

# Welcome.

*Slides this week on Learn /  
Content / Miscellaneous.*

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

## Physical students:

- Take a number from the TA: 1, 2 or 3.
- Sit in a group of 3, where all three numbers are represented.
  - You can form a group of two if needed.

## Online students:

- Make sure mic is off. Turn video off when lecture begins.
- Feel free to send a morning greeting in the Zoom chat. 🙌



# 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.

**LIVE**



# Let's pull new course material.

- Live coding. Please follow along.
- If you have not cloned the course repo in advance:
  - Use this time to complete Steps 4, 5, and 6 on the pre-class preparation. See week00\_prep on GitHub (link on Learn).
- If this doesn't work for you: slides are on Learn. Contact TA/instructor at end of lecture for help.
- **Physical students:**
  - After you successfully pulled, try to help your neighbors/groupmates if they have issues. Flag TA/teacher as needed.
- **Virtual students:**
  - Use online TA 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 Windows Terminal: `cd C:\Users\rink\git\G-PiWE\46120-PiWE`
    - Example on git-scm Windows: `cd /c/Users/rink/git/G-PiWE/46120-PiWE`
  - Pull changes from the upstream repository:
    - `git pull origin main`
  - You should now have a subfolder “week01” with the course slides in your local folder.
  - If it doesn’t work, contact TA/instructor at end of lecture.

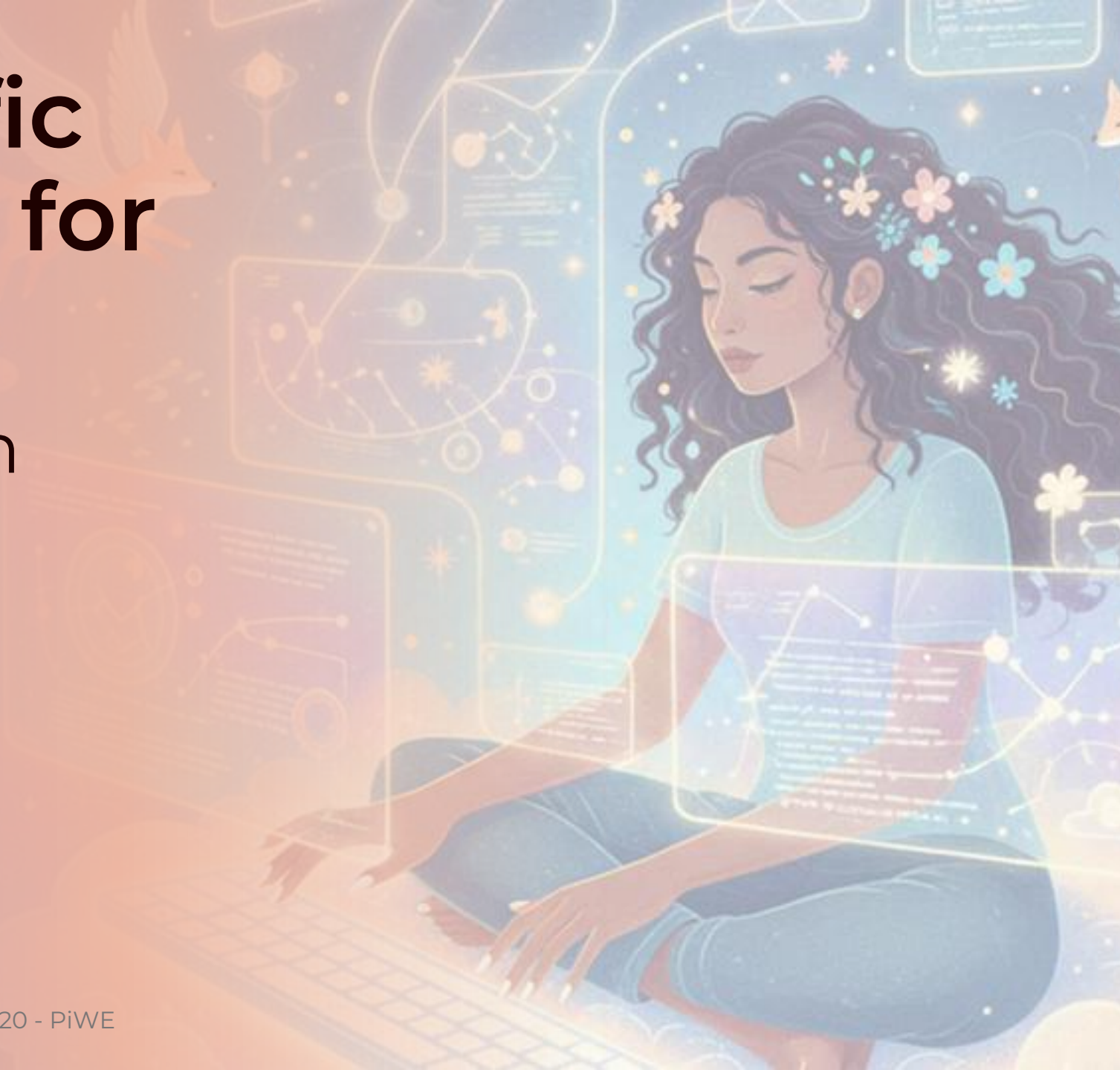




# 46120: Scientific Programming for Wind Energy

## Course Introduction

Jenni Rinker  
Ju Feng



# Discuss week00 solutions.

- Physical students: Talk in your groups.
- Virtual students: Screenshot this slide or open file locally. Accept the breakout room (BOR) you are assigned.
- In your groups/BORs:
  - Pick one of the exercises and present your solutions to each other. Compare different solutions and discuss what process you used to write the code.
  - Repeat for at least 2 more exercises.
- After a few minutes, we'll close the BORs and come back to the main session.

*NB: You will need to work in groups of 2 to 3 for this week's homework. There are 3 sets of "popcorn" groups today for you to meet other students.*



# Agenda for today.

- Pull new course material ✓
- Discuss week00 solutions ✓
- 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 RSP section.
  - Loads, aeroelasticity, turbulence-turbine interaction, etc. Also like 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.



**Ask me about:**  
Castells (“human towers”)

# 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

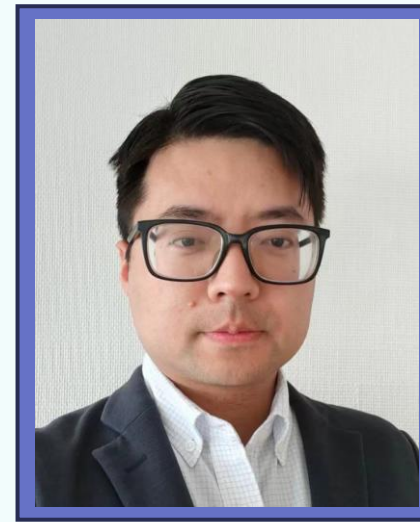
Introduction to Computational Thinking and Data Science (offered by MITx @edX)

| 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
- 2023-2025: PiWE 2023/2024/2025



**Ask me about:**  
*Cooking Chinese dish in DK*

# Bjarke Spangsberg Bak.

- Online TA.
- Industrial PhD Student in the DES section/Konstant Net
  - Timeseries, load forecasting, grid planning, tariff analysis
- My Python history:
  - Pre-2020: MATLAB for physics
  - 2020: Tumor modelling in Python during my math/phys MSc thesis
  - 2022: Detour into SQL in industry
  - 2024: Back to Python for PhD



**Ask me about:**  
*Tree felling*



# Sangwon Lee.



- Physical TA.
- PhD student in the RSP (Turbine Response) section.
  - Project: Innovative rotor-nacelle system design and analysis for modularized blade systems.
- My Python history.
  - 2022-2024: Data analysis and plotting during my master thesis
  - 2025: Teaching Assistant in “Wind Turbine Technology and Aerodynamics” at DTU

***Ask me about:***  
***Marine Engineering***





# Rongxin Wang.

- Physical TA.
- PhD student in the SYS section.
  - Project: AI augmented design optimization for wind farms.
- My Python history.
  - 2017: Started from C++, then switched to Python
  - 2019-2023: Used Python for data analysis for my thesis and internship
  - 2025-: Teaching Assistant in “Scientific Programming” at DTU, and learning and doing ML/AI in my PhD project.



**Ask me about:**  
*Supernovae, general  
aviation*

# Who are you and why are you here.

- Physical students: Talk in your groups.
- Virtual students: Screenshot this slide or open it locally. Accept the breakout room (BOR) you are assigned.
- In your groups/BORs: introduce yourself.
  - Name
  - Program
  - Where you're from
  - Why you're here
  - What you most want to learn in this course
  - Something you do outside of your MSc
- 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.

- 18 students in 2023, to 32 in 2024, 75 in 2025, almost 130 in 2026.
- Scaling up to 130 students, with diverse backgrounds. Multiple physical classrooms.
- We appreciate your patience if there is some...



# (Hybrid) Classroom culture.

- Let's set expectations.
- [46120-PiWE/AttendanceExpectations.md at main · DTUWind-46120-2026/46120-PiWE](#)



# 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/ChatGPT 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. TBD.



# 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.
  - Individual contributions evaluated through commits, participation in discussions on Pull Requests, etc.



# Group programming projects.

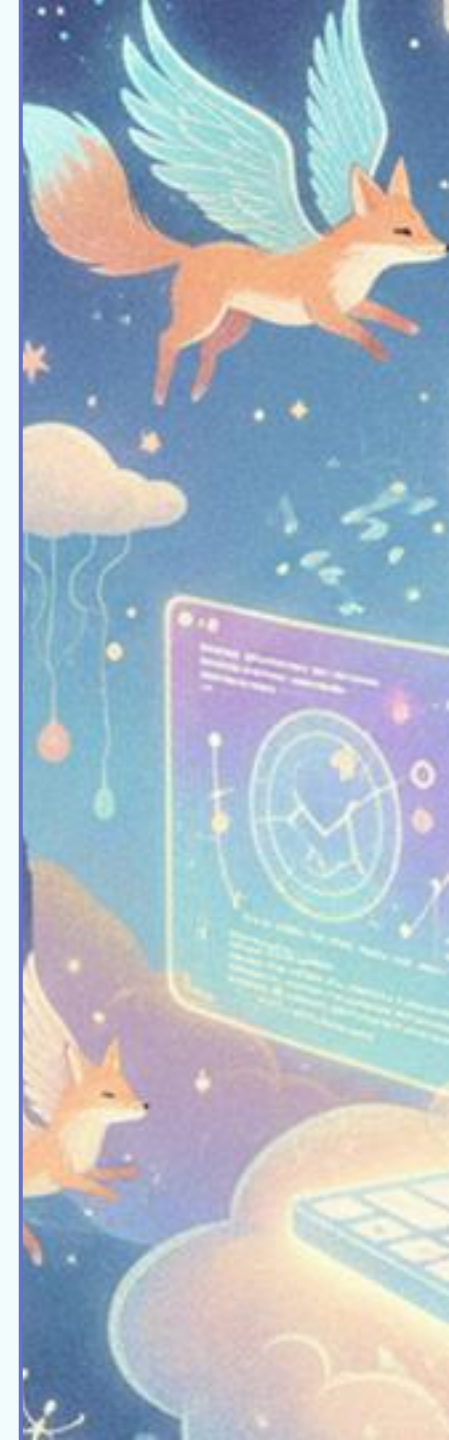
*Note: Weeks 1 & 2, you work in “P0 Team”, which you choose today.*

## 1. CodeCamp final project. Turbie.

- Due March 9.
- 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 4.
- 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.



# GenAI in this course.

- From Melisa et al. (2025):

*“...ChatGPT **can enhance students' critical thinking** by facilitating quick access to diverse perspectives, aiding information analysis, and supporting argument construction. However, **over-reliance on AI can hinder students' motivation for self-reflection and critical evaluation**, raising concerns about academic integrity and the verification of AI-generated content.*

- I.e., GenAI can help your critical thinking and learning when used well.
- Use GenAI chat (e.g., ChatGPT, Gemini, Copilot, etc.) to...
  - Fuel your curiosity
  - Define context, motivation, or industrial relevance for a given topic
  - Get the “big picture” or alternative techniques than what is taught in class
  - Etc.

Melisa et al. 2025. [Critical Thinking in the Age of AI: A Systematic Review of AI's Effects on Higher Education](#)

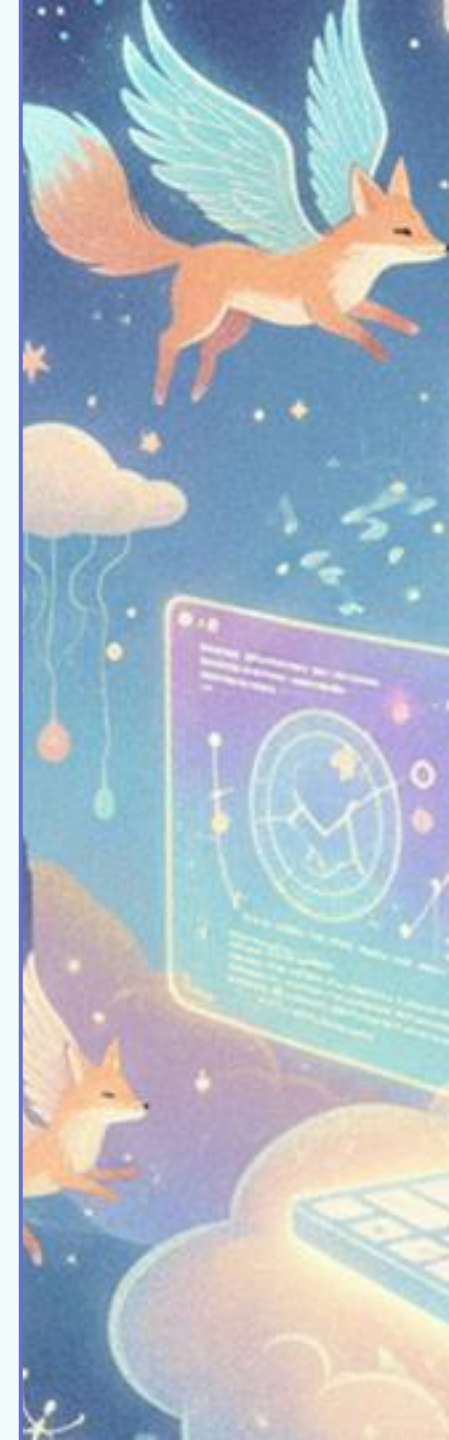


# ...But use it well.



- Be critical!
  - AI can regurgitate bulls\*t reddit comments as if they were truth.
  - Read original sources and judge their authenticity critically.
- DISABLE GitHub Copilot in VS Code.
  - Remember Melisa et al.: “...over-reliance on AI can hinder students' motivation for self-reflection and critical evaluation...”
  - Tab-autocomplete and automatic code generation is extremely risky, unethical, and prevents learning.
  - You want to learn programming skills to be more employable.
  - Practice. Writing. Your code. Yourself.

Melisa et al. 2025. [Critical Thinking in the Age of AI: A Systematic Review of AI's Effects on Higher Education](#)





# GenAI as a colleague.

- Analogy for the ethics of when to use AI.
- Would I have a colleague write my function for me?
  - **NO.** This is plagiarism.
- Would I ask my colleague about different techniques to program a certain function?
  - **YES.** I'm critical/curious of their answer and I code it myself.

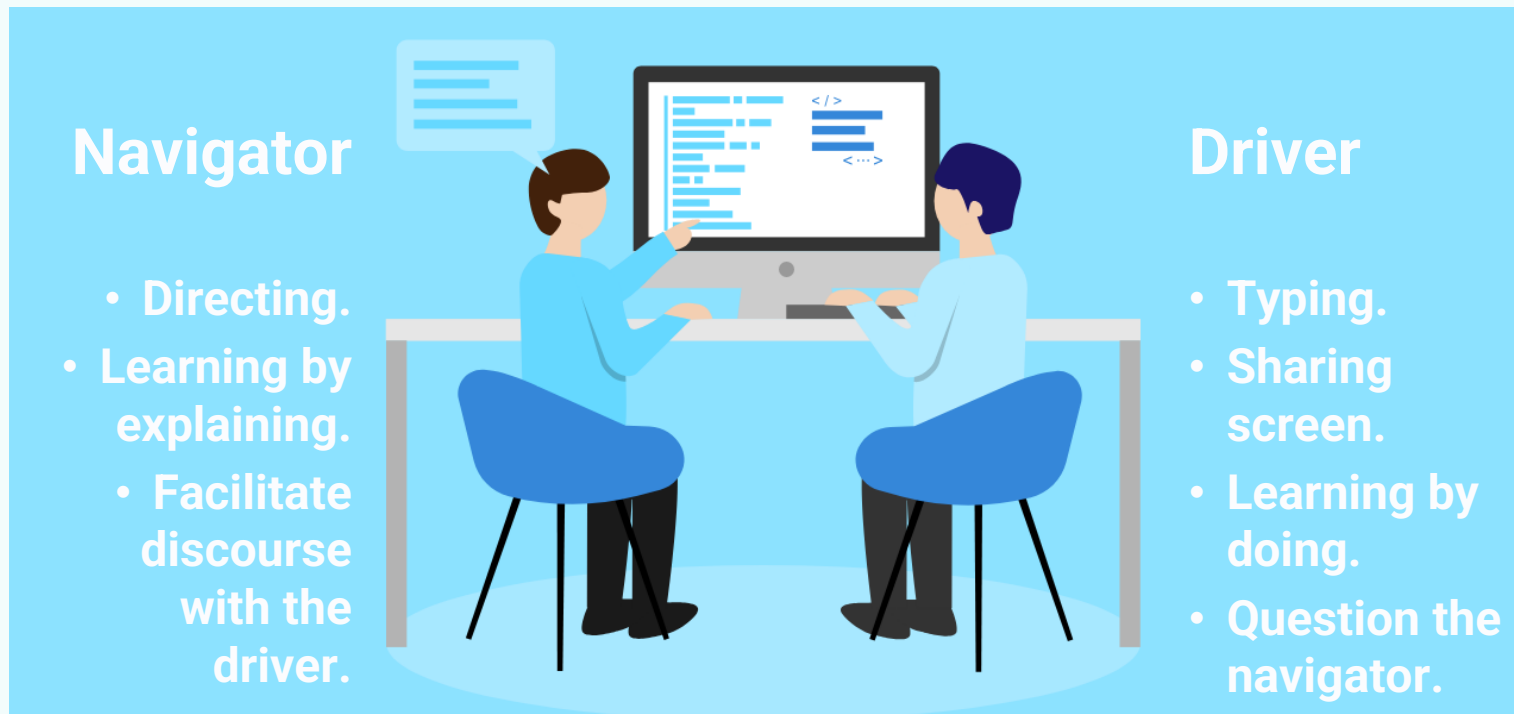


Image: Getty Images/iStockphoto



# 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: 46120-2026

- 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).

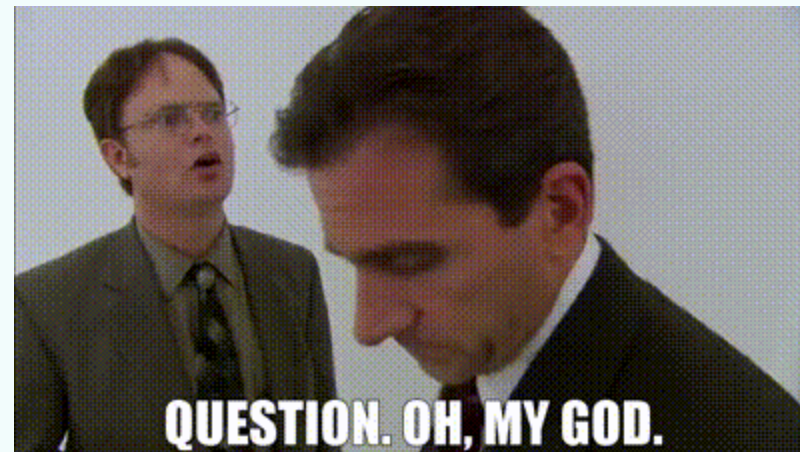
## Learn

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



# Questions?

Post in chat or ask verbally. 😊





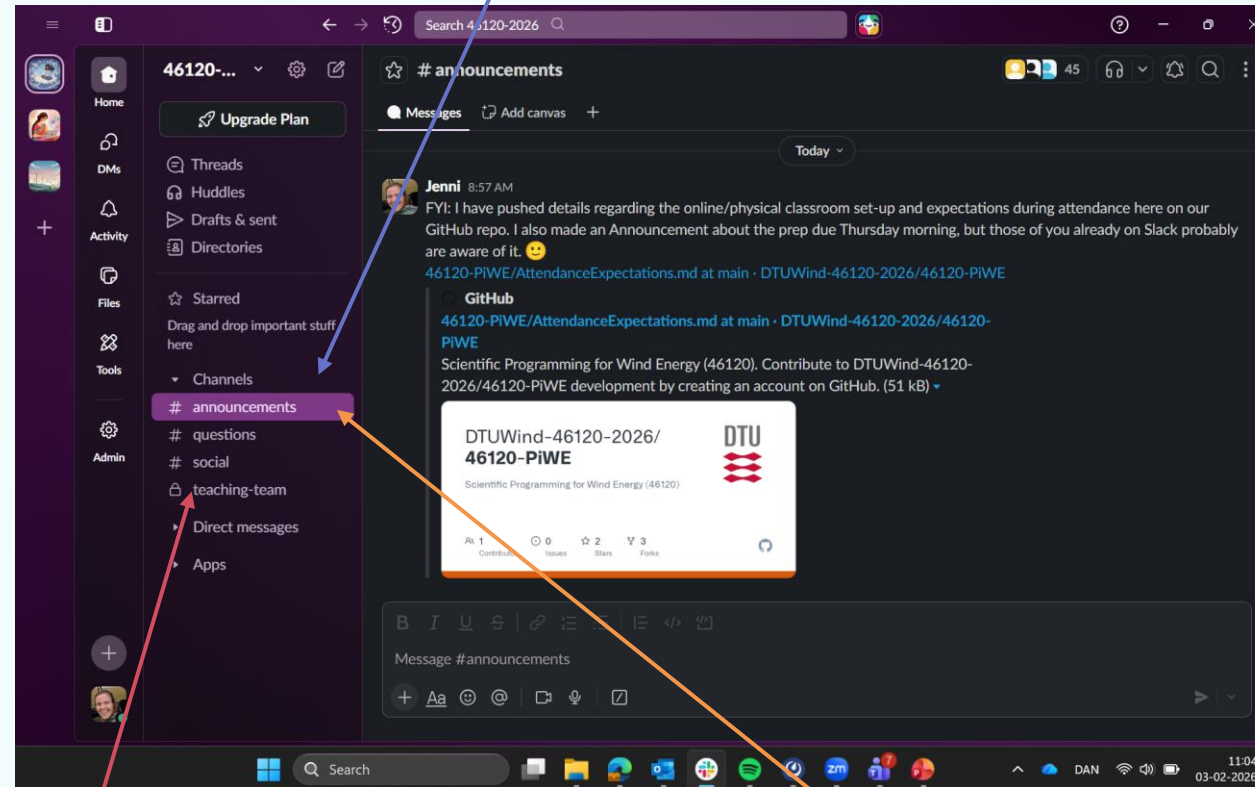
# Let's take a break. 😊

Get ready for new groups at end of break.

During the break, join class Slack if you haven't!

- Link is on Learn page, top right corner.
- Can access slack from the web or download the app.

#announcements:  
for...announcements (shock)  
[only TAs/instructors can post here]



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

#questions: when  
something is not clear



# New groups and introductions.

- **Physical students**, sit in a NEW group of 3, where all three numbers are represented.
  - No overlaps with previous group, please!
  - You can do a group of 2 if needed.
- **Online students**: screenshot this or open locally. Accept BOR invitation.

## In your groups:

1. Introduce yourselves.
2. What makes code “good”?
3. What are some examples of “bad” code you’ve seen? What made it “bad”?



# 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!

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







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

```
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.



```
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
```



```
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



# The challenge of programming

- **Syntax:** This is the grammar of the programming language. It's the set of rules that dictates how statements must be structured.
- **Semantics:** This is the meaning behind the code. It's the logic and purpose of a program. You can write syntactically correct code that still doesn't do what you want it to.
- The **real challenge of programming** lies in mastering semantics — telling the computer what to do in a logical and coherent way.



# Programming as a tool for problem-solving

The basic challenge in this regard for us:

Unlimited complexity  
of the problems we  
try to solve



Limited mental  
capacity of human  
beings

**Summarizing: as a slow-witted human being I have a very small head and I had better learn to live with it and to respect my limitations and give them full credit, rather than to try to ignore them, for the latter vain effort will be punished by failure.**

- E. W. Dijkstra in *Notes on Structured Programming*

