Welcome.

Slides this week on Learn / Content / Miscellaneous.

All other weeks, will be pushed to course GitHub. ;)

Physical students:

- **Sit with your P0 Team**. Seating chart on walls.
- Turn off laptop volume (mute). ←IMPORTANT!
- Log into the Zoom meeting.
  - Link is on Learn webpage, under “PiWE Links” on main page.
- Double-check your microphone is muted. You can keep your camera off.

Please note!

- By attending this class, you consent to being recorded.

- This recording will be placed in an online folder accessible to the students of this class. It may also be distributed to other DTU students for education purposes.

Let’s pull new course material.

- Live coding, please follow along.
- If you have not forked/cloned the course repo in advance:
  - Use this time to complete Steps 3, 4, and 5 on the pre-class preparation. See week00_prep on GitHub.
- Slides are on Learn if the pull doesn’t work.

- Physical students:
  - After you successfully pulled, try to help your neighbors/groupmates if they have issues. Flag TA/teacher as needed.
- Virtual students:
  - Use Reza and the Zoom chat if you have issues.

Instructions for reference.

- Open a git terminal and…
  - Navigate to the 46120-PiWE repo you cloned before lecture.
    - ```
      cd <path to 46120-PiWE repo>
      ```
    - Example on git-scm Windows: `cd /c/Users/rink/git/G-PiWE/46120-PiWE`
  - Define a URL called "upstream" that points to the original repo on GitHub (not your fork):
    - ```
      git remote add upstream https://github.com/DTUWindEducation/46120-PiWE.git
      ```
  - Pull changes from the upstream repository:
    - ```
      git pull upstream main
      ```
  - You should now have a subfolder "week01" with the course slides in your local folder.


      ```

46120: Scientific Programming for Wind Energy

Course Introduction

Jenni Rinker
Ju Feng

Agenda for today.

- Pull new course material ✔

- Meet the 46120 Teaching Team.
- Course introduction: Jenni.
- What is good code: Ju Feng.
- Teaser for git/GitHub: Jenni.

- Begin groupwork on Week 1 homework.

Meet the 46120 Teaching Team

Jenni Rinker.

- Associate Professor in the REC section.
  - Expertise in loads, aeroelasticity, turbulence-turbine interaction, etc. Also into programming for scientists/engineers. 🤓
- My Python history.
  - 2007: First learned in introduction to computer science.
  - 2012: Switched from Matlab during my PhD.
  - 2017: Attended the Summer School in Advanced Scientific Python Programming (ASPP).
  - 2018: Became a lecturer for ASPP.
  - 2021: Started CodeCamp (SPP/PiWE) with Antonio.
- I do not consider myself a power user.

Ju Feng.

- Senior Researcher in the SYS section.
  - Expertise in wind farm optimization and control, wake modeling, etc.
- My Python history.
  - 2014: Took online courses in Python programming

Introduction to Computational Thinking and Data Science (offered by MITx @edX)
| Completed with final grade 97% and obtained verified certificate on May 8th, 2014

Introduction to Computer Science and Programming Using Python (offered by MITx @edX)
| Completed with final grade 95% and obtained honor code certificate on January 10th, 2014

  - 2015 to Now: Use Python in multiple projects, like FarmOpt, ModFarm, PyWake/TopFarm
  - 2023-2024: SPP2023/2024

Reza Manami.

- PhD Student in the MES section.
  - Working on pulsed lidar for measuring turbulence.
- My Python history:
  - 2023: Data analysis during my internship at Airparif (France)
  - 2022: Using Python for course projects and research at LMD (France)

- *Certainly, I may be less experienced than others, but I'll do my best to help you!*

Nicolas Gonzalez.

- PhD student in the RAM section.
  - Expertise in mesoscale modeling
  - Project: Wake modeling for large-scale wind farm cluster optimization.
- My Python history.
  - 2017: Switch from R during my BSc. thesis
  - 2018: Worked in data analysis with Python for 2 years
  - 2020: Teaching Assistant in “Introduction to Programming” at DTU
  - 2023: Worked with PyWake, TopFarm and PyWAsP for different projects at DTU Wind for 2 years.

Who are you and why are you here.

- Physical students: talk in groups of 3-4.
- Virtual students: in a minute, turn on your video and self-navigate to a breakout room (BOR). Try to have 3-4 people in each BOR.
- Introduce yourself:
    - Name
    - Program
    - Where you’re from
    - Why you’re here
    - What you most want to learn in this course
- After a few minutes, we’ll close the BORs and come back to the main session.

Course format, expectations, etc.

(a.k.a., the boring stuff)

Our philosophy.

Engineering is collaborative problem solving.

- Writing code that "works" is not enough.
- Need to be able to design and write code *in teams* and *for other people*.
- Thus, emphasis on version control, code reviews, communication skills.

Be ready for some growing pains.

- From 18 students in 2023, to 32 in 2024, to **almost 80** in 2025.
- Some course changes: switched from GitLab to GitHub, added a course Slack chat, and are trying GitHub classroom for the first time.
- We appreciate your patience if there is some…

Hybrid-classroom culture. (1/2)

- Physical or virtual, with your team or solo: it’s up to you.
  - Thursday lecture and team brunch at someone’s house? 🥞 👌
- Our objective is to provide the same quality teaching to virtual and physical students.
- During lecture, all students log onto Zoom.
  - We use BORs to connect physical and virtual students during breakout activities.
  - Physical students may need to shuffle around to reduce noise interference. We will minimize how often you do this.
  - Please bring headphones to class! If you have a meeting speaker, that too.

Hybrid-classroom culture. (2/2)

- During lecture, feel free to post questions in the Zoom chat.
    - Our TAs are monitoring, will either respond in-chat or ask the lecturer.
    - React to someone else's question if you have the same question!

- Video and microphone. During…
    - Monologuing, boring lectures: video off or on. Laptop sound off for physical students.
    - Breakout activities: video on, if bandwidth allows.
    - Microphone: always off unless speaking.

Course format.

- The course is **student-driven**.
  - Relatively little time of classic “lectures”. (Today is an exception.)
  - Our goal is to explain enough that you can Google it and teach yourself.
  - In this sense, it is very easy to fail this course.

- Weekly schedule.
  - Thursday mornings we meet all together. Virtual or physical. 09.00 unless we announce otherwise.
  - On another day, you meet with your group.
  - Office hours probably on Mondays or Tuesdays. TBC.

Learning objectives.

- Utilize Python packages common to wind energy, such as **numpy**, **matplotlib**, **scipy**, PyWake, TOPFARM, xarray, **pandas**, etc.
- Manipulate data stored in the most common wind-energy formats, demonstrating skills such as **loading from/saving to file**, performing computations, and **visualizing** results.
- **Design and publish a Python package** for a wind-energy application with **tests** and **documentation**.
- **Collaborate with a team on a code base** hosted on GitHub, demonstrating basic git skills such as adding, committing, pushing, and branching.
- Develop, debug, and lint code using VS Code and related extensions.
- Communicate code orally and also in writing, via diagrams, comments, commit messages, and documentation.
- Critically analyze code for good coding practices such as modularity, maintainability, adherence to stylistic conventions, etc.
- **Write tests** for scientific code.
- Execute code on a **computing cluster**.

How you're evaluated.

- Formative evaluation.
  - Weekly group assignments we discuss in class. Peer feedback.
  - Feedback from your peers and from the instructors.
- Two group programming projects.
  - "Handed in" as GitHub repositories.
  - More details on next slide.
- Short quiz on git basics in Week 13.

- Final grade: cumulative evaluation of individual contributions to final project and individual quiz.

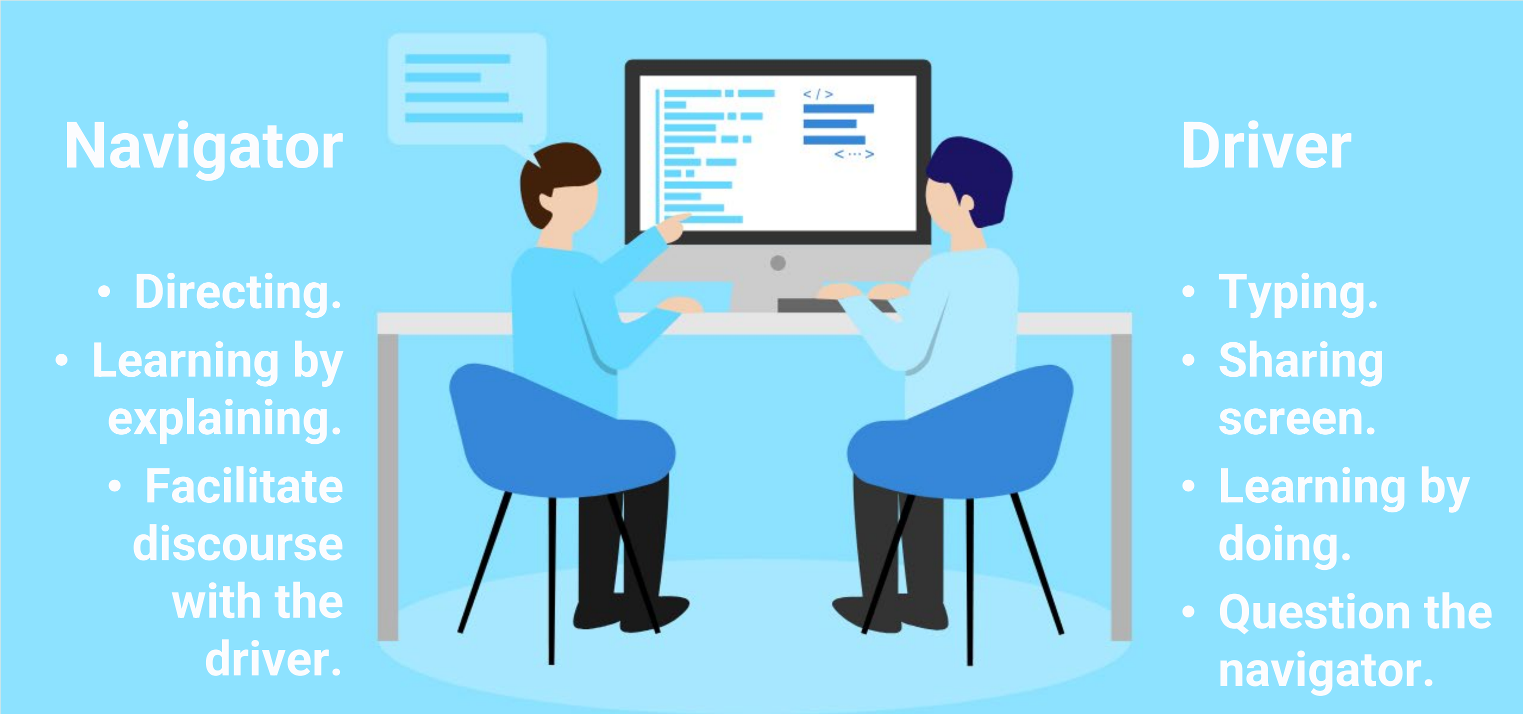
Group programming projects.

Note: Weeks 1 & 2, you work in pre-assigned "P0 Team", today's seatmates.

1. **CodeCamp final project**. Turbie.
   - Due March 12.
   - Groups of 2 -3 people that you choose.
   - Pass/fail plus qualitative feedback.
   - Must pass to submit a final project.
2. **Final project**. Design/make/publish a Python package.
   - Due May 7.
   - Groups of 2 to 3 that you choose. May work solo if granted special permission.
   - Project must be based on one of the provided templates unless granted special permission.

Coding in groups: pair programming.

- A method to accomplish a coding task, while exchanging coding knowledge with a partner.
  - Frequently used in software companies and other software development situations.



Platforms and getting help.

GitHub: 46120-PiWE

- Course information
- Slides
- Homework & project requirements

GitHub: else

- Team repo for Weeks 1 & 2 (P0 Team)
- Team repo for CodeCamp
- Team repo for final project

Course Slack: 46120-2025

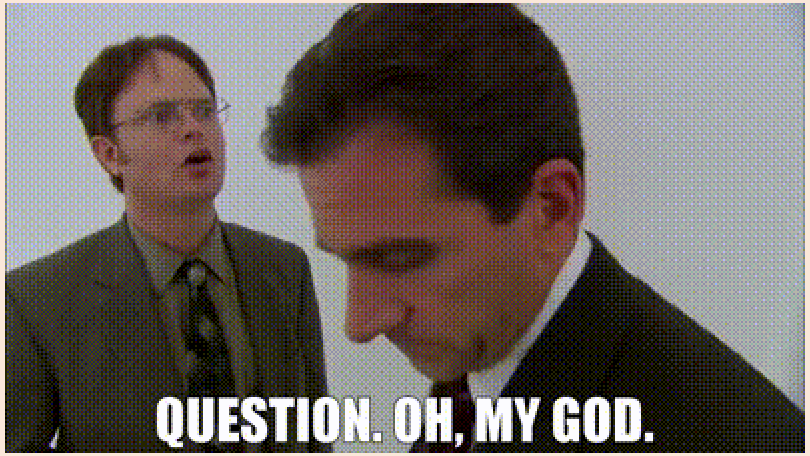
- Chat/meeting software.
- Ask questions outside of class.
- Can make a channel for your team(s).
- Post cat gifs or arrange class get-togethers (in #social).

Learn

- Sensitive information, such as...
- Links to GitHub assignments.
- Sign-up sheets for groups.

Questions?

Post in chat or ask verbally. ☺

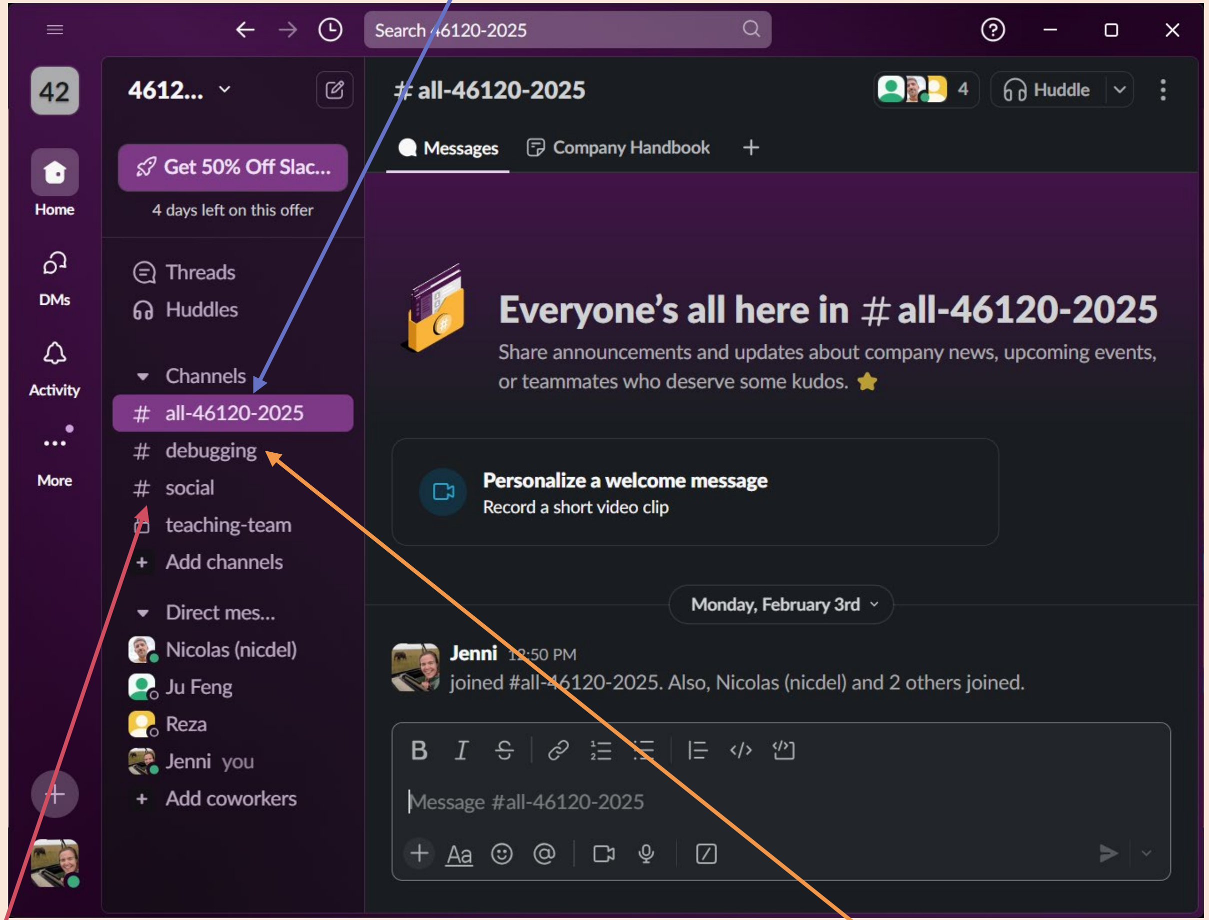


Let’s take a break. ☺

During the break, consider joining the class Slack!

- Link is on Learn page, top right corner. “Invite link to Slackchat”.
- Can access slack from the web or download the app.
- USE YOUR **DTU EMAIL** if prompted!

#all: general questions



#social: for memes, inspiration, arranging LAN parties, etc.

#debugging: when things aren’t working

What makes code “good”?

Ask AI for the following question:

> What is a good program? Please also write a simple python function to calculate the factorial of a non-negative integer as an example. Thanks!

$$
\begin{aligned}
n! &= n \times (n-1) \times (n-2) \times (n-3) \times \cdots \times 3 \times 2 \times 1 \\
&= n \times (n-1)!
\end{aligned}
$$

ChatGPT

deepseek

ChatGPT

A good program is one that is well-written, efficient, easy to maintain, and solves the problem it was designed for.

Key characteristics of a good program include:

1. **Clarity**: The code should be easy to read and understand.
2. **Efficiency**: The program should make optimal use of resources (time, memory).
3. **Modularity**: It should be broken down into manageable pieces (functions, classes) to make it easier to maintain and test.
4. **Scalability**: It should be able to handle increasing input sizes or more complex operations without major rework.
5. **Error handling**: It should be robust and handle unexpected inputs or errors gracefully.

ChatGPT

```python
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Example usage:
print(factorial(5))  # Output: 120
```

A good program is one that is:

1. **Correct**: It produces the correct output for all valid inputs.
2. **Efficient**: It uses resources (time and space) optimally.
3. **Readable**: The code is easy to understand and maintain.
4. **Modular**: It is broken down into smaller, reusable components.
5. **Robust**: It handles errors and edge cases gracefully.
6. **Well-documented**: It includes comments and documentation to explain its functionality.

deepseek

```python
def factorial(n):
    """
    Calculate the factorial of a non-negative integer n.

    Parameters:
    n (int): A non-negative integer.

    Returns:
    int: The factorial of n.

    Raises:
    ValueError: If n is a negative integer.
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result

# Example usage:
print(factorial(5))  # Output: 120
```

ChatGPT

```python
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Example usage:
print(factorial(5))  # Output: 120
```

What is a good program?

My answer:

A good program should

1. Do the job

2. Be easy to understand

3. Be easy to change

On the usage of AI

Code from Scratch to Think Like a Programmer !

We think:

- Writing code to solve complex scientific and engineering problems is an essential skill.
- Programming is foundational for building other digital capacities.
- AI (LLM) is a powerful tool/assistant for writing code, but over-using it is not effective for learning programming.
- Writing code from scratch remains invaluable for learning programming and developing computational thinking skills.