIVERSITY OF LEEDS

Workshop: Research coding

2022-06-21, SHAPE-IT meeting in Leeds

Gustav Markkula

Aims

UNIVERSITY OF LEEDS

- Consider possible benefits of getting better at coding (regardless of current level)
- Be aware of some good (and less good) coding practices
- Get some ideas on how to work toward better own code

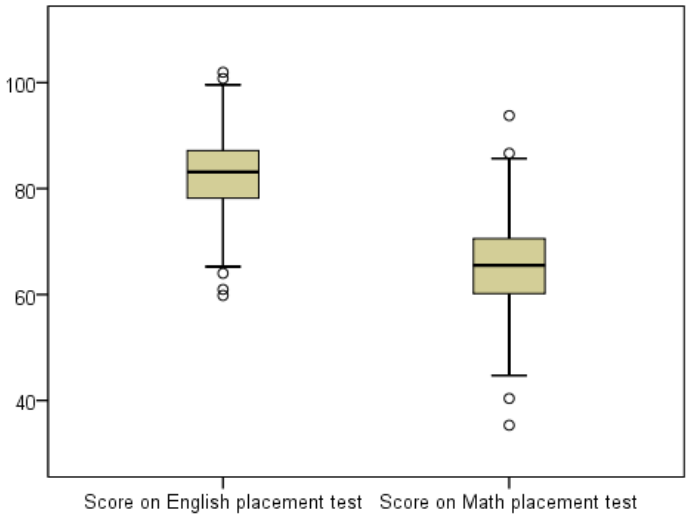
Outline

UNIVERSITY OF LEEDS

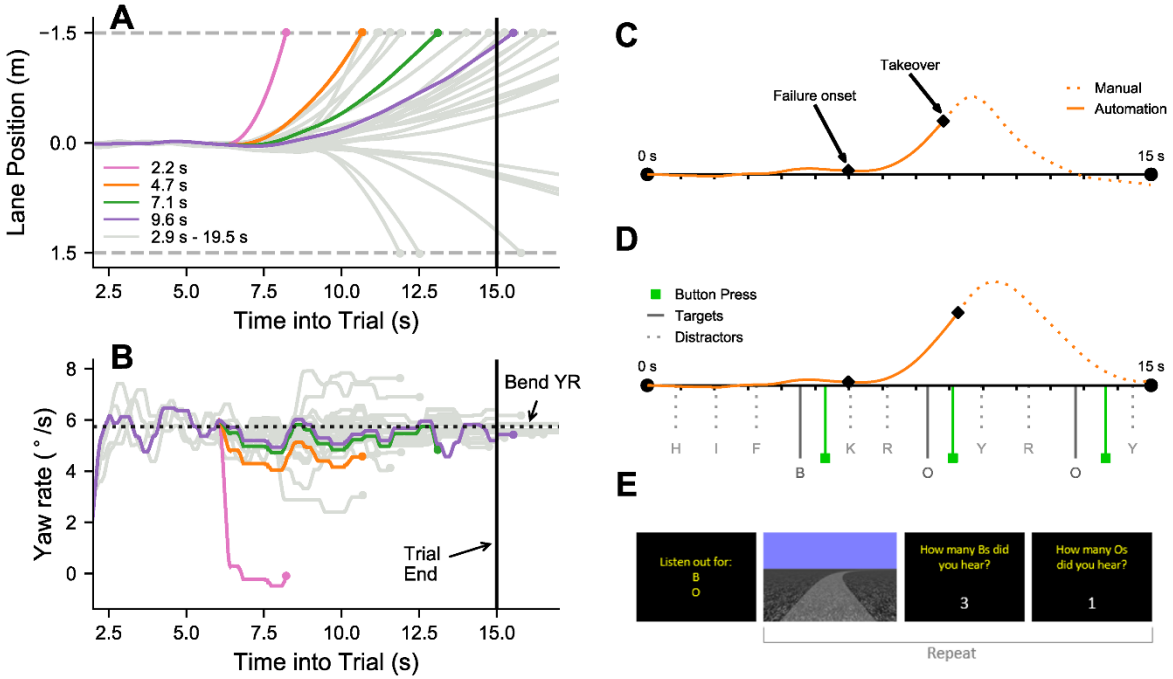
- Lecture – my two cents
  - Coding – why and what and how?
  - Your brain on code
  - Good practices and less good practices
  - … for some of you a lot of this will be familiar
- Code review alone and in pairs
- General discussion

Coding – why?

More powerful and cool/beautiful analyses, models, and figures



(source)



(Mole et al., 2020)

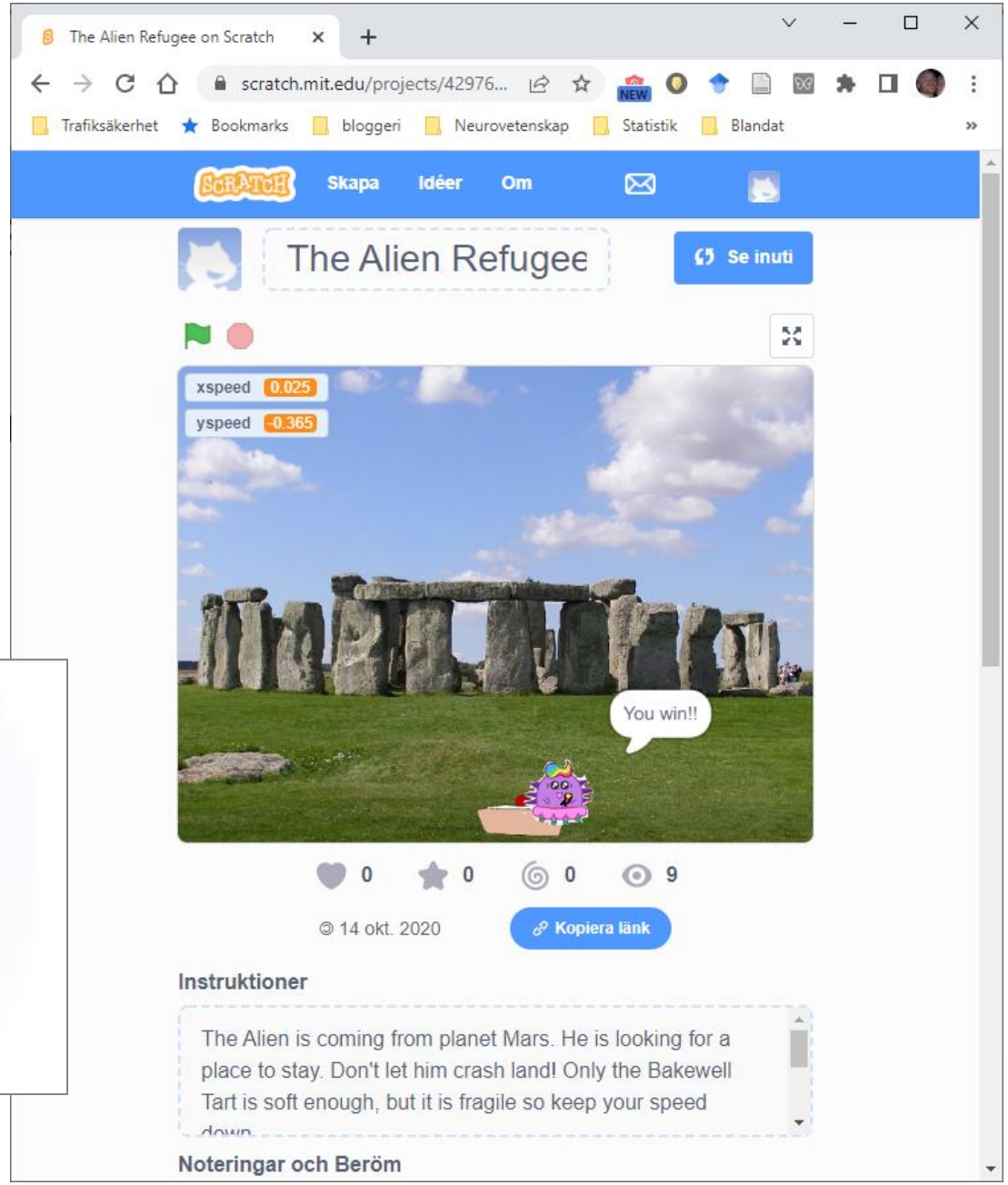
Coding – why?

Fun!

https://scratch.mit.edu/

https://p5js.org/

https://unity.com/



Coding – what?

python™ | JS | HTML | CSS

C++ | TS | R | THE SCHEME PROGRAMMING LANGUAGE

Java | Kotlin | C# | Perl

php | Scala | Swift | MATLAB SIMULINK

Microsoft SQL Server | R | GO | Ruby

Coding – how?

Anyone can learn programming!

Getting **good** at programming takes a lot of time and practice

Coding – how?

… but getting better always happens one step at a time

... and each improvement can make a big difference

If you haven't taken a proper CS/programming course, do consider it, e.g.:

https://www.makeuseof.com/tag/best-free-online-computer-programming-courses/

https://www.edx.org/course/introduction-computer-science-harvardx-cs50x

Your brain on code

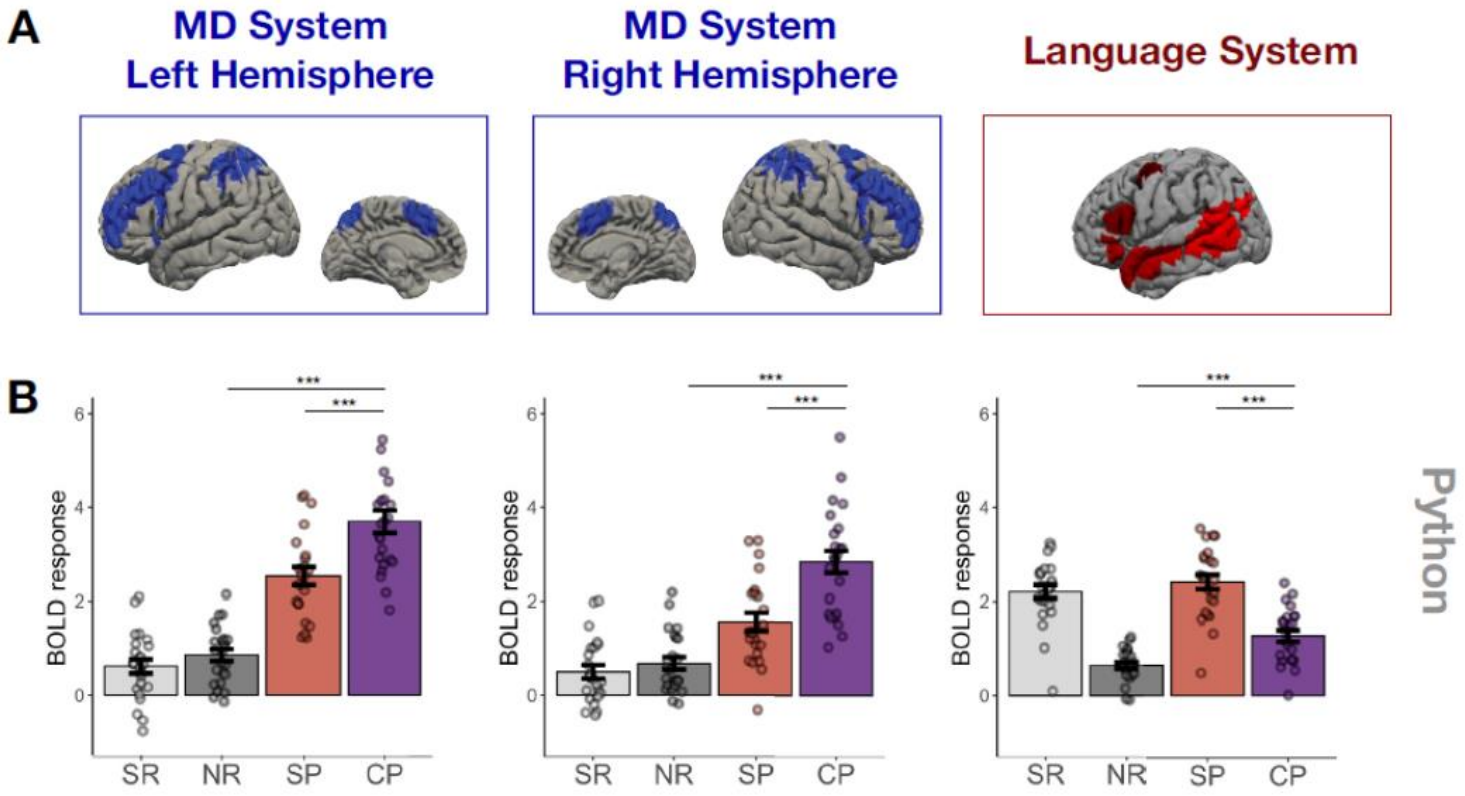
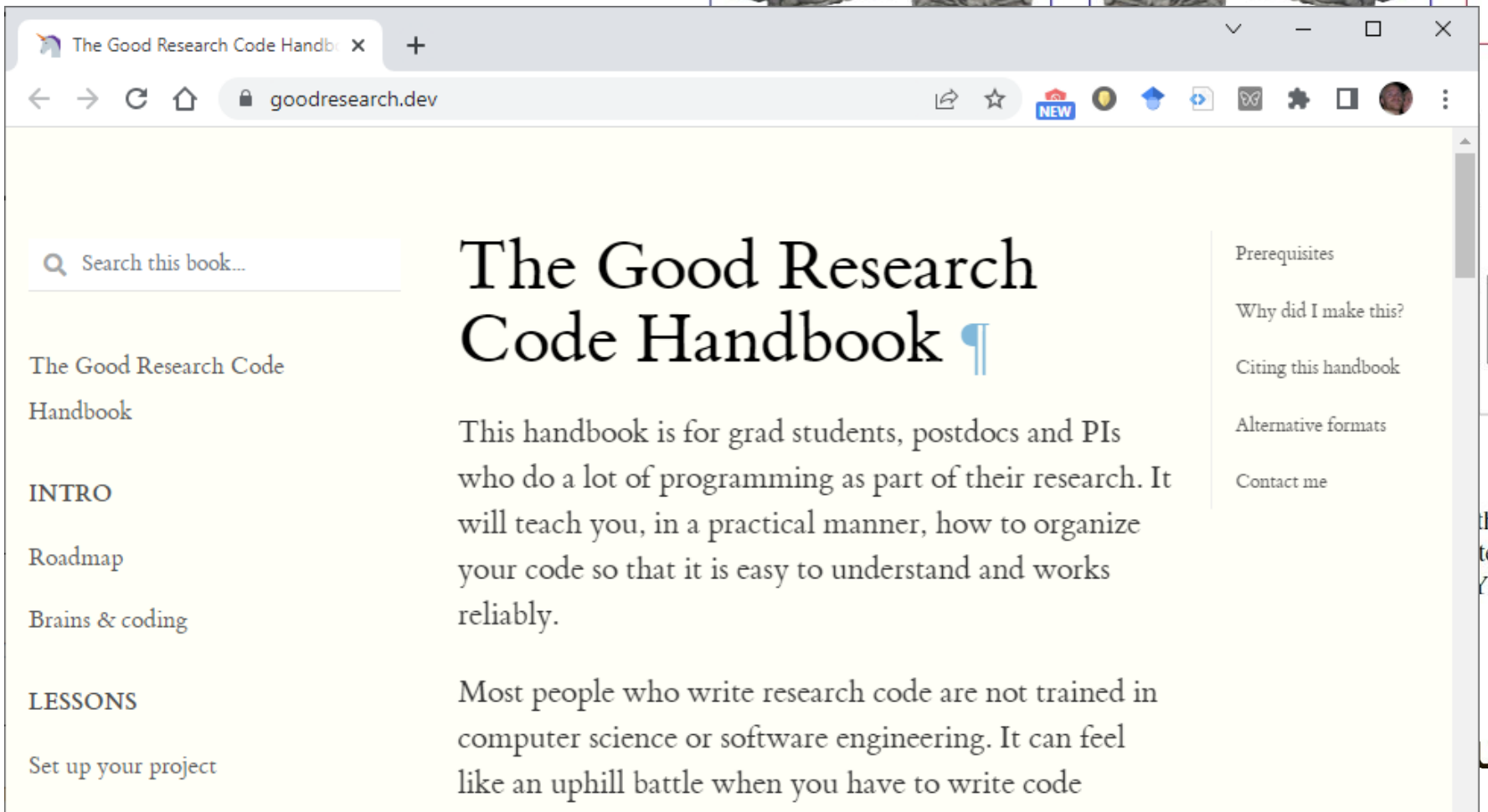
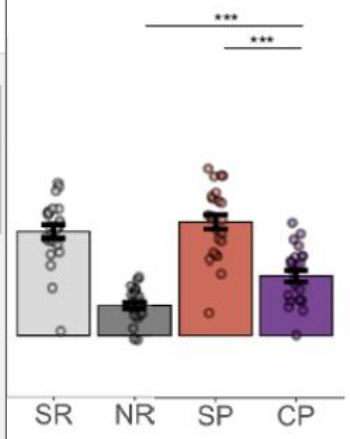


Fig. 1: Code problems (CP; purple bars) created activations with higher overlap with the multiple demand system (left and center) than with the language system (right) compared with other tasks like sentence problems (SP), sentence reading (SR) or non-word reading (NR). From Ivanova et al. (2020), used under a CC-BY 4.0 license.

Your brain on code

Resource tip:
https://goodresearch.dev



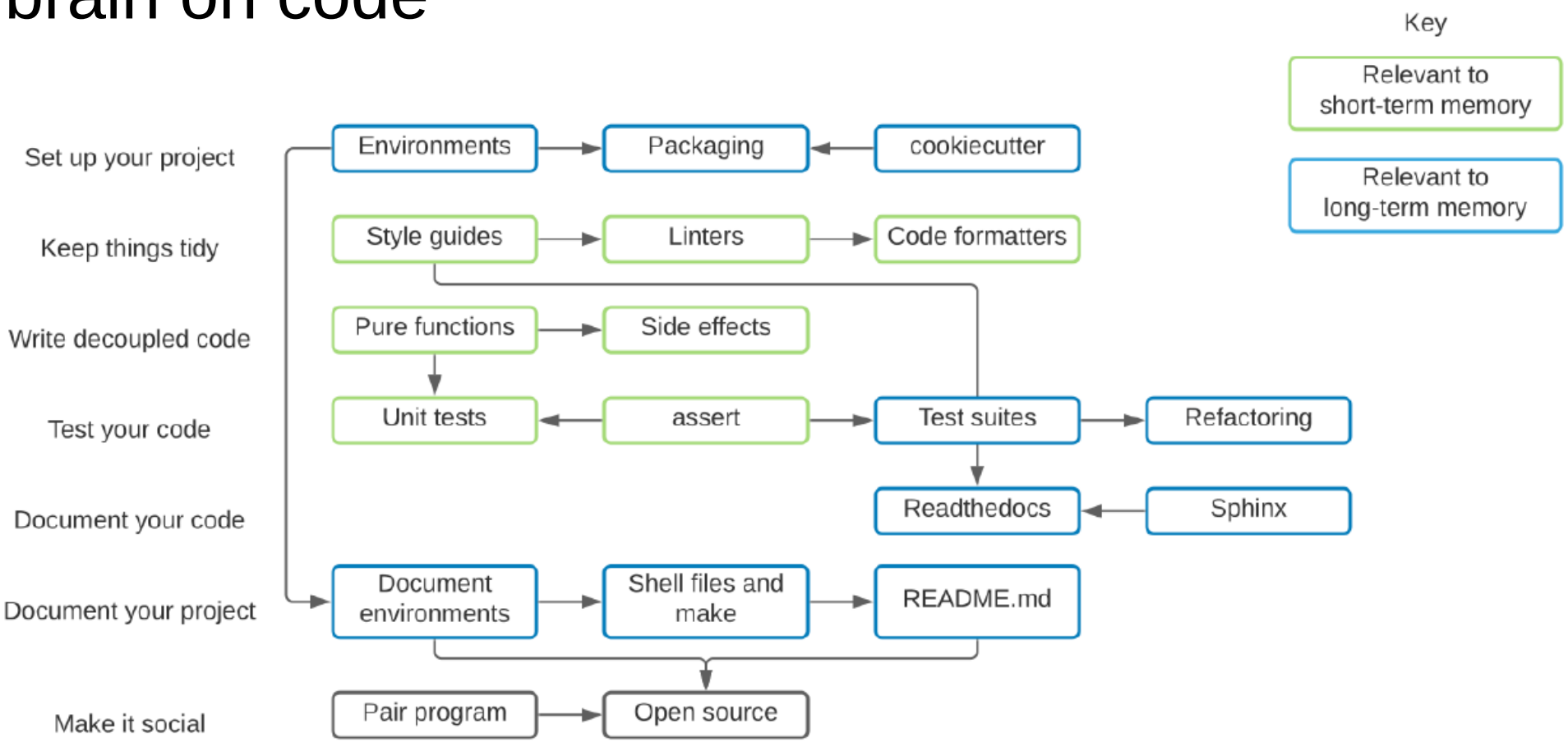


Python

th the multiple demand system (left and
tence problems (SP), sentence reading
Y 4.0 license.

UNIVERSITY OF LEEDS

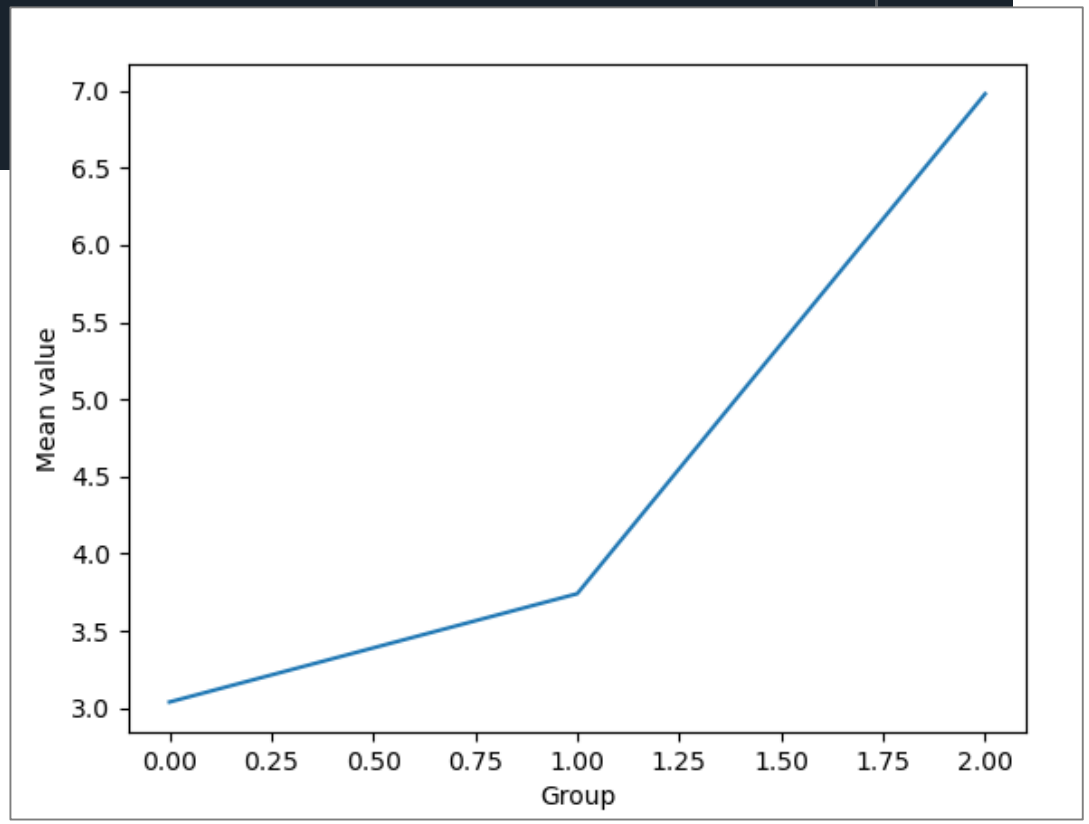
Your brain on code



Time to look at some actual code…

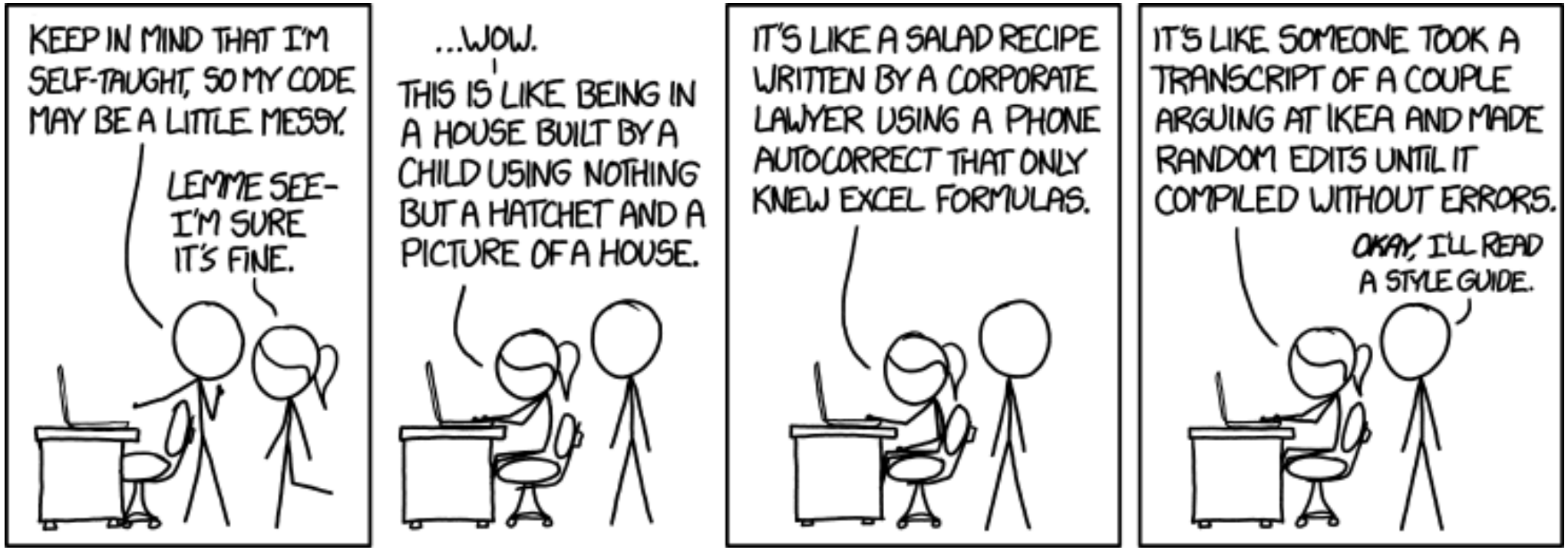
UNIVERSITY OF LEEDS

```python
import numpy as np
import matplotlib.pyplot as plt

ym = np.full(3, np.nan)

ym[0] = np.mean(np.loadtxt('grpdata0.csv', delimiter=', ')[:, 1])
ym[1] = np.mean(np.loadtxt('grpdata1.csv', delimiter=', ')[:, 1])
ym[2] = np.mean(np.loadtxt('grpdata2.csv', delimiter=', ')[:, 1])

plt.plot(ym)
plt.xlabel('Group')
plt.ylabel('Mean value')
plt.show()
```



Code improvement ideas?

Coding style and naming conventions



(xkcd)

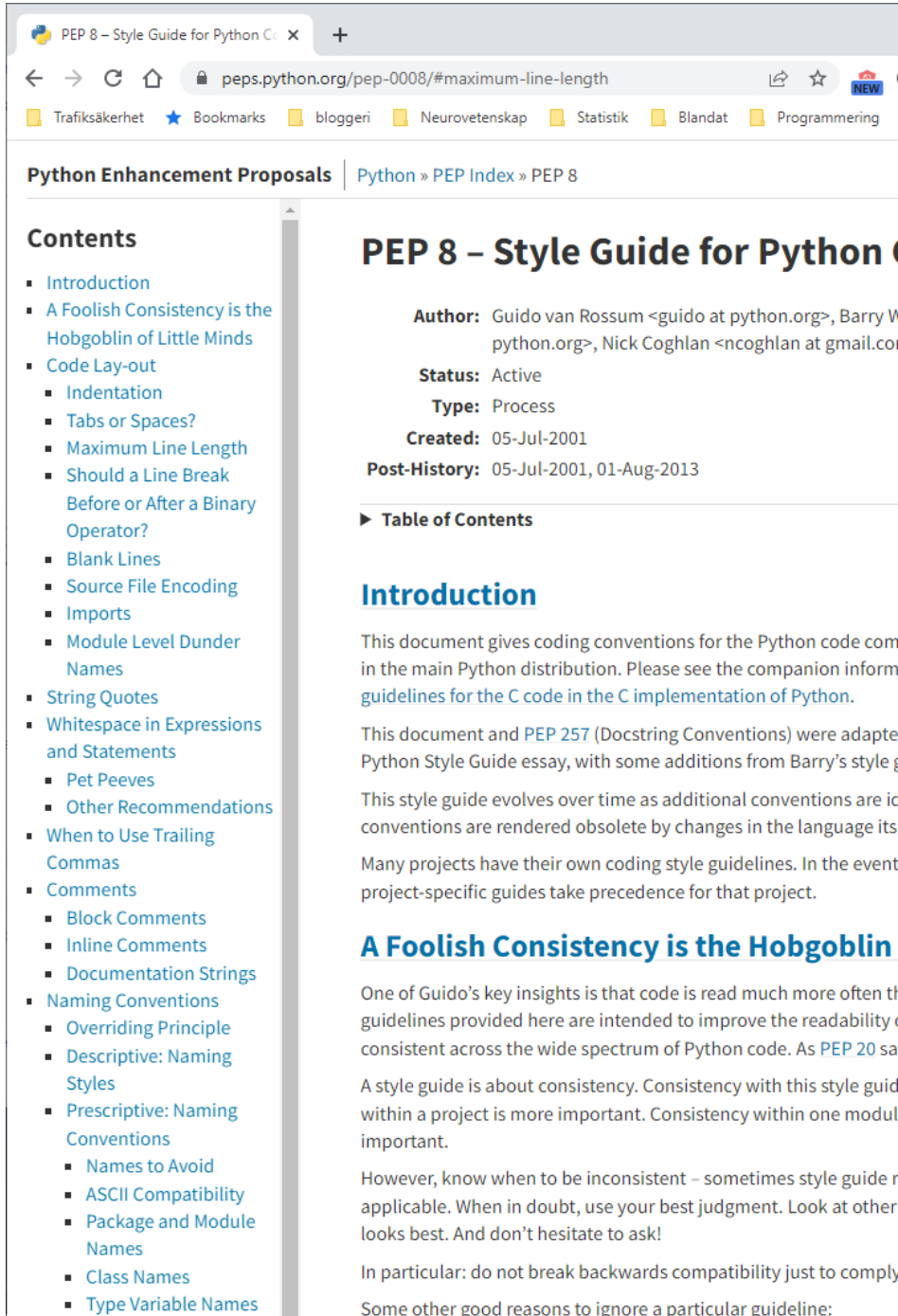
Coding style and naming conventions

Official Python Style Guide:
https://peps.python.org/pep-0008/

R Style Guides from Google and tidyverse:
https://google.github.io/styleguide/Rguide.html
https://style.tidyverse.org/

A community-edited MATLAB style guide:
https://sites.google.com/site/matlabstyleguidelines/home

Let's look at some recurring themes…



… avoid being overly cryptic – e.g., show individual steps

```python
import numpy as np
import matplotlib.pyplot as plt

ym = np.full(3, np.nan)

ayd = np.loadtxt('grpdata0.csv', delimiter=',')
yd = ayd[:, 1]
ym[0] = np.mean(yd)

ayd = np.loadtxt('grpdata1.csv', delimiter=',')
yd = ayd[:, 1]
ym[1] = np.mean(yd)

ayd = np.loadtxt('grpdata2.csv', delimiter=',')
yd = ayd[:, 1]
ym[2] = np.mean(yd)

plt.plot(ym)
plt.xlabel('Group')
plt.ylabel('Mean value')
plt.show()
```

… use meaningful variable names, avoid “magical constants”

```python
import numpy as np
import matplotlib.pyplot as plt

N_GROUPS = 3
VALID_DATA_COLUMN = 1

group_means = np.full(N_GROUPS, np.nan)

all_group_data = np.loadtxt('grpdata0.csv', delimiter=',')
group_data = all_group_data[:, VALID_DATA_COLUMN]
group_means[0] = np.mean(group_data)

all_group_data = np.loadtxt('grpdata1.csv', delimiter=',')
group_data = all_group_data[:, VALID_DATA_COLUMN]
group_means[1] = np.mean(group_data)

all_group_data = np.loadtxt('grpdata2.csv', delimiter=',')
group_data = all_group_data[:, VALID_DATA_COLUMN]
group_means[2] = np.mean(group_data)

plt.plot(group_means)
plt.xlabel('Group')
plt.ylabel('Mean value')
plt.show()
```

… don’t copy/paste!

```python
import numpy as np
import matplotlib.pyplot as plt

N_GROUPS = 3
GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
VALID_DATA_COLUMN = 1

group_means = np.full(N_GROUPS, np.nan)
for i_group in range(N_GROUPS):
    file_name = GROUP_DATA_FILE_FORMAT % i_group
    all_group_data = np.loadtxt(file_name, delimiter=',')
    group_data = all_group_data[:, VALID_DATA_COLUMN]
    group_means[i_group] = np.mean(group_data)

plt.plot(group_means)
plt.xlabel('Group')
plt.ylabel('Mean value')
plt.show()
```

… separate out code into smaller logical units

```python
import numpy as np
import matplotlib.pyplot as plt

N_GROUPS = 3
GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
VALID_DATA_COLUMN = 1

def load_group_data(i_group):
    file_name = GROUP_DATA_FILE_FORMAT % i_group
    all_group_data = np.loadtxt(file_name, delimiter=',')
    group_data = all_group_data[:, VALID_DATA_COLUMN]
    return group_data


group_means = np.full(N_GROUPS, np.nan)
for i_group in range(N_GROUPS):
    group_data = load_group_data(i_group)
    group_means[i_group] = np.mean(group_data)

plt.plot(group_means)
plt.xlabel('Group')
plt.ylabel('Mean value')
plt.show()
```

Benefits of splitting up code

UNIVERSITY OF LEEDS

- Can make code easier to read and understand
- → Typically less bug-prone
- Possible to test each part separately
- Potentially reusable code

… but not always practical to reorganise existing code to reuse parts of it

Researchers can have a little code copying, as a treat.

Comments and documentation

UNIVERSITY OF LEEDS

- For others reading your code
- The most common other person reading your code:
  Future you
- Exactly how much to comment is a matter of taste…

```python
import numpy as np
import matplotlib.pyplot as plt

# constants
N_GROUPS = 3
GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
VALID_DATA_COLUMN = 1

# function for loading the data for one group
def load_group_data(i_group):
    # read from CSV file
    file_name = GROUP_DATA_FILE_FORMAT % i_group
    all_group_data = np.loadtxt(file_name, delimiter=',')
    # get and return only the valid part of the data
    group_data = all_group_data[:, VALID_DATA_COLUMN]
    return group_data


# loop through groups, load data and get mean for each
group_means = np.full(N_GROUPS, np.nan)
for i_group in range(N_GROUPS):
    group_data = load_group_data(i_group)
    group_means[i_group] = np.mean(group_data)

# plot the means
plt.plot(group_means)
plt.xlabel('Group')
plt.ylabel('Mean value')
plt.show()
```

Test test test!

UNIVERSITY OF LEEDS

- Assertions
- Unit tests
- Debugging

Assertions

UNIVERSITY OF LEEDS

Extra code that test things on the fly, silent as long as all is ok

```python
import numpy as np
import matplotlib.pyplot as plt

# constants
N_GROUPS = 3
GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
VALID_DATA_COLUMN = 1

# function for loading the data for one group
def load_group_data(i_group):
    # read from CSV file
    file_name = GROUP_DATA_FILE_FORMAT % i_group
    all_group_data = np.loadtxt(file_name, delimiter=',')
    # make sure all values in the first column are zero
    assert(np.all(all_group_data[:, 0] == 0))
    # get and return only the valid part of the data
    group_data = all_group_data[:, VALID_DATA_COLUMN]
    return group_data


# loop through groups, load data and get mean for each
group_means = np.full(N_GROUPS, np.nan)
for i_group in range(N_GROUPS):
    group_data = load_group_data(i_group)
    group_means[i_group] = np.mean(group_data)

# plot the means
```

Unit tests

UNIVERSITY OF LEEDS

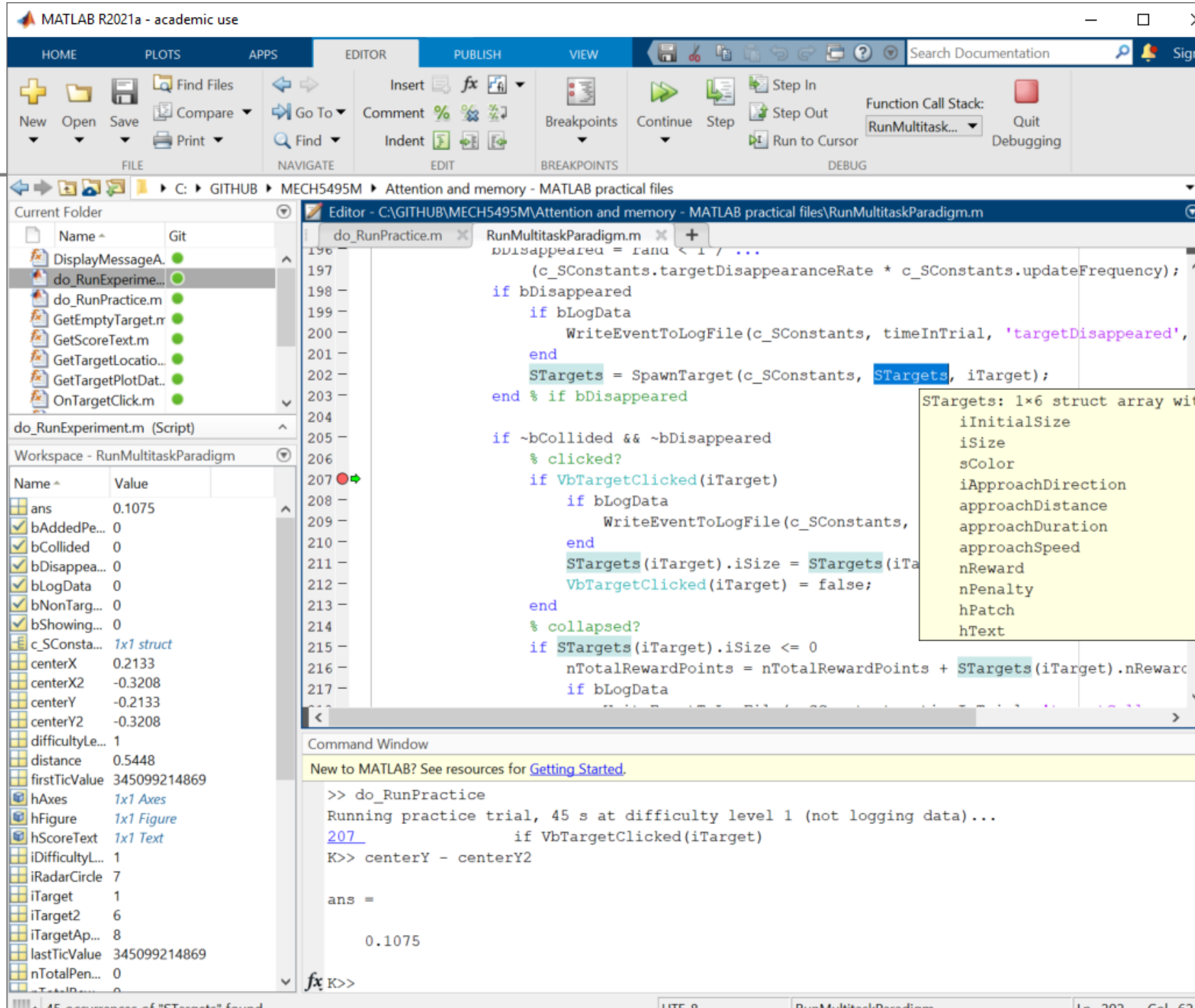
Code that tests some part (unit) of your code by calling/using it

Big SW projects typically rely on lots of automated unit testing – but doesn't need to be big or fancy

```python
import numpy as np

# constants
N_GROUPS = 3
GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
VALID_DATA_COLUMN = 1

# load the data for one group
def load_group_data(i_group):
    # read from CSV file
    file_name = GROUP_DATA_FILE_FORMAT % i_group
    all_group_data = np.loadtxt(file_name, delimiter=',')
    # make sure all values in the first column are zero
    assert(np.all(all_group_data[:, 0] == 0))
    # get and return only the valid part of the data
    group_data = all_group_data[:, VALID_DATA_COLUMN]
    return group_data


if __name__ == '__main__':
    # take a look at the data for the first group
    print(load_group_data(0))
```

Debugging

- Most Integrated Development Environments (IDEs) provide some form of debugger
- Make sure you have a debugger, and learn how to use it!



Gustav's six first coding commandments ☺

UNIVERSITY OF LEEDS

Thou shalt…

1. Not copy/paste code (very much)
2. Split your code into separate functions/scripts/modules/…
3. Learn and adopt conventions for coding style / naming
4. Write code for testing your code
5. Learn to use a debugger
6. Comment and document your code

Thoughts so far?

- Some parts more important?
- Any disagreement?
- Questions?

OK some remaining bits…

UNIVERSITY OF LEEDS

- Project structure conventions
- A note on Notebooks
- Version control
- Open sharing of code
- Code review

Project structure conventions

UNIVERSITY OF LEEDS

A consistent file/folder structure will also help others (including future you) understand your code more quickly

Example from https://goodresearch.dev/setup.html#create-a-project-skeleton:

```
├── data
├── docs
├── results
├── scripts
├── src
├── tests
└── .gitignore
└── environment.yml
└── README.md
```

Project structure conventions

UNIVERSITY OF LEEDS

A consistent file/folder structure will also help others (including future you) understand your code more quickly

Many people also have some convention for indicating analysis sequence, e.g.:

- do_1_beh_CollateAllResponseData.m
- do_2_eeg_LoadBDF_Resample_SaveAsMATFile.m
- do_3_eeg_RunPREPPipeline.m
- do_4_eeg_Filter.m
- do_5_beh_eeg_ConnectDataSetsAndDoExclusion.m
- do_6_beh_ANOVA.m
- do_7_eeg_ICA.m
- do_8_eeg_GetEpochsAndCollateAllERPs.m
- do_9_model_FitToLambleEtAlStudyResults.m
- do_10_model_AnalyseLambleEtAlFits.m
- do_11_model_PrepareLoomingDetectionDataSetForModelFit...
- do_12_model_FitToLoomingStudyParticipantData.m
- do_13_model_AnalyseLoomingStudyFits.m
- do_14_model_VisualiseLoomingStudyFits.m
- do_15_model_LoomingStudyPosteriorPredictiveChecks.m
- do_16_eeg_GetCPPOnsets.m
- do_17_model_MLFitting.m

A note on Notebooks

UNIVERSITY OF LEEDS

A first look at the means

I took a first peek at the means, and they look exactly like I hoped they would! 🍾🎉:

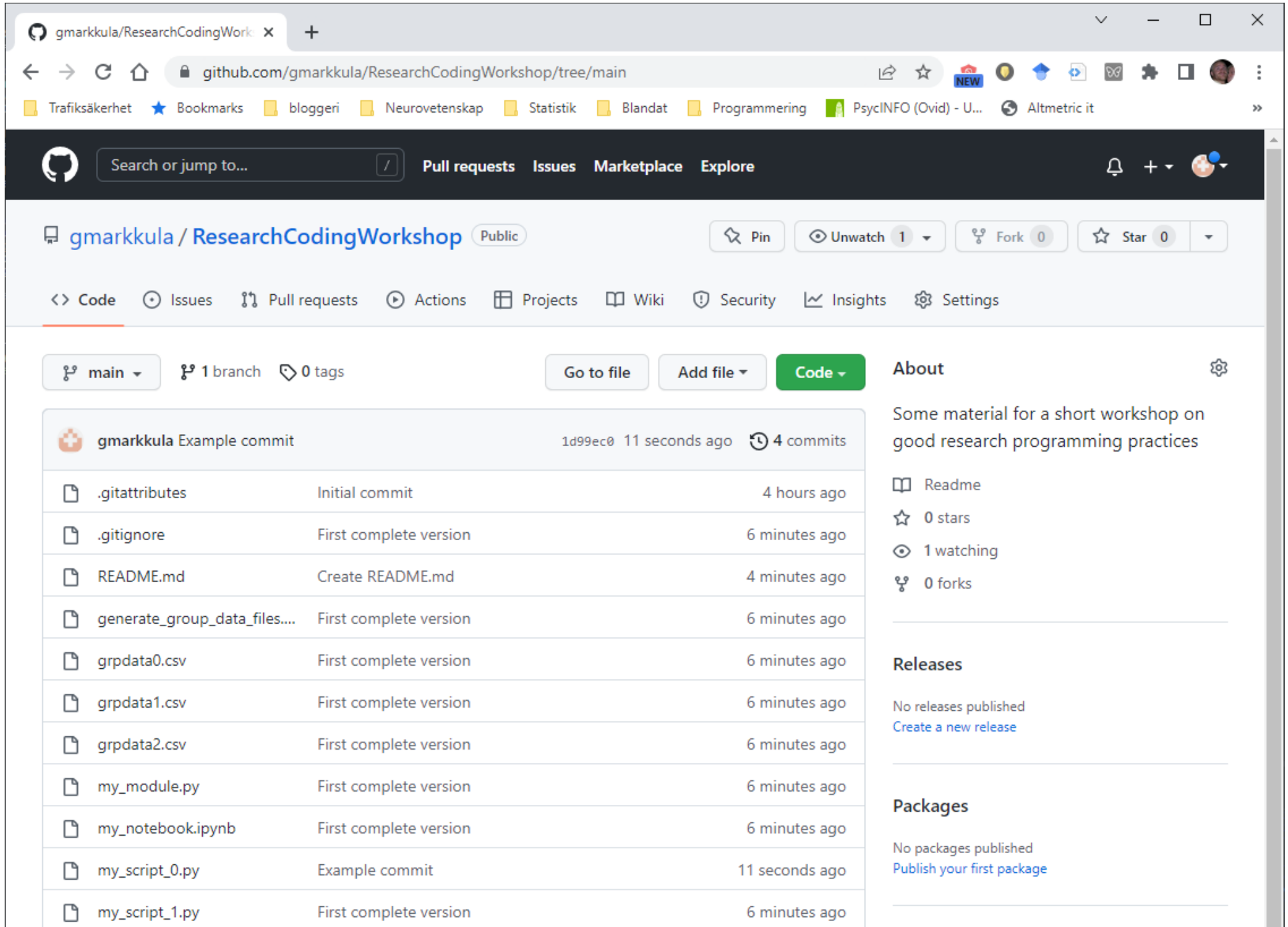
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import my_module

# loop through groups, load data and get mean for each
group_means = np.full(my_module.N_GROUPS, np.nan)
for i_group in range(my_module.N_GROUPS):
    group_data = my_module.load_group_data(i_group)
    group_means[i_group] = np.mean(group_data)

# plot the means
plt.plot(group_means)
plt.xlabel('Group')
plt.ylabel('Mean value')
plt.show()
```

A note on Notebooks

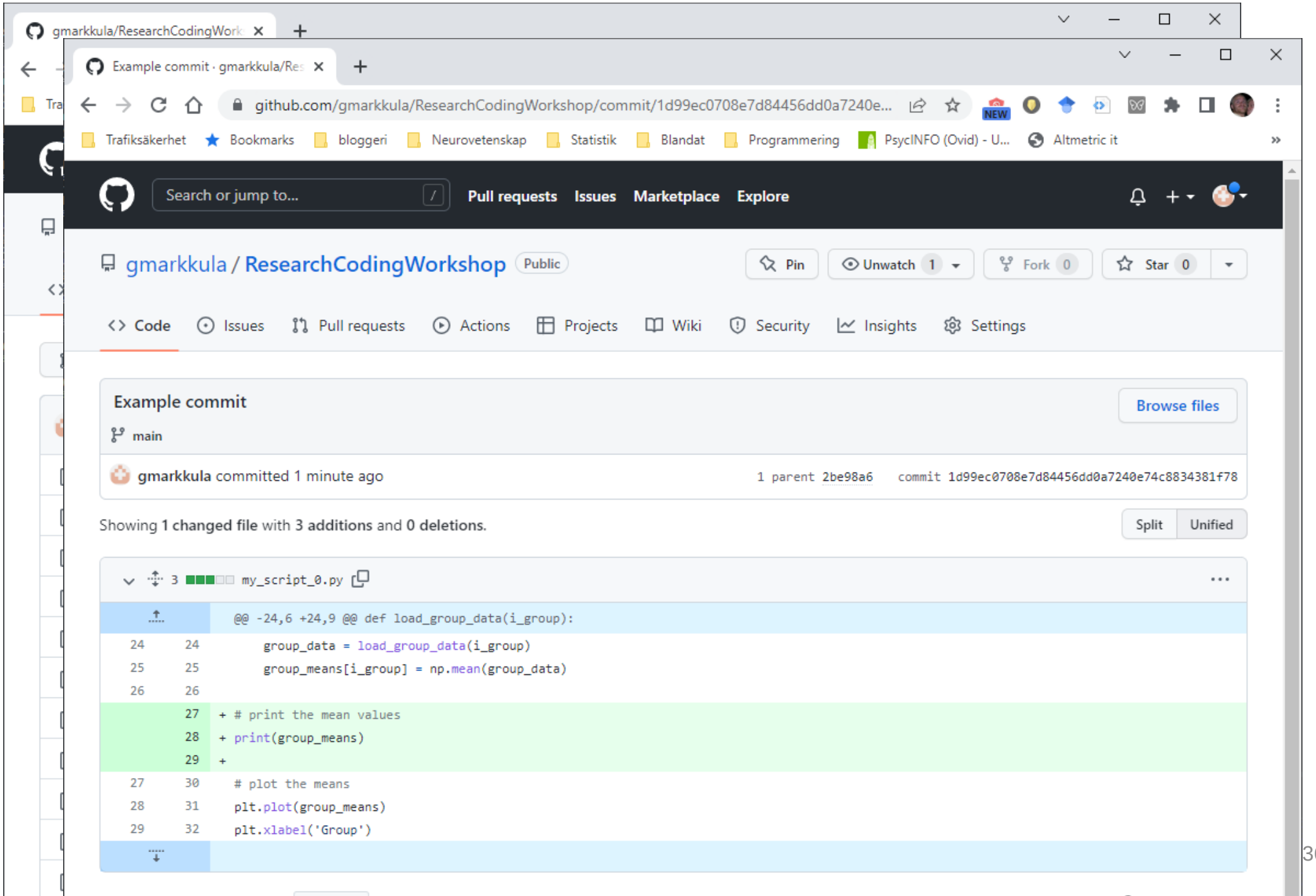
UNIVERSITY OF LEEDS

- Available in Python, R, MATLAB, …
- Wonderful for mixing text, code, images, plots, videos, … !
- Not so wonderful for coding in general
  - https://towardsdatascience.com/the-case-against-the-jupyter-notebook-d4da17e97243
  - State uncertainty → nasty bugs
  - Debugger not always available
  - Can subtly lead you away from other good coding practices, such as modularisation and proper testing…
- My own rules of thumb:
  - Code in notebook itself should be brief and basic
  - Click “Restart kernel and run all cells” often

Version control



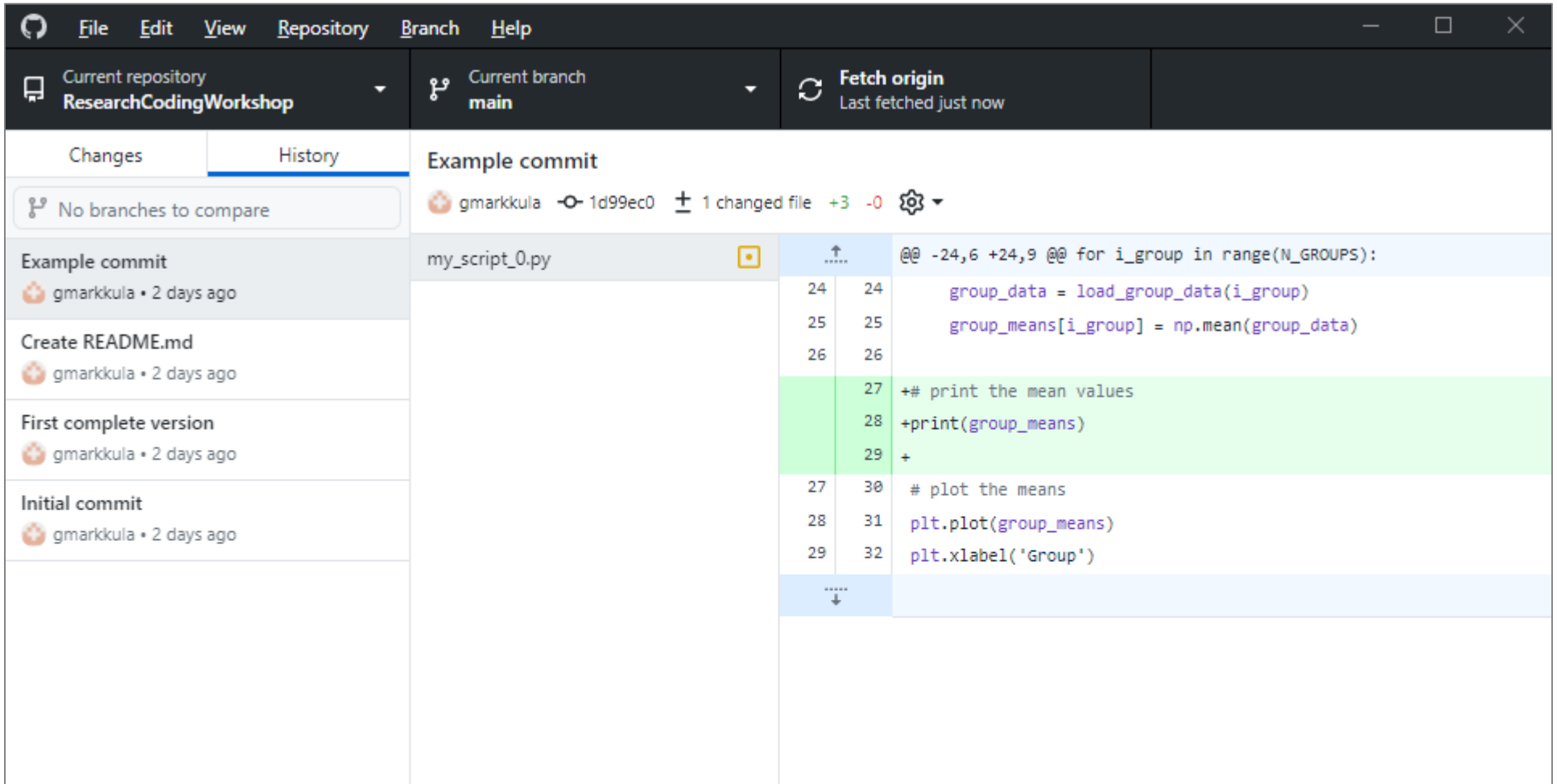
Version control



Version control

Non-hacker suggestion: Try Github Desktop

See e.g.: https://www.codecademy.com/article/what-is-git-and-github-desktop



Open sharing of code

UNIVERSITY OF LEEDS

Increasingly required by funders and journals

→Adds further weight to above points on:

- Code readability
- Documentation and comments

Code review

UNIVERSITY OF LEEDS

"A lab meeting for code"

Example format:

- One person presents own code (e.g., up to a few hundred lines) to a group of peers
- The group provides constructive feedback

http://fperez.org/py4science/code_reviews.html

https://github.com/OxfordCodeReviewNet/forum#guidelines

Gustav's ten coding commandments ☺

UNIVERSITY OF LEEDS

Thou shalt…

1. Not copy/paste code (very much)
2. Split your code into separate functions/scripts/modules/…
3. Learn and adopt conventions for coding style / naming
4. Write code for testing your code
5. Learn to use a debugger
6. Comment and document your code
7. Consider adopting project structure conventions
8. Use Notebooks for the right things, not everything
9. Seriously consider using tools for version control and open code sharing
10. Try code review

So let’s try (one form of) code review

UNIVERSITY OF LEEDS

1. Alone: Check your own code against the commandments and write down:
   →What is the main area where you would like to improve?
   →What can you do concretely to achieve this improvement?
2. In pairs:
   →Show your neighbour where your code is lacking, discuss together
   →Try reviewing each others’ code for any further weaknesses

→Any insights to share with the group?

UNIVERSITY OF LEEDS

General discussion