

Welcome.

Everyone:

- Pull the updates from the course GitHub repo:
 - `cd <46120-PiWE repo>`
 - `git pull upstream main` ← you might have “upstream2” instead

Physical students:

- Sit WHEREVER you want. 😎
- Turn off laptop volume (mute). ⬅️**IMPORTANT!**
- Log into the Zoom meeting.
 - Microphone muted. Camera off.



46120: Scientific Programming for Wind Energy

Packaging, linters and more

Jenni Rinker



Agenda for today

- Results from the midterm evaluation
- CodeCamp tips and tricks
- PEPs and linters
- Packaging

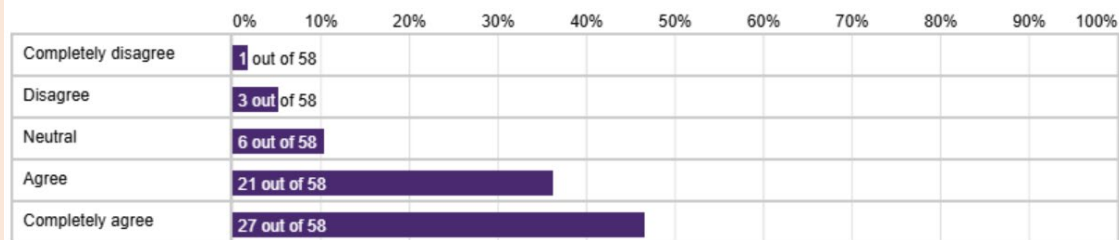


Midterm evaluation results

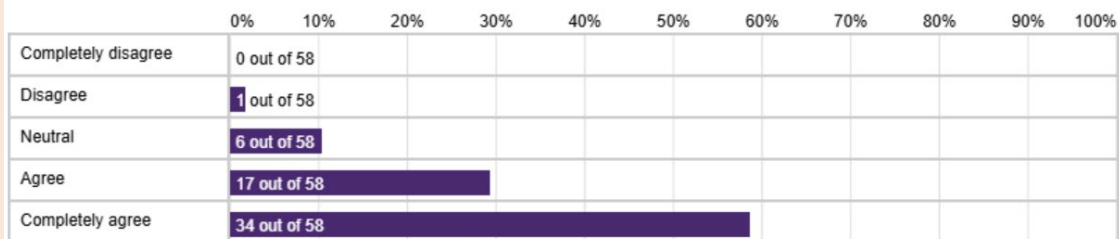
How your feedback has changed the
course plan.



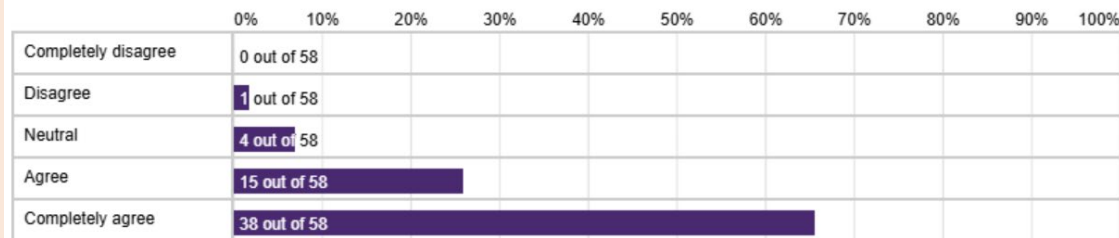
1 In my opinion, I am learning a lot in this course



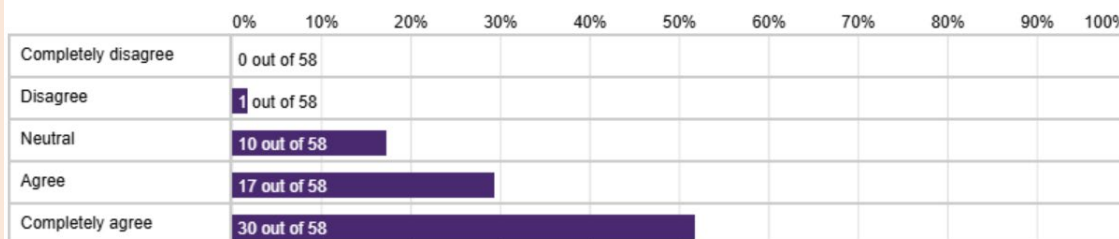
2 The learning activities motivate me to work with the material



3 During the course, I have had the opportunity to get feedback on my performance

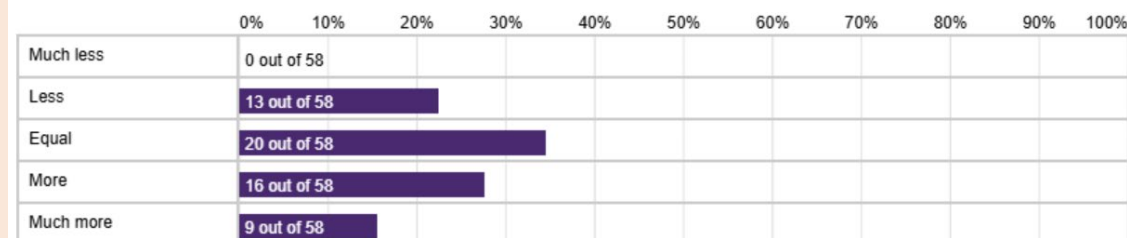


4 It is generally clear what is expected of me in exercises, project work, etc.

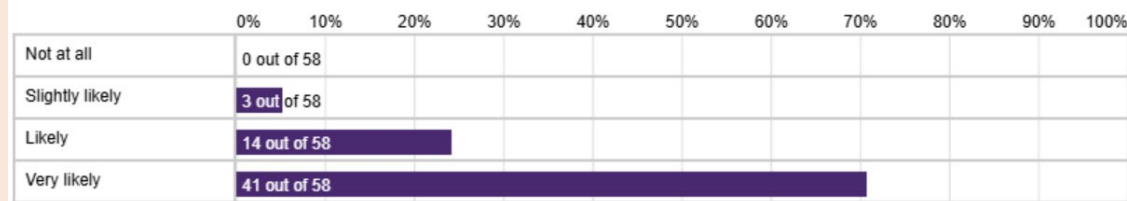


- 58/72 (81%), pretty okay completion.
- Generally very good. Esp. 1/2/4/6/7.
- Some room for improvement on clarity, mentioned in free text.
- Workload a little too high.
- But looks like students mostly like it. Cool!

5 5 points corresponds to 9h./week. I think that my average workload in this course is



6 How likely are you to recommend this class to other students?



7 I am generally enjoying working with my group on the assignments



ChatGPT summary of “What (if anything) did you like about the CodeCamp project?”

raw text at end

Positive Aspects:

- **Project Organization:** The project was well-organized, with clear instructions and structured tasks that made it easy to follow.
- **Learning Experience:** Participants learned a lot about coding, scripting, and using **GitHub**, even if they initially disliked coding.
- **Group Work:** Working in groups was beneficial, providing insights into coding and collaboration.
- **Practical Application:** The project allowed for practical application of theoretical concepts, particularly in wind energy systems.
- **Feedback and Improvement:** Detailed instructions and peer feedback were helpful, motivating participants to improve their coding skills.
- **Gradual Progression:** The project was **built up week by week**, which helped in understanding and fixing errors gradually.
- **GitHub Integration:** The use of GitHub for collaborative coding was appreciated and found to be very useful.

Areas for Improvement:

- **Task Clarity:** Some task descriptions were unclear, affecting the outcome and workload.
- **Workload Division:** Dividing group work led to some participants missing out on certain skills.
- **Python Understanding:** Some participants found Python difficult to understand at a deeper level.

Overall, the feedback indicates that the project was highly interactive, educational, and well-structured, with a few areas that could be improved for better clarity and skill distribution.



ChatGPT summary of “What would you like to see improved with the CodeCamp project?”?

raw text at end

- **Task Clarity:** Clearer explanations are needed to reduce confusion and repeated questions.
- **GitHub Instruction:** More detailed explanations and a **crash course on pull requests and merging** would be beneficial.
- **Live Coding:** Incorporating **more live coding** sessions could enhance understanding.
- **Project Structure:** Consider making the course graded to motivate participants and having two projects for different skill levels.
- **Feedback Sessions:** Round Robin feedback sessions were hectic; **more time for peer review** would be appreciated.
- **Evaluation Stress:** Extending deadlines or reducing the number of teams evaluated could help alleviate stress.
- **Assignment Clarity:** Clearer instructions for assignments are needed.
- **Task Availability:** Making all tasks available at the start would allow participants to work at their own pace.
- **Commit and Pull Requests:** Ensuring fair distribution of commits and pull requests is important.
- **Advanced Skills:** Including more advanced skills like code cleaning and handling messy data would be helpful.
- **Support Material:** Providing more support material for beginners would be beneficial.
- **Evaluation Time:** Increasing the time for evaluation or reducing the number of teams evaluated would help.
- **Grading Flexibility:** **More flexibility in grading, allowing a small margin of error, would better reflect real-world scenarios.**



What we will change based on your feedback (1/2).

- Y'all like learning GitHub and how to code in teams. AWESOME.
- Codecamp is generally very well reviewed. We keep it. Minor tweaks for clarity.
 - E.g., present the project so it's easier to understand. A lot of comments about clarity, presumably about the plot to deliver.
- Will “enforce” PRs from Week 1, make it explicit in homework.
- Need more support for students with zero programming experience.
 - Part of a larger discussion in the department. What skills are expected for someone starting MSc DTU Wind?
 - My dream: have online, asynchronous courses for basic git, getting started with VS Code, basic Python, numpy and matplotlib. Students accepted to DTU without those skills can learn them before arriving on campus.



What we will change based on your feedback (2/2).

- Round Robins feel too long. You want more live coding/best practices.
 - We'll only do the RRs every once in awhile from now to end of semester. Try to add more live coding.
- CodeCamp evaluations were STRESSFUL. Need more time to give good feedback.
 - In hindsight, makes perfect sense.
 - Hand-in date for final project will be a few days before final presentations. So you have a few days to look through the repos and collect your individual feedback before you meet with your team in class to submit feedback forms.



Notes from CodeCamp

Tips and tricks based on recurring issues.



Notes from codecamp (1/3).

- How to properly handle paths. (live coding)
 - Use `pathlib.Path` instead of `os` when possible.
 - Use `Path.glob(<search pattern>)` to find files in a folder.
 - Paths with backslashes (e.g., `r'.\data\CT.txt'`) will not work on UNIX systems. Use forward slashes and `Path` objects.
 - When defining relative paths, can either start from working directory or use `__file__` to get location of the file being executed. Latter reduces dependency on working directory, but if you move the file, the code breaks.
- Robustness:
 - For maximum robustness, don't specify TI or wind speed. Find it from listed files. But what if user wants to run subset? Asking user input is nice, but need sanitization.
 - Do you make assumption about folder/file name patterns? Should be documented in your README.



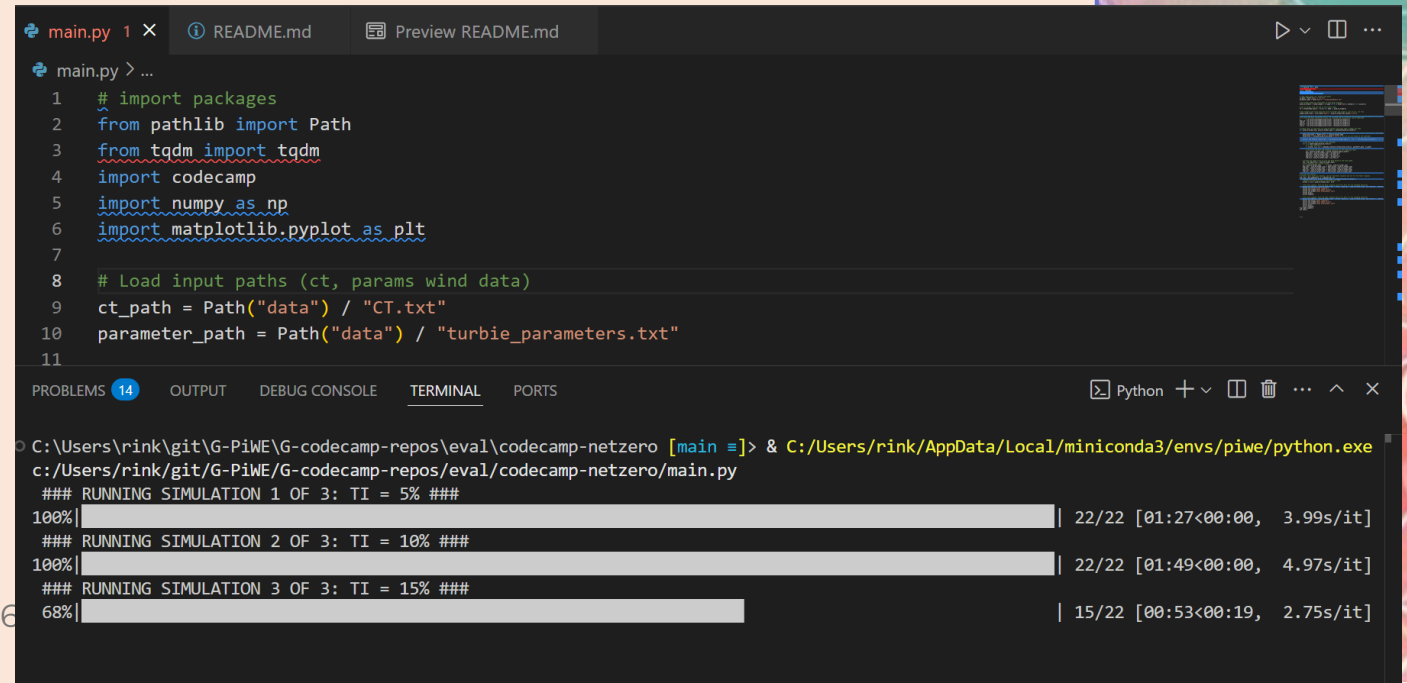
Notes from codecamp (2/3).

- Random git stuff. (live coding)
 - Insights / Network on GitHub. Volunteer team with feature branches?
 - Can filter merge requests by base branch, date. `is:pr is:closed base:main merged:>2025-03-01`
 - Use gitignore file to ignore certain files/folders. Use .gitkeep to keep empty folders.
 - Be careful with case sensitivity and operating systems. Someone on Mac pushing “collaboration.md” when “Collaboration.md” already exists will wreak havoc for Windows user.
- Docstrings
 - Correctly formatted function docstrings allow you to, e.g., [automatically build websites with function inputs/outputs](#).
 - Use [autoDocString extension](#) to automatically make correctly formatted docstring templates based on your function inputs/outputs.



Notes from codecamp (3/3).

- README:
 - Preview README using ctrl-shift-v.
 - Can add your images to README. [Example from BugHunters](#).
 - In general, quick-start guides should include (1) instructions to clone/install, (2) requirements to run, (3) which working directory you need to be in, if relevant.
- Package tqdm for progress bar:



The screenshot shows a VS Code editor with a Python file named `main.py` and a terminal window. The `main.py` file contains the following code:

```
1 # import packages
2 from pathlib import Path
3 from tqdm import tqdm
4 import codecamp
5 import numpy as np
6 import matplotlib.pyplot as plt
7
8 # Load input paths (ct, params wind data)
9 ct_path = Path("data") / "CT.txt"
10 parameter_path = Path("data") / "turbie_parameters.txt"
11
```

The terminal window shows the output of the script, which is running three simulations. The progress bars are displayed as follows:

```
### RUNNING SIMULATION 1 OF 3: TI = 5% ###
100%|████████████████████████████████████████████████████████████████████████████████| 22/22 [01:27<00:00, 3.99s/it]
### RUNNING SIMULATION 2 OF 3: TI = 10% ###
100%|████████████████████████████████████████████████████████████████████████████████| 22/22 [01:49<00:00, 4.97s/it]
### RUNNING SIMULATION 3 OF 3: TI = 15% ###
68%|████████████████████████████████████████████████████████████████████████████████| 15/22 [00:53<00:19, 2.75s/it]
```

Questions?



PEPs and Linters

Standardizing code styles from different authors.



General

- PEP = "Python Enhancement Proposal"
- **Conventions** to standardize python code
 - Not a standard, not enforced (recommendations/suggestions)
 - Violating PEP does not break your code (not like div 0)
 - Aim is to be helpful
- Focus here:
 - PEP8: Style Guide for Python Code [2] - Code layout, naming conventions etc.
 - PEP257: Docstring Conventions [3] - How to document function, classes for user



Why do we need conventions? ^[2,4]

- Reading code happens more often than writing code
- "The power of consistency"
 - Improved readability (also for debugging)
 - Easier recognition of variables & their purpose
 - Easier to follow code logic (e.g. consistent spaces)
 - Prevents "Wild West" in big collaborative efforts
 - Professionalism
- However:
 - Be consistent with surrounding code more important (e. g. in-house project guidelines)



PEP8: Examples - a personal selection

[2, 5]

Binary operators & spaces

```
# Wrong:  
# operators sit far away from their operands  
income = (gross_wages +  
          taxable_interest +  
          (dividends - qualified_dividends) -  
          ira_deduction -  
          student_loan_interest)
```

VS

```
# Correct:  
# easy to match operators with operands  
income = (gross_wages  
          + taxable_interest  
          + (dividends - qualified_dividends)  
          - ira_deduction  
          - student_loan_interest)
```

multiple statements per line (compound statement)

```
# Wrong:  
if foo == 'blah': do_blah_thing()  
do_one(); do_two(); do_three()
```

VS

```
# Correct:  
if foo == 'blah':  
    do_blah_thing()  
do_one()  
do_two()  
do_three()
```

Naming conventions

Type	Public	Internal
Packages	lower_with_under	
Modules	lower_with_under	_lower_with_under
Classes	CapWords	_CapWords
Exceptions	CapWords	
Functions	lower_with_under()	_lower_with_under()
Global/Class Constants	CAPS_WITH_UNDER	_CAPS_WITH_UNDER
Global/Class Variables	lower_with_under	_lower_with_under
Instance Variables	lower_with_under	_lower_with_under
Method Names	lower_with_under()	_lower_with_under()
Function/Method Parameters	lower_with_under	
Local Variables	lower_with_under	

- Complete style guide: [2]

PEP257: Docstrings ^[3]

- In short: strings that appear at top of module or right after definition of a class, function etc. (3x quotation mark)
- Summarize purpose, input arguments and return values (+exceptions raised)
- Single-line (minimum), multi-line (detailed, PEP257)
- Can be accessed via `__doc__` attribute or `help()` function (e.g., `print(np.array.__doc__)` or `help(np.array)`).
- Docstrings for USERS, comments for CODE MODIFIERS
- Different styles possible (e.g. NumPy/SciPy, google style etc.)

```
def square(n):  
    '''Takes in a number n, returns the square of n'''  
    return n**2
```

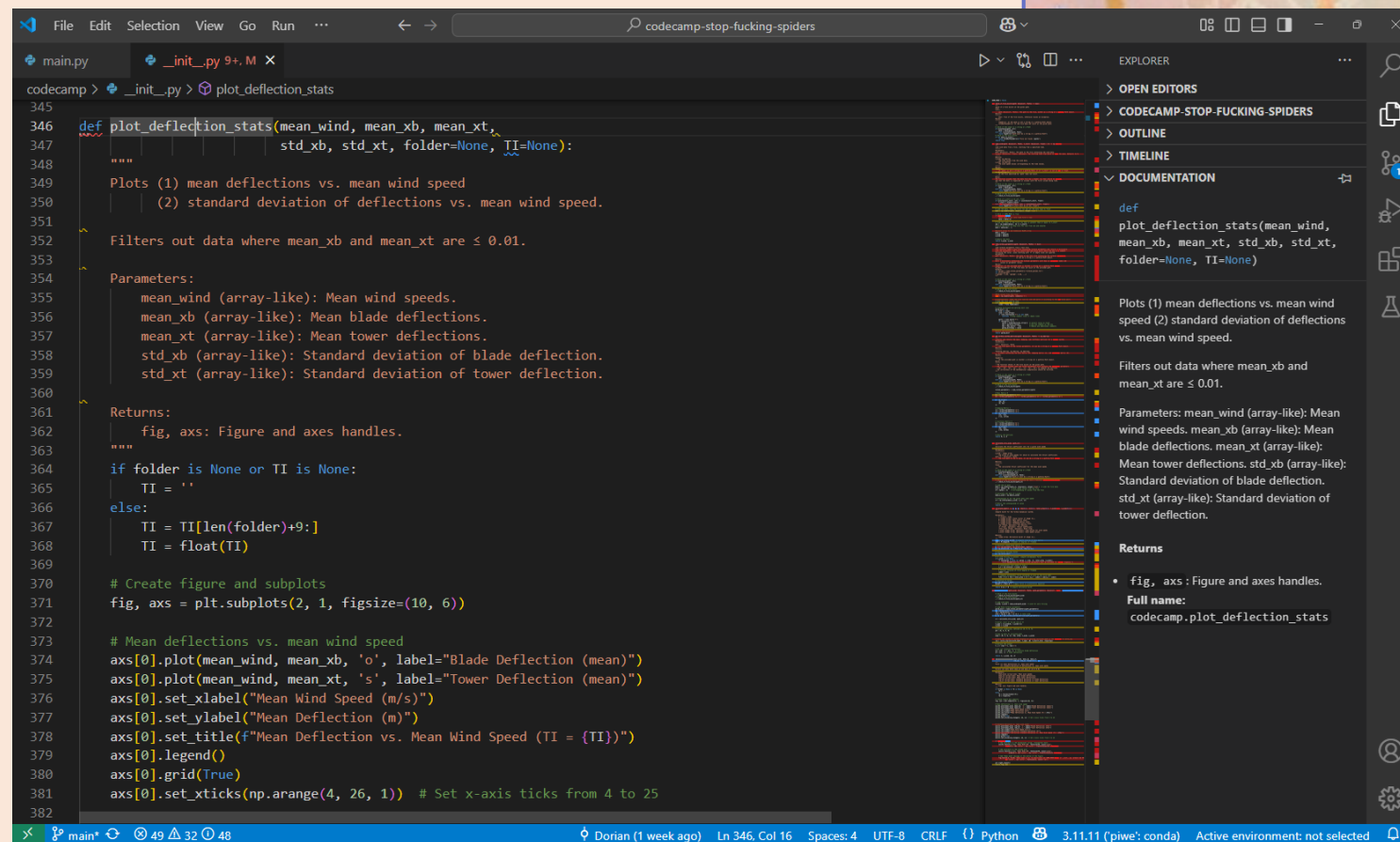
Source: <https://www.programiz.com/python-programming/docstrings>

```
def add(num1, num2):  
    """  
    Add up two integer numbers.  
  
    This function simply wraps the ``+`` operator, and does not  
    do anything interesting, except for illustrating what  
    the docstring of a very simple function looks like.  
  
    Parameters  
    -----  
    num1 : int  
        First number to add.  
    num2 : int  
        Second number to add.  
  
    Returns  
    -----  
    int  
        The sum of ``num1`` and ``num2``.  
  
    See Also  
    -----  
    subtract : Subtract one integer from another.  
  
    Examples  
    -----  
    >>> add(2, 2)  
    4  
    >>> add(25, 0)  
    25  
    >>> add(10, -10)  
    0  
    """  
    return num1 + num2
```

Source: https://pandas.pydata.org/docs/development/contributing_docstring.html

Why docstrings?

- Again: Easier to read and understand what's happening
- Think about the user: what does this function, class etc. do?
- Greatest strength: standardization makes post-processing by other tools easy!
 - Use [Docs View extension](#) to show your documentation in sidebar
 - Tools like Sphinx (documentation generator, [7]) can parse docstring to make beautiful documentation (example [here](#))



The screenshot shows a VS Code editor with a Python file named `__init__.py` open. The file contains a function `plot_deflection_stats` with a detailed docstring. The docstring describes the function's purpose, parameters, filters, and returns. The Docs View extension is installed, and its sidebar on the right displays the function's documentation, including the function signature, parameters, filters, and returns.

```
345
346 def plot_deflection_stats(mean_wind, mean_xb, mean_xt,
347                          std_xb, std_xt, folder=None, TI=None):
348     """
349     Plots (1) mean deflections vs. mean wind speed
350     (2) standard deviation of deflections vs. mean wind speed.
351
352     Filters out data where mean_xb and mean_xt are ≤ 0.01.
353
354     Parameters:
355     mean_wind (array-like): Mean wind speeds.
356     mean_xb (array-like): Mean blade deflections.
357     mean_xt (array-like): Mean tower deflections.
358     std_xb (array-like): Standard deviation of blade deflection.
359     std_xt (array-like): Standard deviation of tower deflection.
360
361     Returns:
362     fig, axes: Figure and axes handles.
363     """
364     if folder is None or TI is None:
365         TI = ''
366     else:
367         TI = TI[len(folder)+9:]
368         TI = float(TI)
369
370     # Create figure and subplots
371     fig, axes = plt.subplots(2, 1, figsize=(10, 6))
372
373     # Mean deflections vs. mean wind speed
374     axes[0].plot(mean_wind, mean_xb, 'o', label="Blade Deflection (mean)")
375     axes[0].plot(mean_wind, mean_xt, 's', label="Tower Deflection (mean)")
376     axes[0].set_xlabel("Mean Wind Speed (m/s)")
377     axes[0].set_ylabel("Mean Deflection (m)")
378     axes[0].set_title(f"Mean Deflection vs. Mean Wind Speed (TI = {TI})")
379     axes[0].legend()
380     axes[0].grid(True)
381     axes[0].set_xticks(np.arange(4, 26, 1)) # Set x-axis ticks from 4 to 25
382
```

Docs View sidebar content:

- def plot_deflection_stats(mean_wind, mean_xb, mean_xt, std_xb, std_xt, folder=None, TI=None)
- Plots (1) mean deflections vs. mean wind speed (2) standard deviation of deflections vs. mean wind speed.
- Filters out data where mean_xb and mean_xt are ≤ 0.01.
- Parameters: mean_wind (array-like): Mean wind speeds. mean_xb (array-like): Mean blade deflections. mean_xt (array-like): Mean tower deflections. std_xb (array-like): Standard deviation of blade deflection. std_xt (array-like): Standard deviation of tower deflection.
- Returns
 - fig, axes: Figure and axes handles.
- Full name: codecamp.plot_deflection_stats

Questions that PEP8 answers.

- For time, we will not do this exercise.
- If you want, you can search for the answers in the PEP8 guide: <https://peps.python.org/pep-0008/>.

Set A: BOR % 3 == 0	Set B: BOR % 3 == 1	Set C: BOR % 3 == 2
<ol style="list-style-type: none">1. What should you use instead of <code>if y == None</code>?2. If you have an acronym in camel case, should you capitalize the first letter only or the whole acronym?3. How should you check if a variable equals <code>False</code>?4. What is important about how underscores work with variable names? What happens if I put an underscore after a variable? One before? Two before?5. How many spaces should you use after a period in a comment?	<ol style="list-style-type: none">1. What are the capitalization guidelines for classes, functions, and variables? Give examples.2. How should you check if a path ends in <code>.txt</code>?3. What is the difference between mixed case, camel case, and snake case?4. How should you indicate block comments?5. What are the whitespace rules around operators? Give several examples.	<ol style="list-style-type: none">1. If I need to import <code>os</code>, <code>sys</code>, <code>numpy</code>, <code>matplotlib</code>, and a local module called <code>_utils.py</code>, what does my import block look like?2. How many blank lines go before a function I define at the top of a module? How many after?3. Which of these is correct: <code>x[3:4]</code> or <code>x[3 : 4]</code>?4. Should you use single or double quotes to indicate strings?5. Give examples of a long line of code and how you can break it onto multiple lines.

What if I don't know it all by heart?

- Great. Keep space for more important things, there are tools to help you
-> Linter
- From [9]:

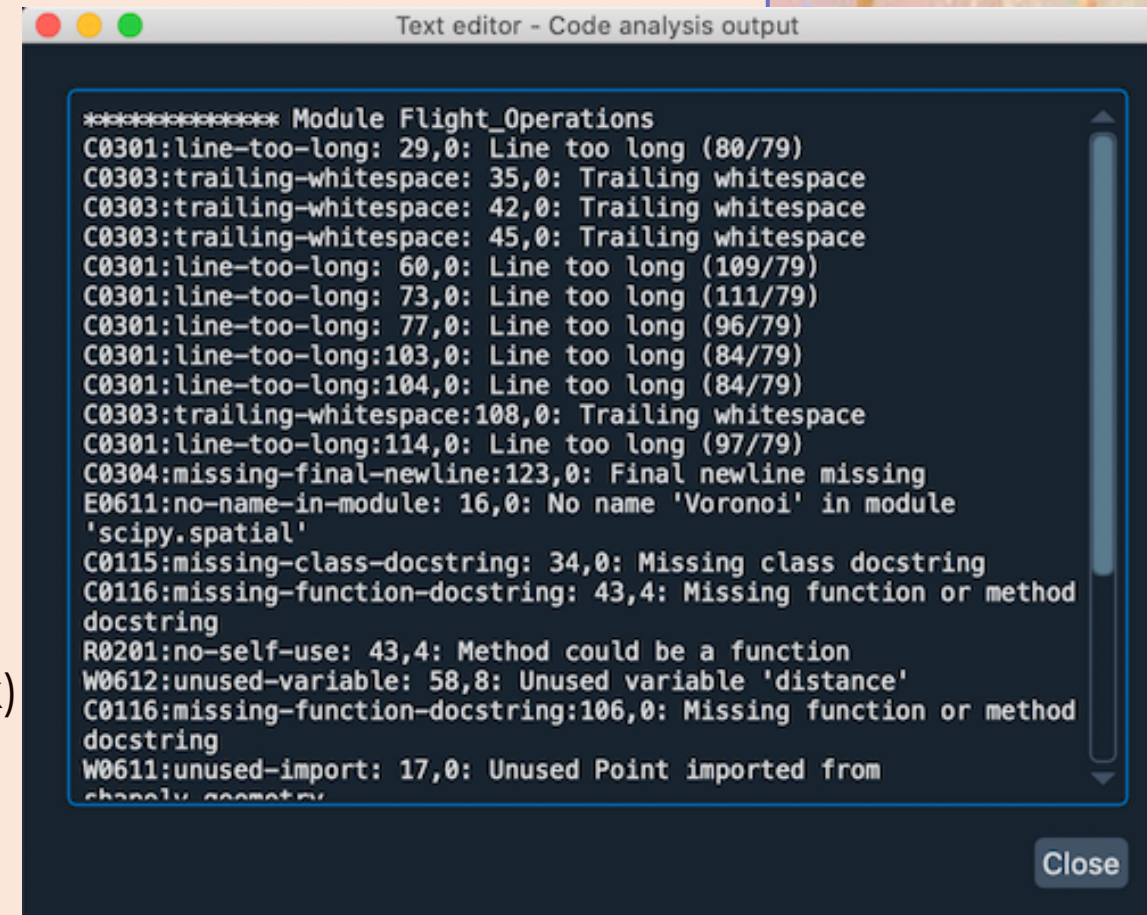
A linter is the most common and widely used of all static code analysis tools. Linters help developers analyze their source code to uncover potential issues, like coding errors, stylistic inconsistencies, bugs, violations of coding standards, and potential security vulnerabilities.

- **Static code analysis:** analysis of code that is done without executing it.
 - Especially important in compiled code, e.g., C.
- In short, a linter is a tool to analyze the quality of your code.



Linters in Python.

- Many different types: pylint (included in VS code, spyder, code errors and style), flake8 (VS Code), pycodestyle (mostly style checks), and more.
 - Comparison is in [11]
- Some linters only analyze PEP8, others (like pylint/flake8) do more.
- We'll use pylint & flake8. mypy for type checking.
 - Integrated in VS code extensions & on PyPI.
 - We'll see how to use it next.
- 1 step further: Auto-formatter (e.g. autopep8, black) in-place reformats your code
- More details on Python linting in [6,7].



```
***** Module Flight_Operations
C0301:line-too-long: 29,0: Line too long (80/79)
C0303:trailing-whitespace: 35,0: Trailing whitespace
C0303:trailing-whitespace: 42,0: Trailing whitespace
C0303:trailing-whitespace: 45,0: Trailing whitespace
C0301:line-too-long: 60,0: Line too long (109/79)
C0301:line-too-long: 73,0: Line too long (111/79)
C0301:line-too-long: 77,0: Line too long (96/79)
C0301:line-too-long:103,0: Line too long (84/79)
C0301:line-too-long:104,0: Line too long (84/79)
C0303:trailing-whitespace:108,0: Trailing whitespace
C0301:line-too-long:114,0: Line too long (97/79)
C0304:missing-final-newline:123,0: Final newline missing
E0611:no-name-in-module: 16,0: No name 'Voronoi' in module
'scipy.spatial'
C0115:missing-class-docstring: 34,0: Missing class docstring
C0116:missing-function-docstring: 43,4: Missing function or method
docstring
R0201:no-self-use: 43,4: Method could be a function
W0612:unused-variable: 58,8: Unused variable 'distance'
C0116:missing-function-docstring:106,0: Missing function or method
docstring
W0611:unused-import: 17,0: Unused Point imported from
channel.geometry
```

Close

Let's use pylint & flake8 together.

- Follow along!
- Pylint and flake8 in VS code.
 - VS code marketplace has pylint, flake8, and mypy [12].
 - Enable both pylint and flake8. It should now run automatically on any open Python code.
- Pylint from the command line.
 - Pylint is available on PyPI. Install it with Anaconda Prompt: `pip install pylint`
 - Call pylint on a file or package: `pylint filename/package`



Break.



Packaging.

Wrap up your code with a bow, give it as a present.



Import functions from modules or packages.

*Should only have
functions/classes in
the package!*

*codecamp/
is a package!*

MODULE

PACKAGE

What it is

A single .py file.

- E.g., turbiefuncs.py.

A folder with a collection of .py files, perhaps with subfolders.

- More details later.

To import from it

Module must be in same folder as main script.

- *(Or you can modify path – not recommended*)*

If not installed:

- Must be in same folder as main script.

If installed:

- Can be imported from any folder.

Example

```
code_folder/  
main_script.py  
codecamp.py, with functions:  
• function load_resp  
• function load_wsp  
• function simulate_turbie
```

```
code_folder/  
main_script.py  
codecamp/  
__init__.py, with functions:  
• function load_resp  
• function load_wsp  
• function simulate_turbie
```



Why packages are nice compared to modules.

- For just you:
 - They can be installed and thus imported anywhere on your computer*.
 - When you have many functions/classes, allows better organization because you can separate them into different files.
- **Disclaimer about modules:*
 - It's possible to also make a module “discoverable” anywhere on your computer.
 - This is done by either updating the `PYTHONPATH` environment variable (global change – don't do it!!!) or updating `sys.path` in each main script (see Appendix).
 - Jenni does not recommend. It (a) makes your code less robust to changes in folder structure and (b) makes your code less shareable due to implicit dependencies. Use (editable) packages!
- For other people:
 - Code is nicely enclosed.
 - You can upload it to PyPI, then your package can be installed via `pip install mypack` (see Appendix).



Anatomy of a package.

- A Python package:
 - Is a folder of .py files with functions/classes meant to be imported.
 - (Optional for Python 3.3+) Often contains a file called `__init__.py`*
 - (Optional) Contains submodules and/or subpackages.
- *The `__init__.py` is not required for “namespace packages”.
 - Namespace packages are new since Python 3.3.
 - We’re going to ignore namespace packages and focus on regular packages. You can read about the namespace packages in [13, 14] (end of slide deck).



Install using a `pyproject.toml` file.

- To make a package installable via `pip`, you need a `pyproject.toml` file in the same folder as your package [15].
 - This is just a text file with a specific format.
 - Disclaimer: some Python projects use an old installation process, which required a file called `setup.py`. Don't be surprised if you see `setup.py` instead of `pyproject.toml`.
- A simple `pyproject.toml` file is in this week's folder.
 - We'll look at it next slide.



See many more settings at <https://packaging.python.org/en/latest/guides/writing-pyproject-toml/#a-full-example>

Simple pyproject.toml

```
pyproject.toml X
C: > Users > rink > git > G-SPP > spp-course-material-OLDGITLAB > week06_packaging_environments > src > pyproject.toml
1  # An extremely simple pyproject.toml
2  # Many more options here: https://packaging.python.org/en/latest/guides/writing-pyproject-toml/
3
4  # How to build your package. "[S]trongly recommended" to include this,
5  # per the docs. We use hatchling, default on Python docs.
6  [build-system]
7  requires = ["hatchling"]
8  build-backend = "hatchling.build"
9
10 # Information on the package
11 [project]
12 name = "examplepackage"
13 description = "An example package"
14 version = "0.0.1"
15 authors = [
16     { name = "I. M. Cool", email = "imcool@noreally.com" }
17 ]
18 dependencies = [
19     "matplotlib",
20     "numpy"
21 ]
22
23 [project.urls]
24 Repository = "https://github.com/me/spam.git"
25
```

name of folder + how to import

what packages need to be present for your package to run (IMPORTANT)

your git repo

Repo structure with a package.

- With a correct `pyproject.toml`, you can use `pip install` to install your package, then use it from anywhere.
- In that case, repo structure might look like this.
- Note that package is nested in `src/` to prevent accidental local imports.

```
data/  
examples/  
|--code_week3.py  
|--...  
|--main.py  
tests/  
|--test_week3.py  
|--...  
src/  
|--codecamp/  
|----__init__.py  
.gitignore  
LICENSE  
pyproject.toml  
README.md
```


A glimpse at what pip install does.

- Many things, which is why it can be so confusing!

- `pip install PACKNAME`

- E.g., `pip install numpy`.
- Looks for package PACKNAME on the online repository, PyPI.
- If found, downloads files for that package into your Python distribution. A place you don't normally see.

- Note: in some installations, pip is not available in your terminal.
 - You can call it using alternative:
`python -m pip install <options>`

- `pip install .`

- Be careful – there is a period at the end!
- Look for `pyproject.toml` in current directory. Period = current directory.
- If found, copies files for the package into your Python distribution.
- Local files are now “dead”.

- `pip install -e .`

- “e” flag = editable installation.
- Adds the package folder to `sys.path`.
- Now you can edit the package files, and you will see updates on import.



Summary of most common pip commands.

`pip install PACKNAME` ← looks for package on PyPI, installs dependencies, downloads package files from internet to Python folder.

`pip install .` ← finds package settings in `pyproject.toml` in current folder, installs dependencies, copies package files from local folder to Python folder.

`pip install -e .` ← finds package settings in `pyproject.toml` in current folder, installs dependencies, adds folder to Python path

Note pip can also directly install from a git repo if git is installed, and much more. See pip docs for more details [16].



Homework for this week.

Clean up your codecamp code.



Last homework with codecamp teams.

Any questions?

PART 1: Make a package

- Turn codecamp/ into a package, i.e., nested in src/ and with proper dependencies in pyproject.toml, move tests and code_week*.py to subfolders.
- Make/activate a new environment called test. Install your codecamp package editably. Run your tests. Run main.py. Do they work?

[Details on GitHub](#)

PART 2: Fix code style

- Enable pylint and flake8 extensions.
- Get your pylint score on your codecamp package above a 7.



References.

1. Summary of PEP8, <https://vuyisile.com/a-summary-of-pep-8-style-guide-for-python-code-and-pep-257-docstring-conventions/>
2. PEP8 – Style Guide for Python Code, official documentation, <https://peps.python.org/pep-0008/>
3. PEP257 – Docstring Conventions, official documentation, <https://peps.python.org/pep-0257/>
4. Jasmine Finer, How to Write Beautiful Python Code With PEP 8, <https://realpython.com/python-pep8/>
5. The Naming Convention Foundation, python Naming Convention, <https://github.com/naming-convention/naming-convention-guides/tree/master/python>
6. Parisa Gregg & Myles Mitchell, Styling your Python code: An introduction to linting and formatting <https://www.jumpingrivers.com/blog/python-linting-guide/>
7. Essential Python Tools, <https://books.agiliq.com/projects/essential-python-tools/en/latest/linters.html>
8. CodeCamp final project [Final project — CodeCamp 2023 documentation \(dtu.dk\)](#)
9. What is a Linter? [What Are Linters? \(+ Why Your Team Should Use Them\) \(codacy.com\)](#)
10. A Deep Dive into Static Code Analysis Tools [A Deep Dive into Static Code Analysis Tools \(codacy.com\)](#)
11. Comparison of Python linters [Linters and formatters — Essential Python Tools 3.7 documentation \(agiliq.com\)](#)
12. Linting in VS code [Linting Python in Visual Studio Code](#)



References continued.

13. Tutorial on packages, implicit namespace package [Python Modules and Packages – An Introduction – Real Python](#)
14. Namespace package and what they're good for [What's a Python Namespace Package, and What's It For? – Real Python](#)
15. Pyproject.toml [Writing your pyproject.toml - Python Packaging User Guide](#)
16. Pip install docs [pip install - pip documentation v24.0 \(pypa.io\)](#)
17. Generating distribution archives and uploading them to TestPyPI [Packaging Python Projects - Python Packaging User Guide](#)



How to upload your package to PyPI.

- This will allow a user to install your package with `pip install <your_package_name>`, without needing git.
- Excellent explanation and steps here on the Python Packaging User Guide:
 - [Packaging Python Projects - Python Packaging User Guide](#)



What did you like about CodeCamp? (1/2)

- I really enjoyed how the project was organized before we got it. Especially the tests were an excellent way of understanding what the problem was. Even though we used chatgpt extensively, we still learned a lot of new about how to write scripts and how to connect them into a project. I have always had a deep disgust for computers and everything that had to do with coding but I really this project. My favorite was documenting and creating coherent designs for markdown files(can be this be used in order to get paid somewhere haha?)
- I think the workload division and the way that the code camp project progresses helps to get comfortable with the necessary programming and git hub.
- Learning a lot. Clear explanation to directly strat into the tasks. Thats very good.
- I liked that the homework lead up to the project.
- How fun the project was overall, loved the fact that we're learning on a base of a wind turbine
- The coding process was very given and I personally learned a lot, and expecially the feedback waas great. After the feed back I have the motivation to improve the coding.
- The detailed instructions and peer feedback
- The setup in itself was fine regarding the wind turbine, but the best was that it really improved my understanding of working in a group with github (PR, merges, repo etc).
- Group work was nice and gave me good insights on coding
- I liked that it was on how we could make the codes
- We divided the group work among ourselves, so I didn't have the opportunity to do all the exercises, knowing that I likely missed out on some skills that others gained while working on their tasks.
- I think some of the task descriptions were a bit unclear and alternated the outcome and workload of the code.
- That it is based on homeworks from previous weeks, so the workload is not overwhelming.
- It was a fun task, good group size
- It is fun tasks, and there is a lot of possibilities to get help if you need it
- It is a really nice way of putting into practice the concepts learned so far (readme, git etc), and to see ourselves how important this is, when trying to run other team's repos.
- I liked that we were doing practical tasks using realistic data to show useful results
- The ability to have to think for ourselves in using the functions we have built over the past few weeks. Pseudo coding!!
- I liked the way you structured the project. It is great for learning that we write all of the functions ourselves and that we get to do it "slowly" instead of releasing the entire project in week 3.
- It has used alle the knowlodge from the weekly homeworks
- How allmosy all of the stuff is explained
- Learning about git hub, very useful
- Co-work with others, getting feedbacks, improving my skills in git things-
- I think the project was well structured, giving a set of functions to complete each week, which forces those who are good at python to wait, and those who are not as good are able to keep up.



What did you like about CodeCamp? (2/2)

- Very straightforward, almost clear instructions and the course is well structured so it is easy to keep up
- I enjoyed working in my group.
- I really liked the GitHub integration. It is the first time that I have actually made a collaborative coding project where it doesn't just consist of sending versions of the same code over messaging apps to each other.
- The way the whole teaching and homeworks are structured make it very easy to follow and understand everything. The course could easily be very hard to follow and too complicated but to the detailed structuring.
- I think it's really good for learning how to code and understanding different things about python. I think I've really learned a lot.
- the interactive approach, working with groups. getting taught the 1 o 1 of git is really helpful
- I like that it connects the material from all the previous weeks. It is nice that the pytest functions are able to evaluate our code so we know if our functions work.
- I liked that we gradually built on it week by week, and we got the completed project after some weeks, which gave us the chance to learn a lot and fix a lot of errors.
- I felt that it was an opportunity to seriously improved my programming (Python and Git) skills, and being able to develop it week by week (with feedback inbetween) was really helpful! Even though I do not have much knowledge in wind turbine analysis, it was not really necessary and was certainly a fun introduction!
- -
- I really enjoyed the CodeCamp project, it was highly interactive, and I feel like I learned a lot about problem-solving, both independently and in collaboration with my teammates.
- That by giving advice/corrections to other teams you actually learn a lot yourself.
- I loved the introduction to git, super useful! I would love to spend more time learning good practices for git (spend more time on gitignore, gitkeep etc..). Overall I am super happy
- Develop skills working with git hub
- It gave an opportunity to explore how various functions can be used in Python, alongside also helped with Git.
- Cooperating with others and learning how to use github and push/pull
- it is a small but complete project
- Great to get accustomed to model engineering problems in wind energy. Gave food for thought.
- It was great that a lot of the code was done previously so you had more time to refine the code, and figure out how to work with your team/in github.
- I liked that the CodeCamp project provided a hands-on approach to learning, allowing me to apply theoretical concepts to practical problems. It helped me improve my coding skills, particularly in numerical modeling and data visualization, while also reinforcing my understanding of wind energy systems. The structured tasks and debugging challenges made the learning process engaging and insightful.
- that things make sense, but it's a little bit difficult to understand python , personally speaking, as a person that has never done programming in that level.
- Taking all the functions i created with my partner and create a whole program



What would you like to see improved with the CodeCamp project? (1/2)

- Because of my limited prior knowledge and the fact that we never tried the extra credit questions i dont think i can provide any feedback on improviements.
- Some sections about the codecamp project were a bit vague so it at times was a bit confusing what was expected. *which ones?*
- Maybe a bit more explanation deeper on github but maybe thats better later in the course.
- Maybe some more live coding *ok*
- It works well.
- The CodeCamp is well explained, but it took a bit of time to completely understand the outcome of the assignment, and we had to ask three or four times to get what we needed to do precisely. This wasn't that big of an issue, just a bit clearer explanation, to reduce people aksing. *yes, we need to present it better*
- Maybe make it a normal course with grades as this can motivate people more and it would allow a less strict setup. *strict on tests or on project grading?*
- Perhaps even more of working without specific instruction, like the last part. *this comes in the 2nd half ☺*
- Maybe less instruction to follow, sometimes it's more the time spent at checking that everything is done correctly than the actual process of learning *hard to balance this & debugging*
- It would be nice to have two projects, one for people that are new to python and one for people that have already some experience
- Round Robin feedback sessions are a bit hectic, I find it difficult to focus there due to noise. Also I feel that I need more time to read others' code to really think along.
- The evaluation today was suuuuper stressful. Would be nice to get a deadline in a few days, especially if you need to run code 3x taking up to 10 minutes. *very good point. let's fix that.*
- Some of the assignments was a bit hard to understand, because it was unclear what the functions we made should do
- Maybe make all tasks available at the start so we can work through at our own pace but have those same wk3 and 4 checkpoints each week
- Sometimes it was a bit difficult to make sure everyone had fair commits and pull requests... we did a lot of the work sitting at the same computer, so it just looked like one person did it. *yea this is a tricky one we're still figuring out*
- Nothing really, I think the project is great as is
- Maybe a resubmission period for those who don't pass the project. *we have this*
- The main.py directives regarding the final graph
- Skills on call functions, code cleaning... *???*
- Some additional (optional) difficulty, perhaps in terms of best practices like, try/excepting, checking for file existing, making everything dynamic, add other data files that have "messy" data, that have to be handled/cleaned/etc. *ooh, thanks!*
- Office hours was not really helpful *hm, why not? timing, communication between instructions?*
- A bit lost at the start. I thought that the time to give feedback was a bit rushed for no reason.
- Maybe the coding part itself could be slightly more challenging. I know that the project is open for additional tasks, but it still seems quite limited to me because of the difference in how much Python people have written before.
- Keep it up. 🍌

What would you like to see improved with the CodeCamp project? (2/2)

- I don't think it was known that we should be using pull requests and merging until a little later in the project so maybe knowing that a little earlier would have been nice. Also a little crash course on how to use pull requests would have been good.
yep, my bad. sorry!
- The peer review process was messy/chaotic. I would have preferred to have more time to properly evaluate/dive into other peoples code and see how it differs from ours.
- I'd like some more straightforward and detailed instructions for github, since I am a beginner with no prior experience and the errors and conflicts caused me a lot of stress about failing the course.
besides the videos in week 1 and the homework, what else could we provide?
- Better guidance for coding beginners *yes, this is something we need to figure out how to address in the future*
- While for the previous weeks homework, the instructions on how to develop the code were quite clear, I felt that the instructions for the last week were a bit thinner, which initially led to a bit of apprehension on what was actually asked and involved some intergroups discussion. But I guess this is all part of the fun :) and to be fair, the evaluation criteria was thoroughly explained. Also, help from the teaching staff on Slack was constant and very helpful! Thanks for that.
- -
- One possible improvement could be providing more support material for beginners (Like a function description file for example). While self-study is always an option, having structured resources available would be helpful. Other than that, I don't see much that needs improvement. The course is amazing.
- Evaluating 3 teams in 1-1.5 hour is too tight on time. If increasing the time from 1-1.5 hour (approx. from 9.10 to 10.30) to maybe 2-2.5 is not possible I would suggest that the number of teams under evaluation per team should decrease from 3 to 2. Keep in mind that the code also needed some time to run, therefore the whole evaluation thing was too tight regarding time management and I think I could have given better evaluation with more time *agreed, we will change the eval procedure*
- Nothing
- An advanced level
- A bit more explained material for the requirements of the project.
- More functions regarding wind programming
- a bit more clear what we need to do and easy to find
- I would like to see more flexibility in the grading or assessment of answers, as they felt quite rigid. While precision is essential in engineering projects, allowing a small margin of error (around 5 to 10%) would better reflect real-world scenarios where minor variations in calculations are expected. This would encourage a deeper understanding rather than just focusing on getting an exact match.
I assume you are referring to the tests? While I see your point, in my opinion the tests are actually reflective of what you would see in real-world software. So having something fail due to tolerances is really beneficial learning, because it will come up in your own code later. If everything was just ballpark, it actually can allow sneaky bugs through, so it's better if it's strict.
- myself. I want to learn more things.
- The code being more straightforward and easy to understand
which part? i'm afraid this is too vague for me to be able to take any action



Do you have any other general feedback?

- maybe a small portion of a lecture the magnificent things you can do with .md files could be included. Then if a student likes it they can find the rest on the internet. Also a few dynamic examples of split screening gitbash and a folder. When the teacher checks out a branch the file or folder content would change. (This is how i started to understand branches) Finally maybe some examples of using only the terminal to run python files or other more cool stuff
- I think it may be nice to, in the beginning of the course, do a quick run down of do's and don't with git and have a file for such information which can be used by students throughout the course as a reference
- No
- The round robin and peer review feedback sometimes felt a bit to long. I like the feedback but just shorter time.
- Really liked learning about Git. It teaches well about working as a group
- Good that you offer online lectures and they are recorded.
- I think the course is very well structured and organized. I like that the teacher takes the feedbacks very seriously
- Very fun and good course
- It is a bit unclear how the git workflow is supposed to be. Like when to create a branch, when to merge it, those kind of things.
- No
- I think a little MCQ on learn would be beneficial for us to "constantly" verify that we are remembering the correct calls. E.g. we use add, commit, and push all the time; but sometimes we need to add 'upstream', sometimes we need to '--set upstream -origin' or something like that. Sometimes we need to fetch the branches from the remote with 'git fetch 'origin''. I think it would be nice to have a MCQ that tests and trains our knowledge/use of these commands. Clearly, it shouldn't be graded, that would ruin the purpose. I think it's a little too harsh this whole "you fail if you get this one thing wrong". I think it would be better to change the course to be a graded course then, if that's the issue.
- Introducing the project was too fast but all of it was in the pdf.
- No
- I dislike the mix of online/physical, either go fully remote or fully physical. It makes discussing feedback and round robin more valuable imo, than the mix. Especially since it is difficult to find a room to have a zoom meeting in, and then meet back in class.
- Not really
- I took this course because I wanted to learn python. I feel like for people with my profile it might be a bit fast pace but it is forcing me to spend a lot of time to learn. I would suggest giving more support. Maybe explaining more during class or making a cheat sheet document for starter python students. I personally got overwhelmed with the amount of information at first and it would be nice to have a place that you can look towards that you trust.
- Maybe the feedback in break-out-rooms every week is a bit much. Try to stress even more that when giving peer review (specifically for the end of the CodeCamp) your own grade won't increase by rating other projects lower, because it is slightly frustrating to get negative feedback unnecessarily.
- I like the course and I am enjoying it a lot.
- I think that by making the course hybrid, it effectively makes the experience a little worse for students who show up. Maybe instead of assigned breakout rooms, physical students could team up separately from online students, so there can be in person discussions



Do you have any other general feedback?

- Generally I'd like to have a bit more help and guidance during office hours, especially on the days close to deadlines, since we needed some specific answers about our project.
- It might not be for new programmers. Python basics are not explained, so if you have never used it you will have to work a lot by yourself. As the description of the course mentioned I thought it was for new programmers, but I am not so sure now. I would say it for people with some knowledge to improve their coding.
- I think the course is quite challenging for beginners, taking into account that is supposed to be for users with no prior experience in coding. Maybe it should be more smoothly upscaled to complete the project. More help and explanations could be provided for inexperienced users. AI tools were mostly the guidance even if for the basics.
- I am having fun and, while I am putting in more hours than I initially expected (I believe due to some rustiness in programming) I feel motivated and I am enjoying this course a lot, so keep up the good work!
- -
- No
- In general, the course is amazing.
- No, loving the course. I am not a wind engineer but rather an energy engineer, so I struggled a bit on the equation part of codecamp, but all the rest was straightforward.
- I like the class, I think it should become into a grade 10 class
- The course is good and handled very well
- I've really appreciated the help from Professor Jenny throughout the course. She has been incredibly supportive, always offering clear explanations and resolving any issues I had with both the code and Git. Her positive attitude and great smile made the learning experience even better."
- I love how the professors are. eager to help you and very kind and friendly. My personal issue is that I can handle learning as much as I would like to learn, due to my heavy schedule (other courses + job + personal life). So I think that we are running a lot for a person like me.
- Some more examples on how to create functions to work to beginner coders

