



UNIVERSITY OF LEEDS

Workshop: Research coding

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

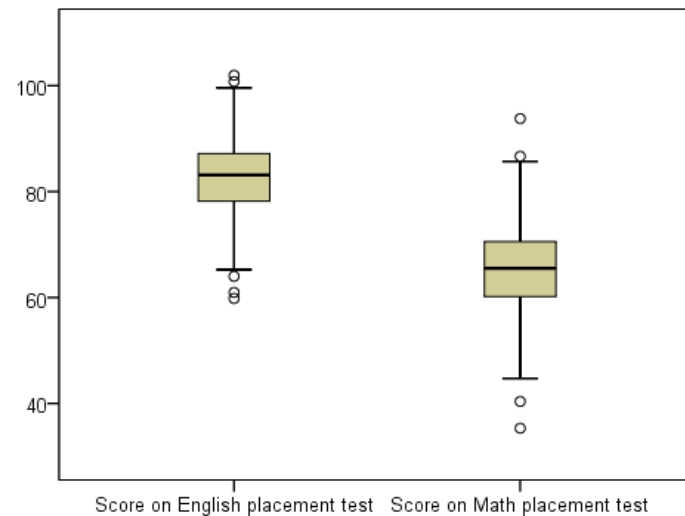
Gustav Markkula

-
- 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

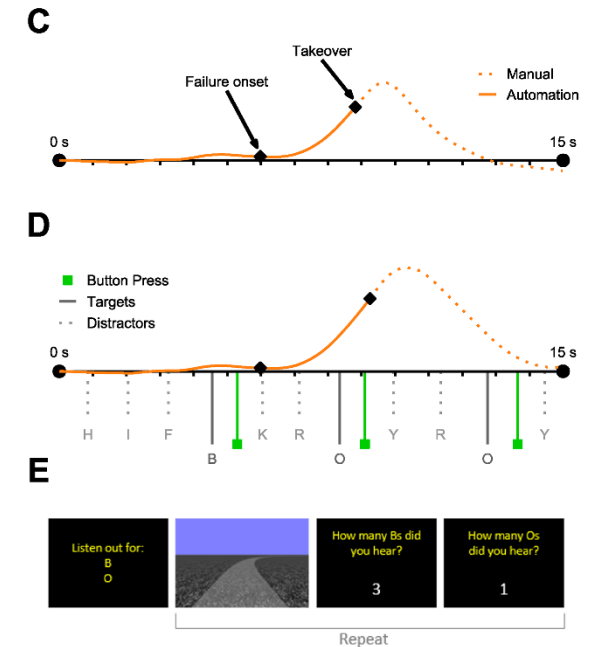
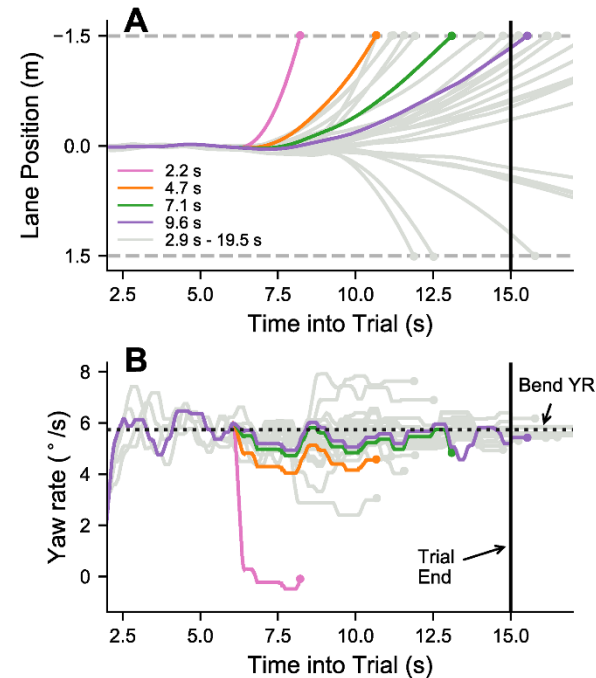
- Lecture – my two cents
 - Coding – why and what and how?
 - Your brain on code
 - Good practices and less good practices
 - ... for some of you a lot of this will be familiar
- Code review alone and in pairs
- General discussion

Coding – why?

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



(source)



(Mole et al., 2020)

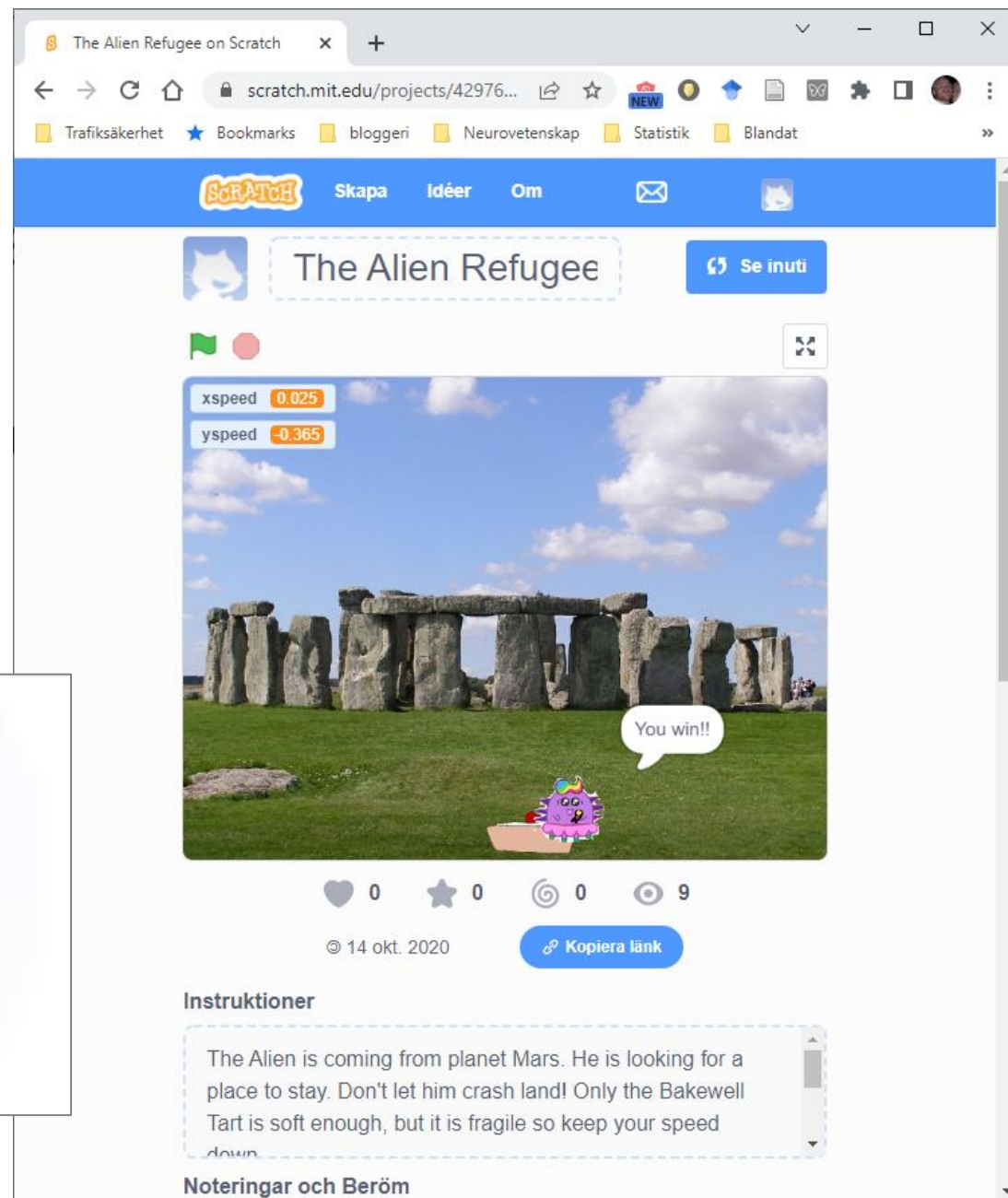
Coding – why?

Fun!

<https://scratch.mit.edu/>

<https://p5js.org/>

<https://unity.com/>



Coding – what?



Coding – how?

Anyone can learn programming!

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



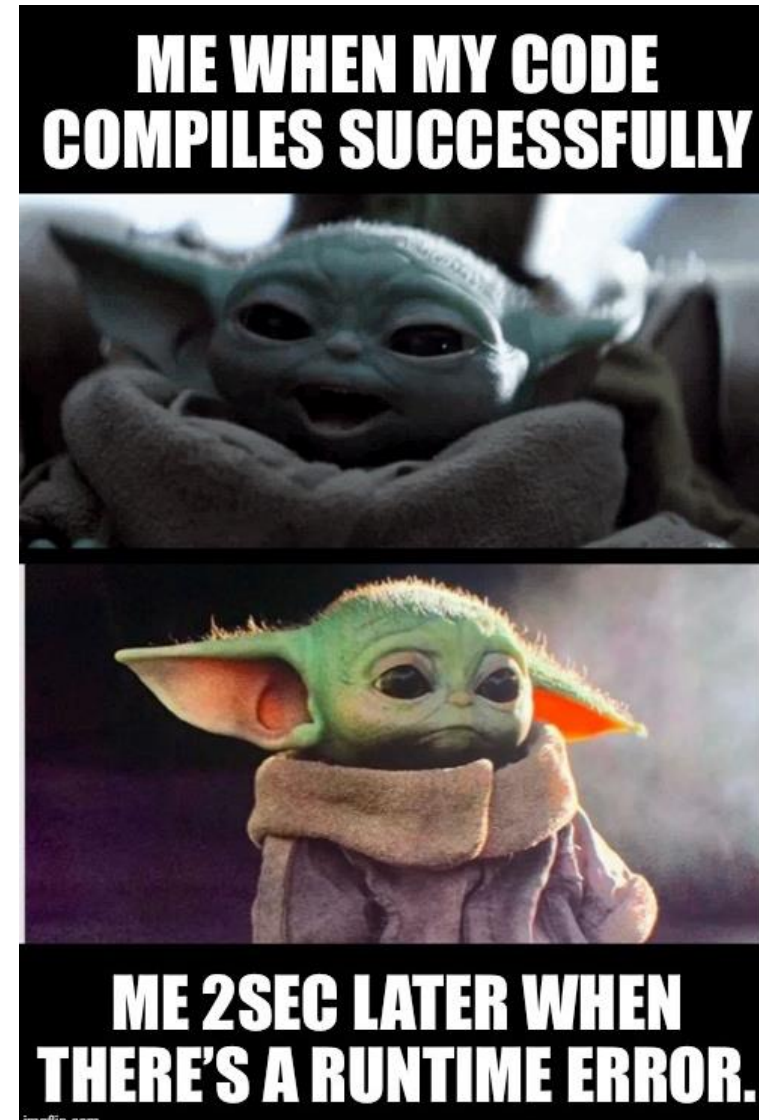
Coding – how?

... 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

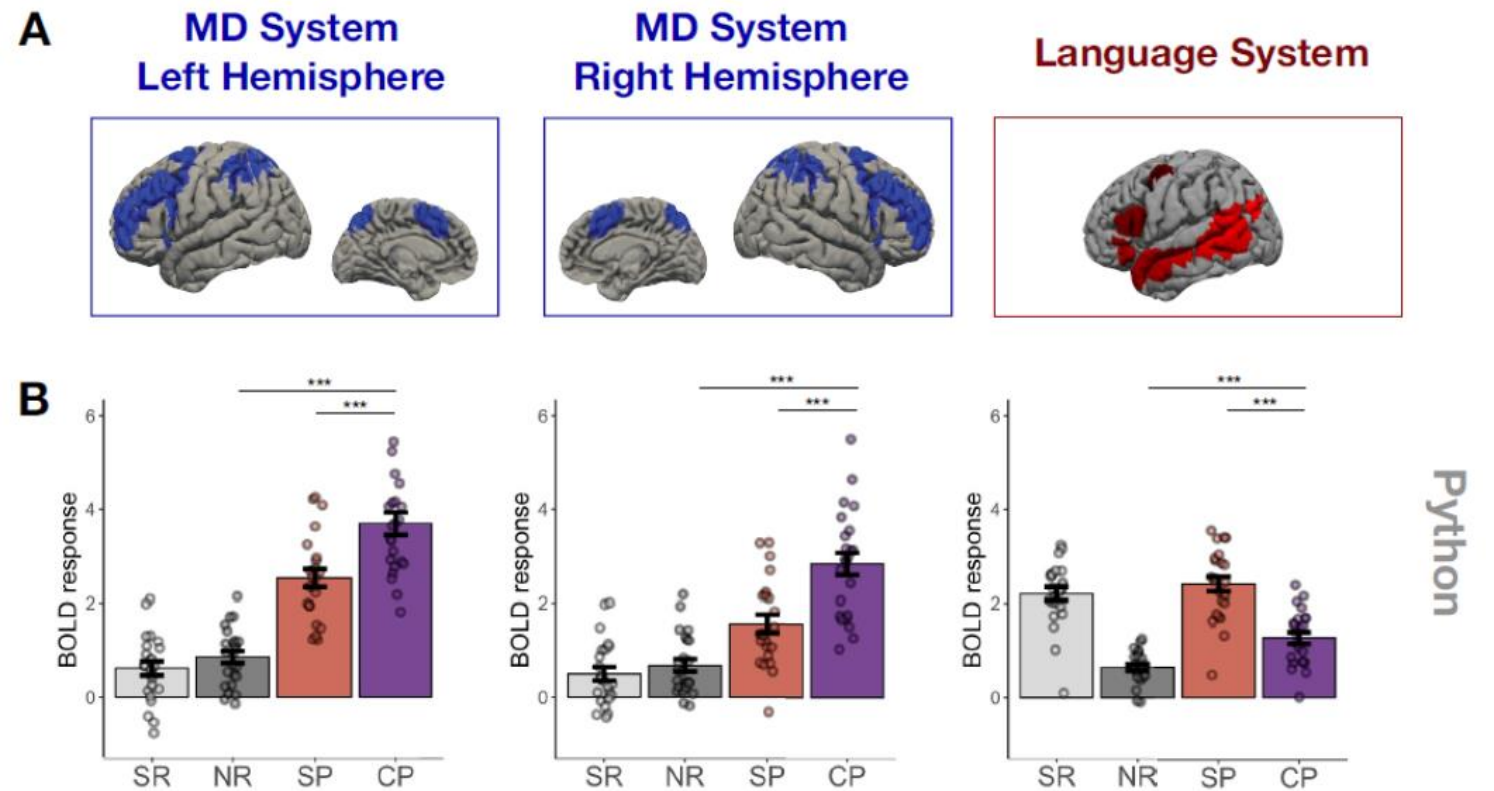


Fig. 1: Code problems (CP; purple bars) created activations with higher overlap with the multiple demand system (left and center) than with the language system (right) compared with other tasks like sentence problems (SP), sentence reading (SR) or non-word reading (NR). From Ivanova et al. (2020), used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

Your brain on code

Resource tip:

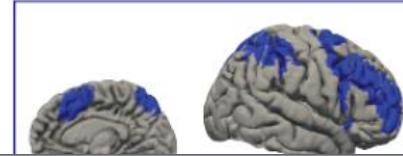
<https://goodresearch.dev>

A

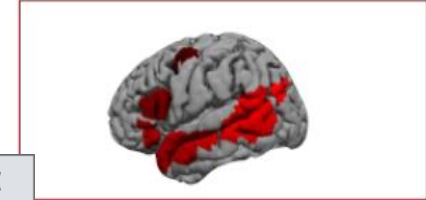
MD System
Left Hemisphere



MD System
Right Hemisphere



Language System



The Good Research Code Handbook

Search this book...

The Good Research Code Handbook

INTRO

Roadmap

Brains & coding

LESSONS

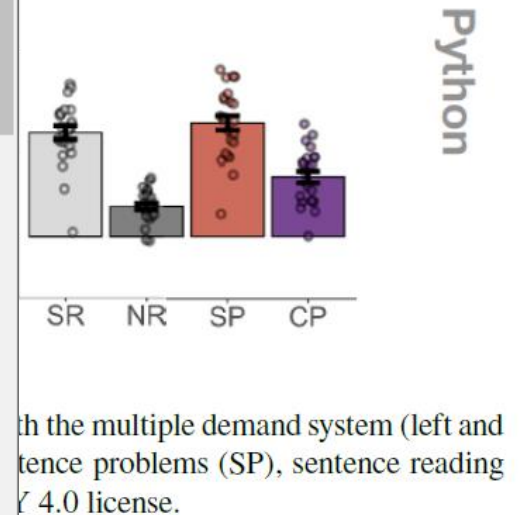
Set up your project

The Good Research Code Handbook

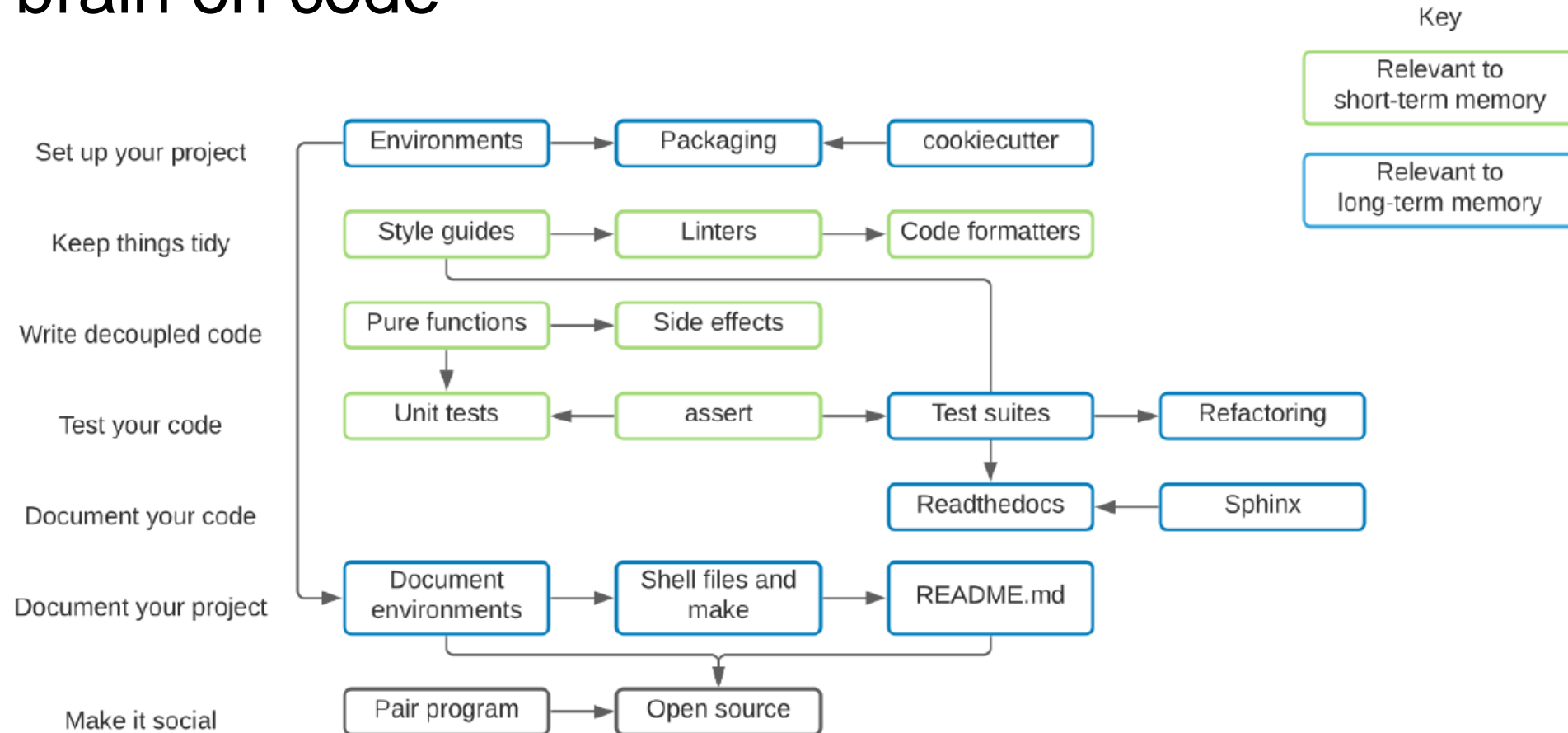
This handbook is for grad students, postdocs and PIs who do a lot of programming as part of their research. It will teach you, in a practical manner, how to organize your code so that it is easy to understand and works reliably.

Most people who write research code are not trained in computer science or software engineering. It can feel like an uphill battle when you have to write code

- Prerequisites
- Why did I make this?
- Citing this handbook
- Alternative formats
- Contact me



Your brain on code



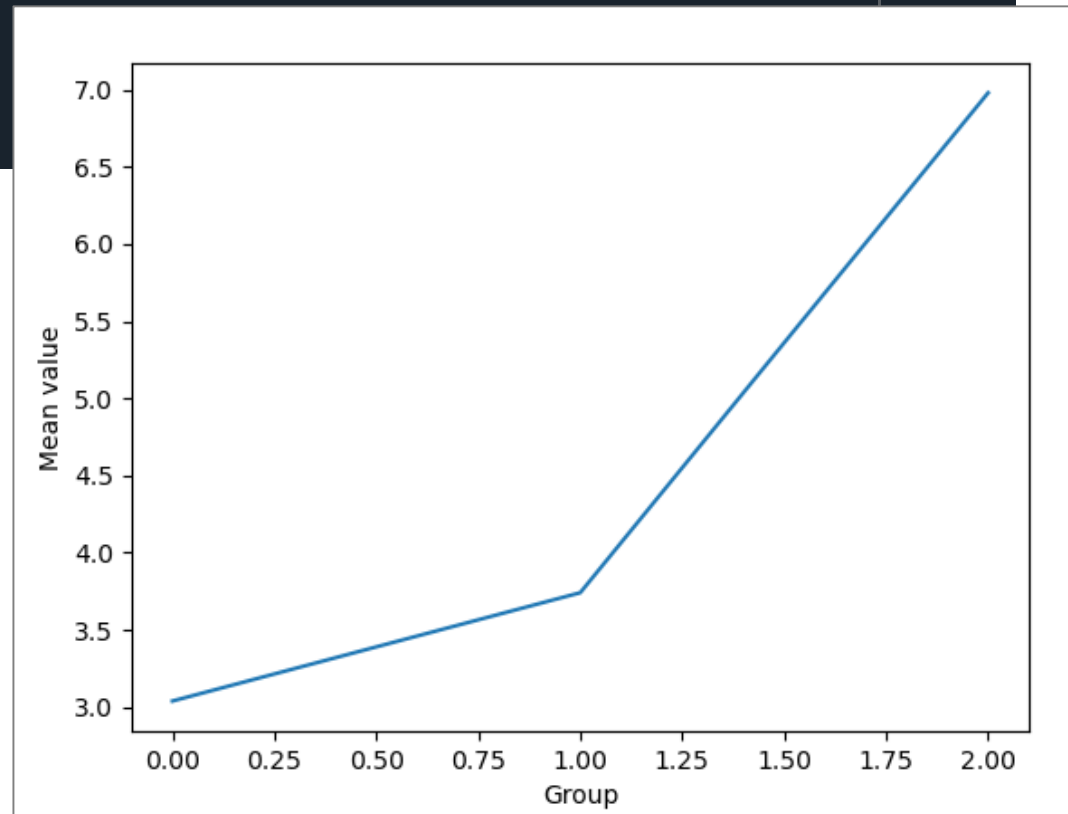
Time to look at some actual code...

```

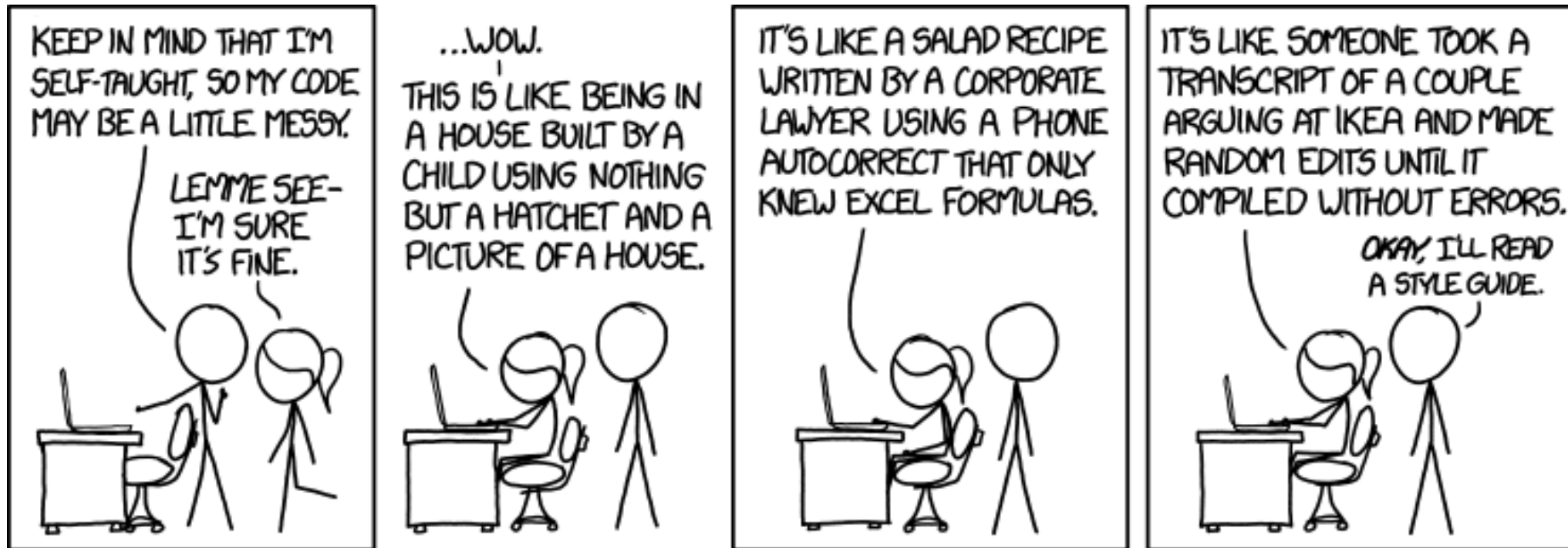
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 ym = np.full(3, np.nan)
5
6 ym[0] = np.mean(np.loadtxt('grpdata0.csv', delimiter=',')[:, 1])
7 ym[1] = np.mean(np.loadtxt('grpdata1.csv', delimiter=',')[:, 1])
8 ym[2] = np.mean(np.loadtxt('grpdata2.csv', delimiter=',')[:, 1])
9
10 plt.plot(ym)
11 plt.xlabel('Group')
12 plt.ylabel('Mean value')
13 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...

The screenshot shows a web browser displaying the PEP 8 – Style Guide for Python Code page. The browser's address bar shows the URL `peps.python.org/pep-0008/#maximum-line-length`. The page has a navigation bar with links like "Python Enhancement Proposals", "Python » PEP Index", and "PEP 8". A sidebar on the left contains a "Contents" section with a list of topics including Introduction, A Foolish Consistency is the Hobgoblin of Little Minds, Code Lay-out, Indentation, Tabs or Spaces?, Maximum Line Length, Should a Line Break Before or After a Binary Operator?, Blank Lines, Source File Encoding, Imports, Module Level Dunder Names, String Quotes, Whitespace in Expressions and Statements, Pet Peeves, Other Recommendations, When to Use Trailing Commas, Comments, Block Comments, Inline Comments, Documentation Strings, Naming Conventions, Overriding Principle, Descriptive: Naming Styles, Prescriptive: Naming Conventions, Names to Avoid, ASCII Compatibility, Package and Module Names, Class Names, and Type Variable Names. The main content area has the title "PEP 8 – Style Guide for Python Code" and includes metadata: Author (Guido van Rossum, Barry Warsaw, Nick Coghlan), Status (Active), Type (Process), Created (05-Jul-2001), and Post-History (05-Jul-2001, 01-Aug-2013). Below this is a "Table of Contents" section and an "Introduction" section. The introduction text states: "This document gives coding conventions for the Python code comprised by the main Python distribution. Please see the companion informal guidelines for the C code in the C implementation of Python." It also mentions that the document and PEP 257 (Docstring Conventions) were adapted from the Python Style Guide essay, with some additions from Barry's style guide. It notes that the style guide evolves over time as additional conventions are added and that many projects have their own coding style guidelines, with project-specific guides taking precedence.

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

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 ym = np.full(3, np.nan)
5
6 ayd = np.loadtxt('grpdata0.csv', delimiter=',')
7 yd = ayd[:, 1]
8 ym[0] = np.mean(yd)
9
10 ayd = np.loadtxt('grpdata1.csv', delimiter=',')
11 yd = ayd[:, 1]
12 ym[1] = np.mean(yd)
13
14 ayd = np.loadtxt('grpdata2.csv', delimiter=',')
15 yd = ayd[:, 1]
16 ym[2] = np.mean(yd)
17
18 plt.plot(ym)
19 plt.xlabel('Group')
20 plt.ylabel('Mean value')
21 plt.show()
```

... use meaningful variable names, avoid “magical constants”

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 N_GROUPS = 3
5 VALID_DATA_COLUMN = 1
6
7 group_means = np.full(N_GROUPS, np.nan)
8
9 all_group_data = np.loadtxt('grpdata0.csv', delimiter=',')
10 group_data = all_group_data[:, VALID_DATA_COLUMN]
11 group_means[0] = np.mean(group_data)
12
13 all_group_data = np.loadtxt('grpdata1.csv', delimiter=',')
14 group_data = all_group_data[:, VALID_DATA_COLUMN]
15 group_means[1] = np.mean(group_data)
16
17 all_group_data = np.loadtxt('grpdata2.csv', delimiter=',')
18 group_data = all_group_data[:, VALID_DATA_COLUMN]
19 group_means[2] = np.mean(group_data)
20
21 plt.plot(group_means)
22 plt.xlabel('Group')
23 plt.ylabel('Mean value')
24 plt.show()
```

... don't copy/paste!

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 N_GROUPS = 3
5 GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
6 VALID_DATA_COLUMN = 1
7
8 group_means = np.full(N_GROUPS, np.nan)
9 for i_group in range(N_GROUPS):
10     file_name = GROUP_DATA_FILE_FORMAT % i_group
11     all_group_data = np.loadtxt(file_name, delimiter=',')
12     group_data = all_group_data[:, VALID_DATA_COLUMN]
13     group_means[i_group] = np.mean(group_data)
14
15 plt.plot(group_means)
16 plt.xlabel('Group')
17 plt.ylabel('Mean value')
18 plt.show()
```

... separate out code into smaller logical units

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 N_GROUPS = 3
5 GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
6 VALID_DATA_COLUMN = 1
7
8 def load_group_data(i_group):
9     file_name = GROUP_DATA_FILE_FORMAT % i_group
10    all_group_data = np.loadtxt(file_name, delimiter=',')
11    group_data = all_group_data[:, VALID_DATA_COLUMN]
12    return group_data
13
14
15 group_means = np.full(N_GROUPS, np.nan)
16 for i_group in range(N_GROUPS):
17     group_data = load_group_data(i_group)
18     group_means[i_group] = np.mean(group_data)
19
20 plt.plot(group_means)
21 plt.xlabel('Group')
22 plt.ylabel('Mean value')
23 plt.show()
```

Benefits of splitting up code

- 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.

- 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...

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # constants
5  N_GROUPS = 3
6  GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
7  VALID_DATA_COLUMN = 1
8
9  # function for loading the data for one group
10 def load_group_data(i_group):
11     # read from CSV file
12     file_name = GROUP_DATA_FILE_FORMAT % i_group
13     all_group_data = np.loadtxt(file_name, delimiter=',')
14     # get and return only the valid part of the data
15     group_data = all_group_data[:, VALID_DATA_COLUMN]
16     return group_data
17
18
19 # loop through groups, load data and get mean for each
20 group_means = np.full(N_GROUPS, np.nan)
21 for i_group in range(N_GROUPS):
22     group_data = load_group_data(i_group)
23     group_means[i_group] = np.mean(group_data)
24
25 # plot the means
26 plt.plot(group_means)
27 plt.xlabel('Group')
28 plt.ylabel('Mean value')
29 plt.show()

```

Test test test!

- Assertions
- Unit tests
- Debugging

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

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # constants
5 N_GROUPS = 3
6 GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
7 VALID_DATA_COLUMN = 1
8
9 # function for loading the data for one group
10 def load_group_data(i_group):
11     # read from CSV file
12     file_name = GROUP_DATA_FILE_FORMAT % i_group
13     all_group_data = np.loadtxt(file_name, delimiter=',')
14     # make sure all values in the first column are zero
15     assert(np.all(all_group_data[:, 0] == 0))
16     # get and return only the valid part of the data
17     group_data = all_group_data[:, VALID_DATA_COLUMN]
18     return group_data
19
20
21 # loop through groups, load data and get mean for each
22 group_means = np.full(N_GROUPS, np.nan)
23 for i_group in range(N_GROUPS):
24     group_data = load_group_data(i_group)
25     group_means[i_group] = np.mean(group_data)
26
27 # plot the means
```

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

```
1  import numpy as np
2
3  # constants
4  N_GROUPS = 3
5  GROUP_DATA_FILE_FORMAT = 'grpdata%i.csv'
6  VALID_DATA_COLUMN = 1
7
8  # load the data for one group
9  def load_group_data(i_group):
10     # read from CSV file
11     file_name = GROUP_DATA_FILE_FORMAT % i_group
12     all_group_data = np.loadtxt(file_name, delimiter=',')
13     # make sure all values in the first column are zero
14     assert(np.all(all_group_data[:, 0] == 0))
15     # get and return only the valid part of the data
16     group_data = all_group_data[:, VALID_DATA_COLUMN]
17     return group_data
18
19
20 if __name__ == '__main__':
21     # take a look at the data for the first group
22     print(load_group_data(0))
```

- 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 😊

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...

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

Project structure conventions

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

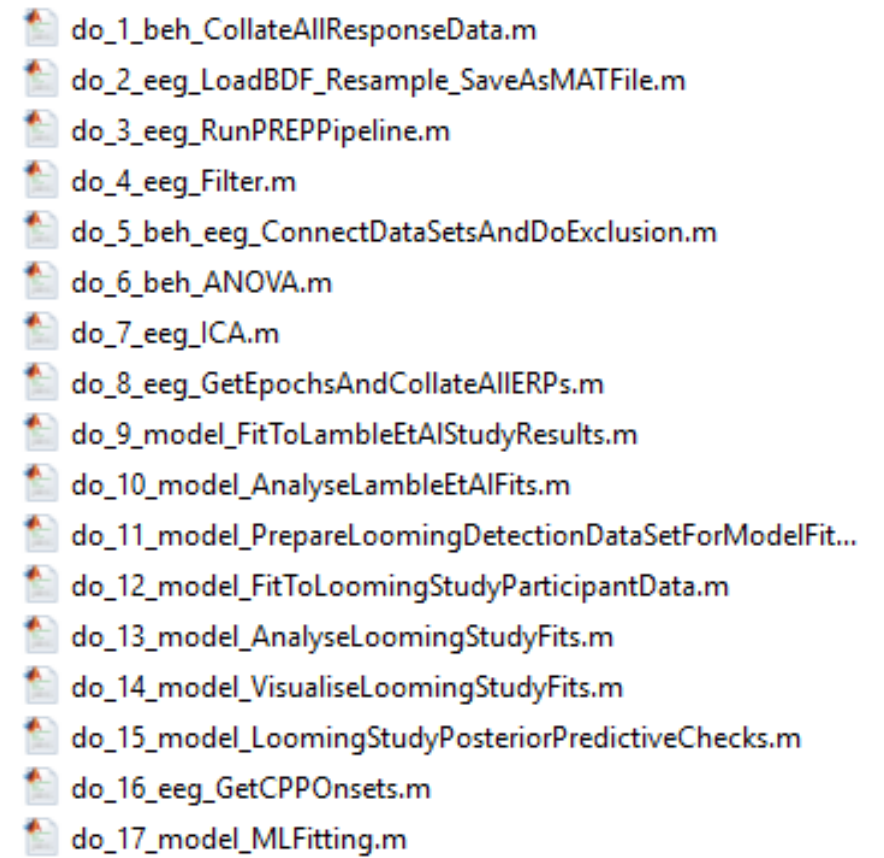
Example from <https://goodresearch.dev/setup.html#create-a-project-skeleton>:

```
|— data  
|— docs  
|— results  
|— scripts  
|— src  
|— tests  
└— .gitignore  
└— environment.yml  
└— README.md
```

Project structure conventions

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 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()
```



- 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

The screenshot shows a web browser displaying the GitHub repository page for 'gmarkkula/ResearchCodingWorkshop'. The browser's address bar shows the URL 'github.com/gmarkkula/ResearchCodingWorkshop/tree/main'. The repository is public and has 1 branch (main) and 0 tags. The 'main' branch is selected. The repository contains several files, including .gitattributes, .gitignore, README.md, generate_group_data_files..., grpdata0.csv, grpdata1.csv, grpdata2.csv, my_module.py, my_notebook.ipynb, my_script_0.py, and my_script_1.py. The 'About' section on the right provides a brief description of the repository: 'Some material for a short workshop on good research programming practices'. It also shows 0 stars, 1 watching, and 0 forks. The 'Releases' section indicates that no releases have been published, and the 'Packages' section indicates that no packages have been published.

gmarkkula/ResearchCodingWorkshop x +

github.com/gmarkkula/ResearchCodingWorkshop/tree/main

Trafiksäkerhet ★ Bookmarks bloggeri Neurovetenskap Statistik Blandat Programmering PsyclINFO (Ovid) - U... Altmetric it

Search or jump to... / Pull requests Issues Marketplace Explore

gmarkkula / ResearchCodingWorkshop Public

Pin Unwatch 1 Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

Go to file Add file Code

gmarkkula Example commit 1d99ec0 11 seconds ago 4 commits

.gitattributes	Initial commit	4 hours ago
.gitignore	First complete version	6 minutes ago
README.md	Create README.md	4 minutes ago
generate_group_data_files....	First complete version	6 minutes ago
grpdata0.csv	First complete version	6 minutes ago
grpdata1.csv	First complete version	6 minutes ago
grpdata2.csv	First complete version	6 minutes ago
my_module.py	First complete version	6 minutes ago
my_notebook.ipynb	First complete version	6 minutes ago
my_script_0.py	Example commit	11 seconds ago
my_script_1.py	First complete version	6 minutes ago

About

Some material for a short workshop on good research programming practices

Readme 0 stars 1 watching 0 forks

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Version control

The screenshot shows a web browser displaying a GitHub commit page for the repository `gmarkkula/ResearchCodingWorkshop`. The commit is titled "Example commit" and is on the `main` branch. It was committed 1 minute ago by `gmarkkula`. The commit message is "Example commit". The commit hash is `1d99ec0708e7d84456dd0a7240e74c8834381f78`. The commit shows 1 changed file with 3 additions and 0 deletions. The file is `my_script_0.py`. The diff shows the following changes:

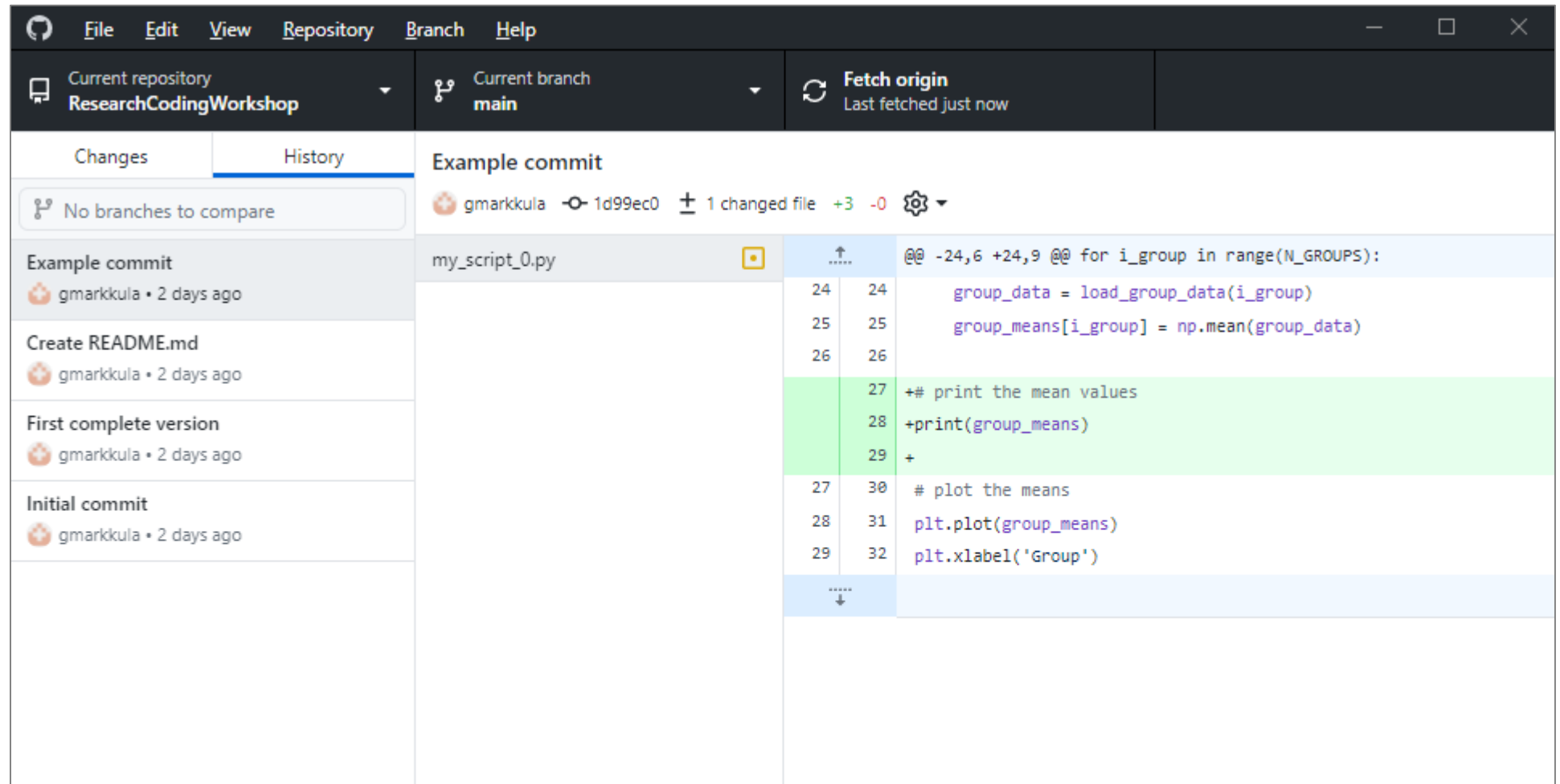
```
@@ -24,6 +24,9 @@ def load_group_data(i_group):
24      24      group_data = load_group_data(i_group)
25      25      group_means[i_group] = np.mean(group_data)
26      26
27      27      + # print the mean values
28      28      + print(group_means)
29      29      +
30      30      # plot the means
31      31      plt.plot(group_means)
32      32      plt.xlabel('Group')
```

The interface includes a search bar, navigation links (Pull requests, Issues, Marketplace, Explore), and a sidebar with repository details (gmarkkula / ResearchCodingWorkshop, Public, Pin, Unwatch, Fork, Star).

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

Increasingly required by funders and journals

→ Adds further weight to above points on:

- Code readability
- Documentation and comments

“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 😊

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

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