

# Welcome.

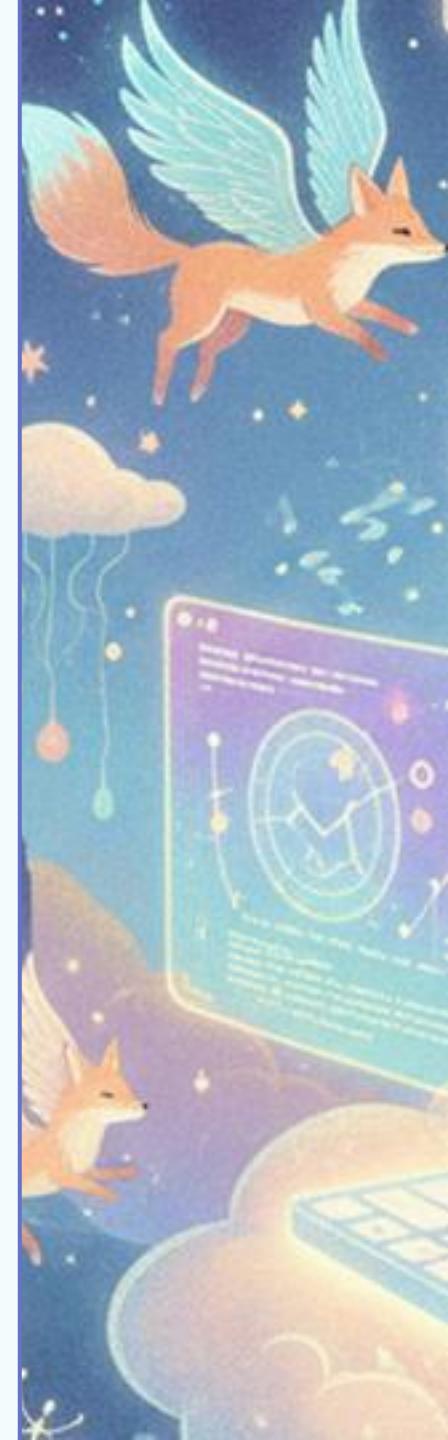
Everyone:

- Pull the updates from the course GitHub repo:
  - `cd <46120-PiWE repo>`
  - `git pull origin main`

LIVE

NB:

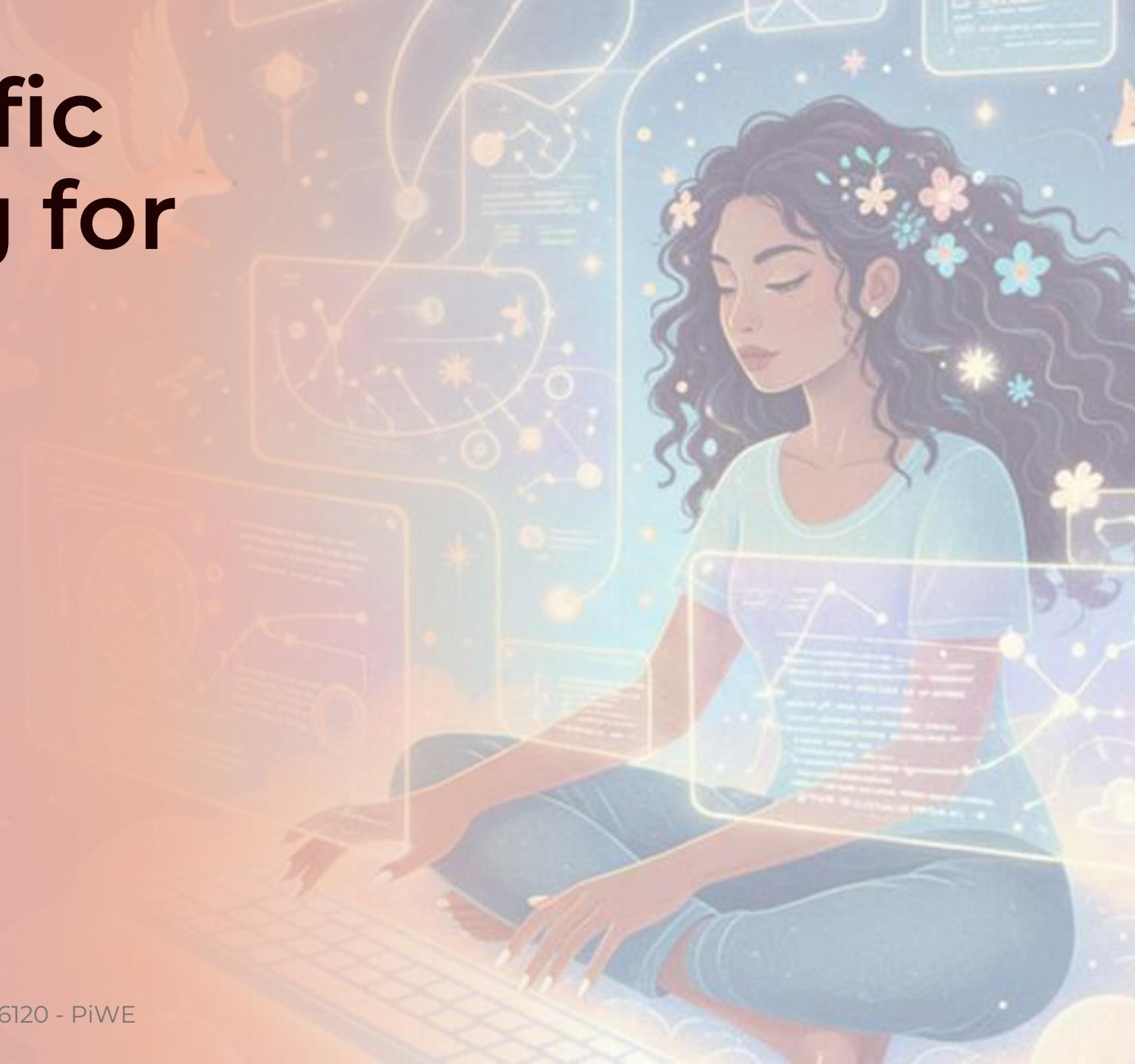
- By attending this class, you consent to being recorded. Recording will be shared to this class and possibly other DTU students for training purposes.



# 46120: Scientific Programming for Wind Energy

## Function handles

Jenni Rinker



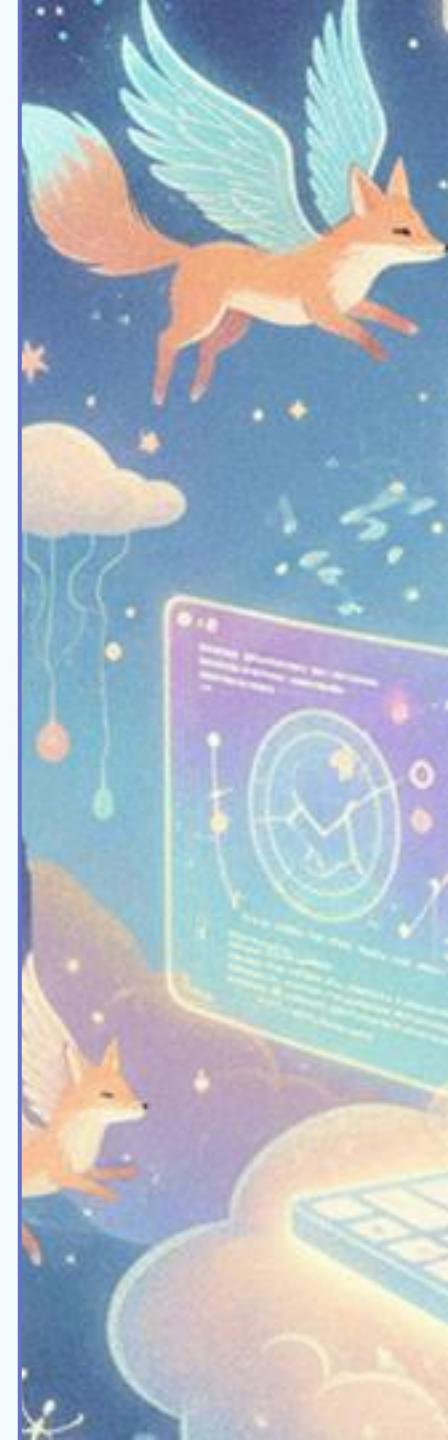
# Agenda for today.

- Pull new course material ✓
- Round robin.
- Function handles.
- Your homework for next week.
  - And preview of what you'll hand in for codecamp.



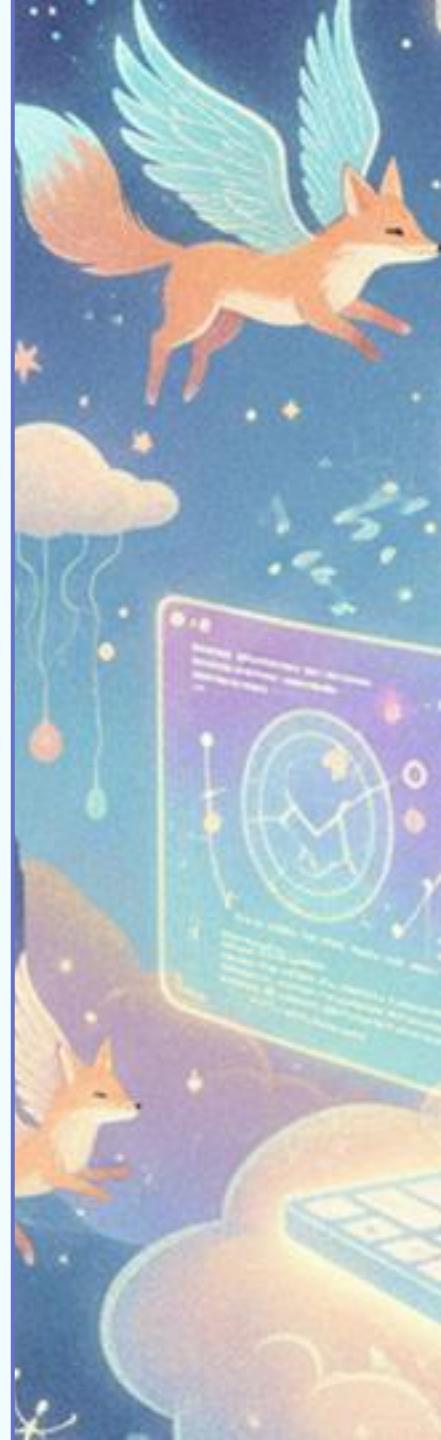
# Round robin

Share solutions with your peers and give feedback.



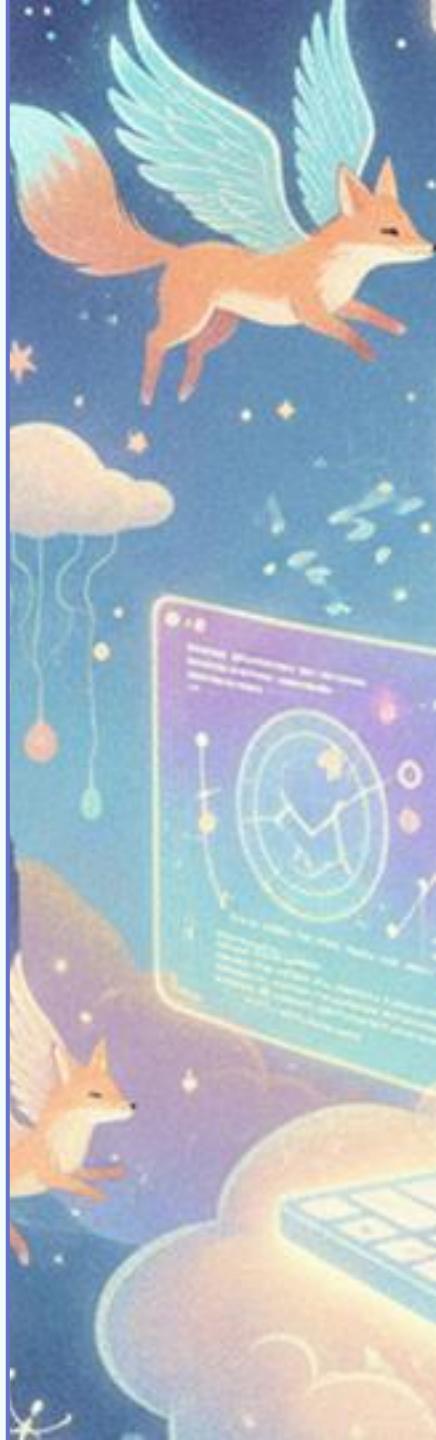
# Time to review and collaborate.

- 1 round of 20 minutes.
- 5 minutes: chaos.
- 15 minutes: present/discuss homework.
  - How “clean” do you feel the team’s code is? How easy to understand?
  - How is collaborating with git going? Any changes since Week 1?
- Afterwards: plenum discussion.
  - Be ready with questions!



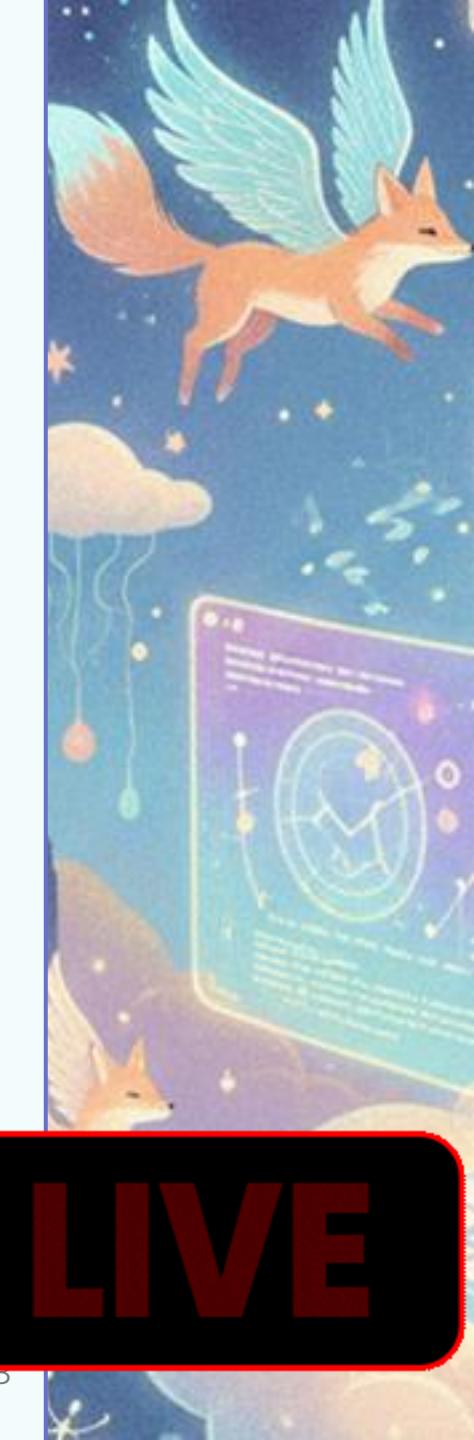
# Notes in plenum.

- (add here)



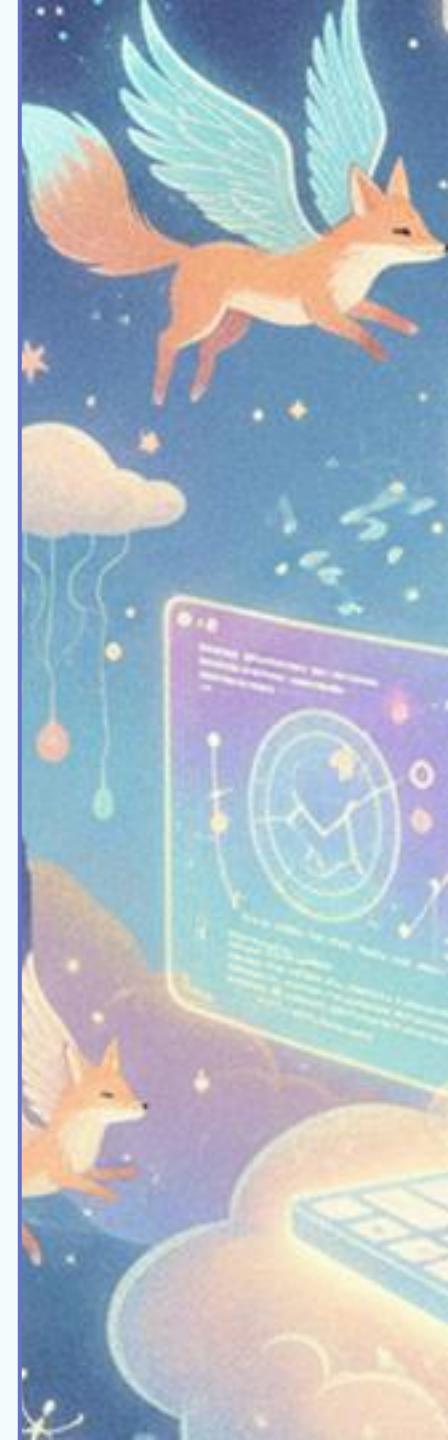
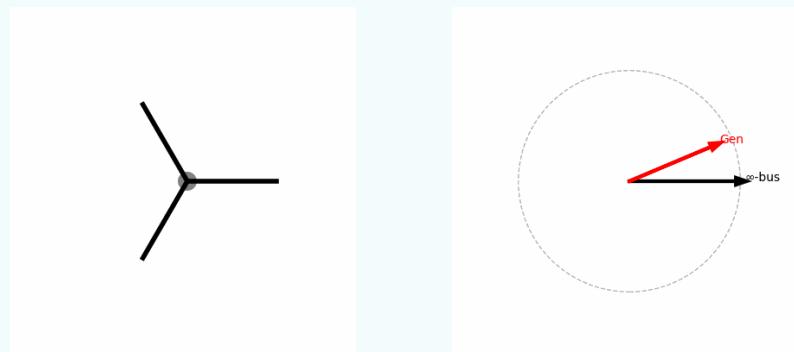
# Solving dynamical systems

A use case for function handles.



# Dynamical systems are fundamental.

- Dynamical systems:
  - A system with changes in state governed by a series of differential equations.
  - A key part of engineering/applied science.
- Examples in our field:
  - A spinning wind turbine.
  - A (dis)charging battery in an EV.
  - Power grid dynamics, e.g., single-machine infinite-bus (SMIB) system.



# Common dynamics expression.

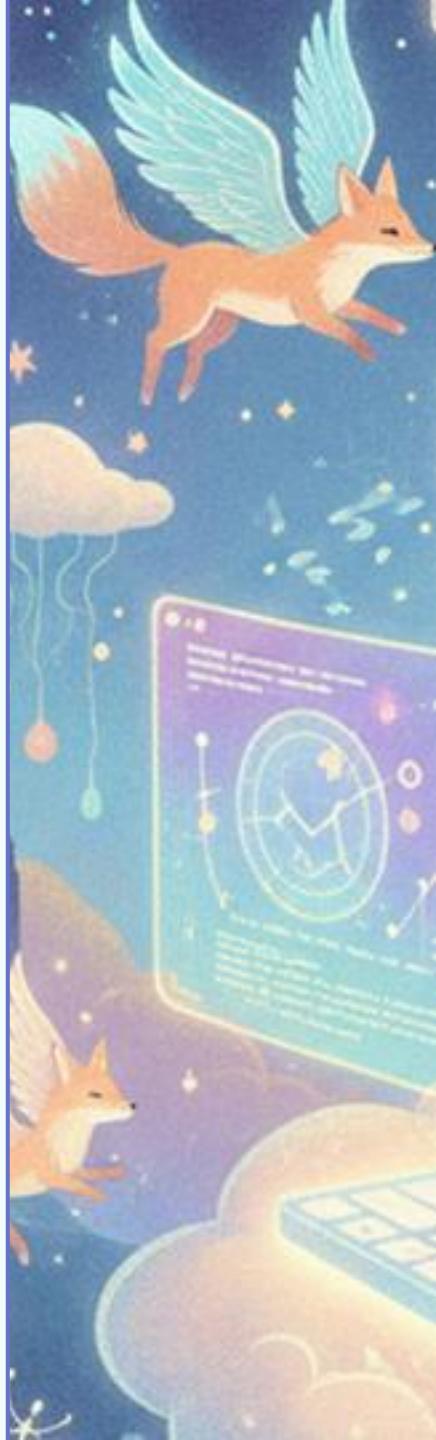
- Express the system dynamics as a differential equation:

$$\frac{dy}{dt} = f(t, y)$$

where  $y$  is a vector with as many degrees of freedom as your system.

- Examples:

	$y$	$\frac{dy}{dt} = f(t, y)$
<b>SMIB</b>	$y = [\delta, \omega]^T$	$f(t, y) = [\dot{\delta}, \dot{\omega}]^T = \left[ \omega, \frac{1}{M} \left( P_m - \frac{EV}{X} \sin \delta - D\omega \right) \right]^T$
<b>Battery (1<sup>st</sup>-order RC)</b>	$y = [SOC, V_{RC}]^T$	$f(t, y) = [S\dot{O}C, \dot{V}_{RC}]^T = \left[ -\frac{1}{Q} I(t), -\frac{1}{R_1 C_1} V_{RC} + \frac{1}{C_1} I(t) \right]^T$
<b>Forced SDOF oscillator</b>	$y = [x, \dot{x}]^T$	$f(t, y) = [\dot{x}, \ddot{x}]^T = \left[ \dot{x}, \frac{1}{M} (F(t) - C\dot{x} - Kx) \right]^T$



# Simulate dynamics with a computer.

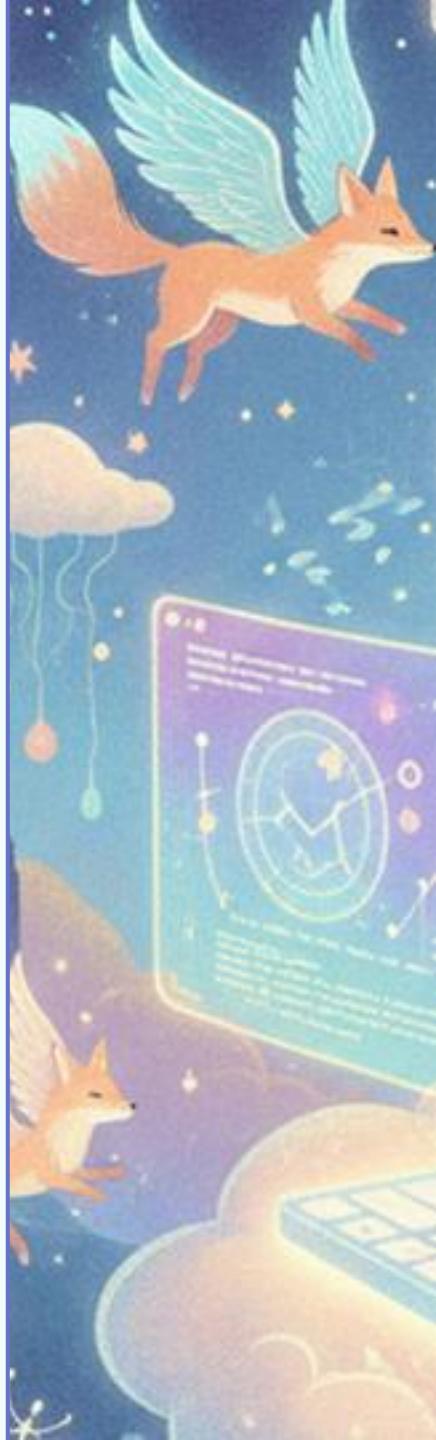
- Algorithms exist to efficiently approximate a discrete solution to dynamical systems expressed as

$$\frac{dy}{dt} = f(t, y)$$

- These algorithms take *functions* we define and numerically evaluate/integrate to produce  $y(t)$ .



- Our goal is thus to define a function to calculate  $f(t, y)$  and correctly pass it into one of these algorithms.
  - Requires *function handles*.



# Function handles

Ya gotta grab things.



# Passing functions into functions.

```
def red_firework():
    return "red"
```

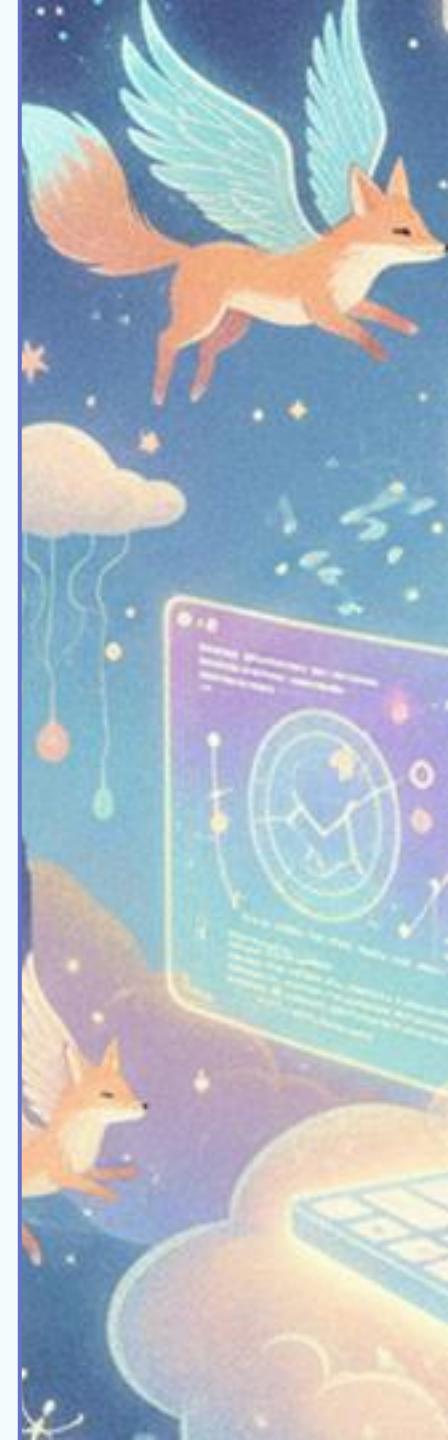
None →



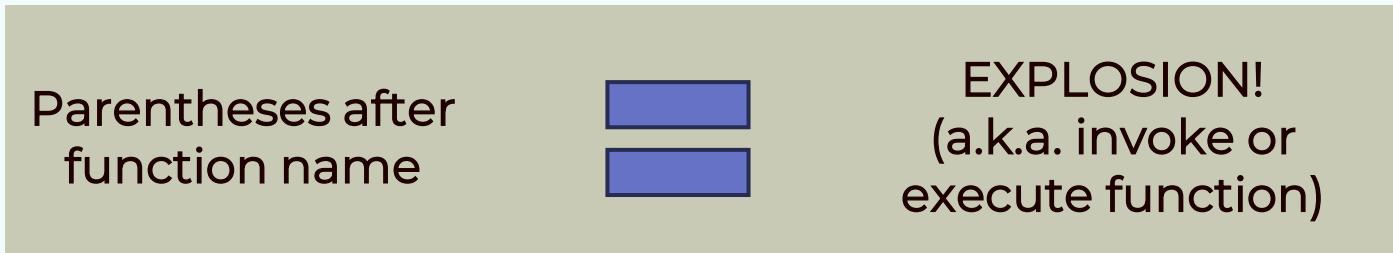
color (str)



a firework-like  
function →



# How not to explode.



- I.e.: No parentheses = no explosion (no execution).

```
def red_firework(): ←  
    return "red"
```

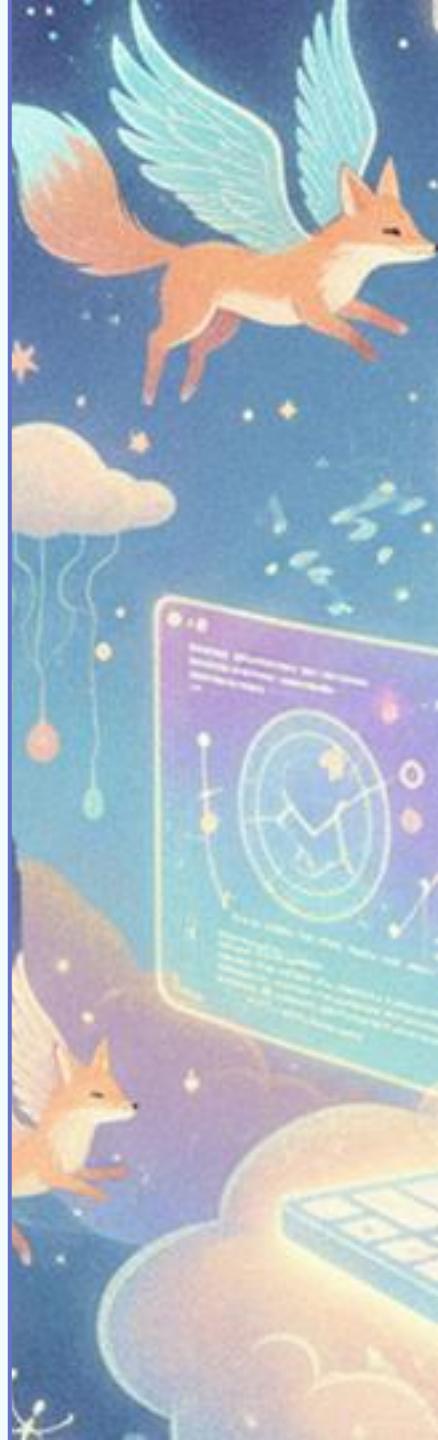
Defines symbolic variable `red_firework` and associate code with variable. Code is **NOT** executed at definition!

```
>>> red_firework ←  
<function red_firework at 0x000002E1E2999E40>
```

No parentheses, so function NOT invoked. Prints metadata of `red_firework`, including where stored in memory.

```
>>> red_firework() ←  
red
```

Parentheses, so function explodes (is invoked).



# Short individual/pair exercise.

1. Open `demo_fireworks.py` in VS Code.
2. Drop-down by arrow button in upper right corner, click “Run Current File in Interactive Window”.
  - This opens an interactive Python terminal where you can enter commands.
  - You might need to re-select the correct Python environment and/or install ipykernel.
3. Predict what you think will be printed when you enter the following commands.

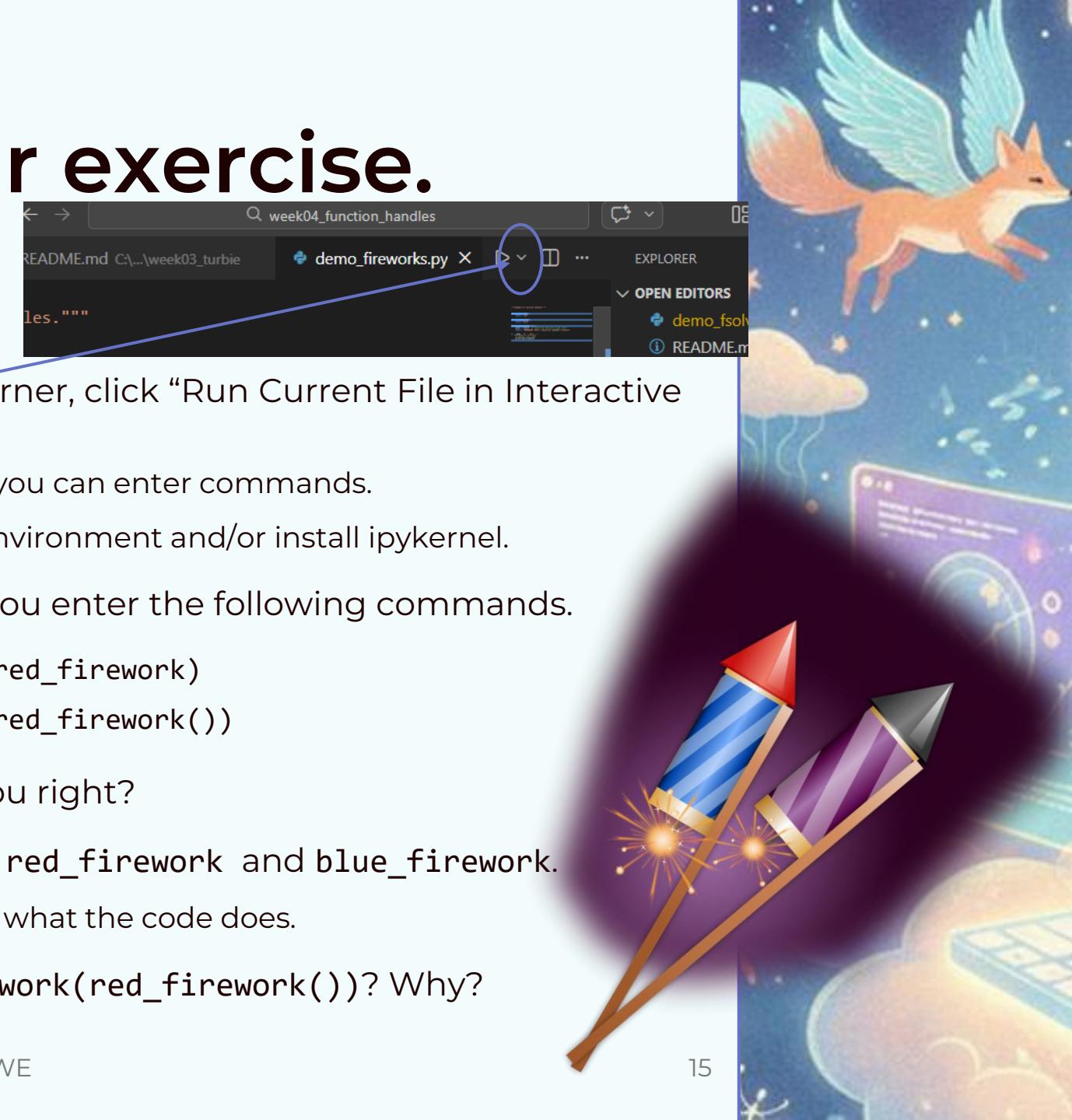
`red_firework`

`red_firework()`

`type(red_firework)`

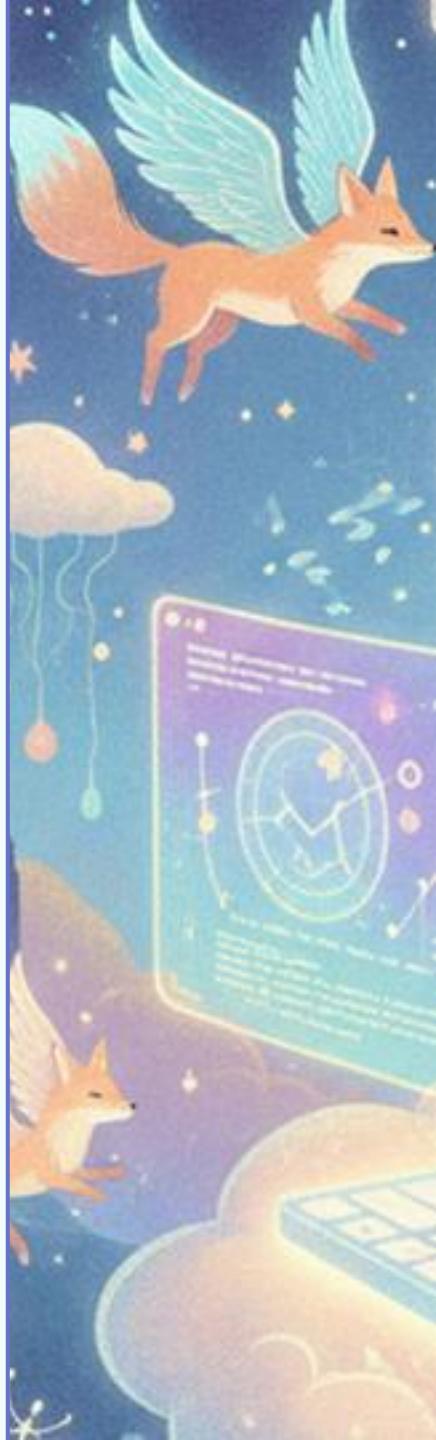
`type(red_firework())`

4. Enter the commands in the terminal. Were you right?
5. Use `light_firework` function to explode both `red_firework` and `blue_firework`.
  - Check out the arguments to `light_firework` and what the code does.
6. Bonus: What happens if you enter `light_firework(red_firework())`? Why?



# Let's discuss.

- In plenum.



# More complicated example. In this week's folder.

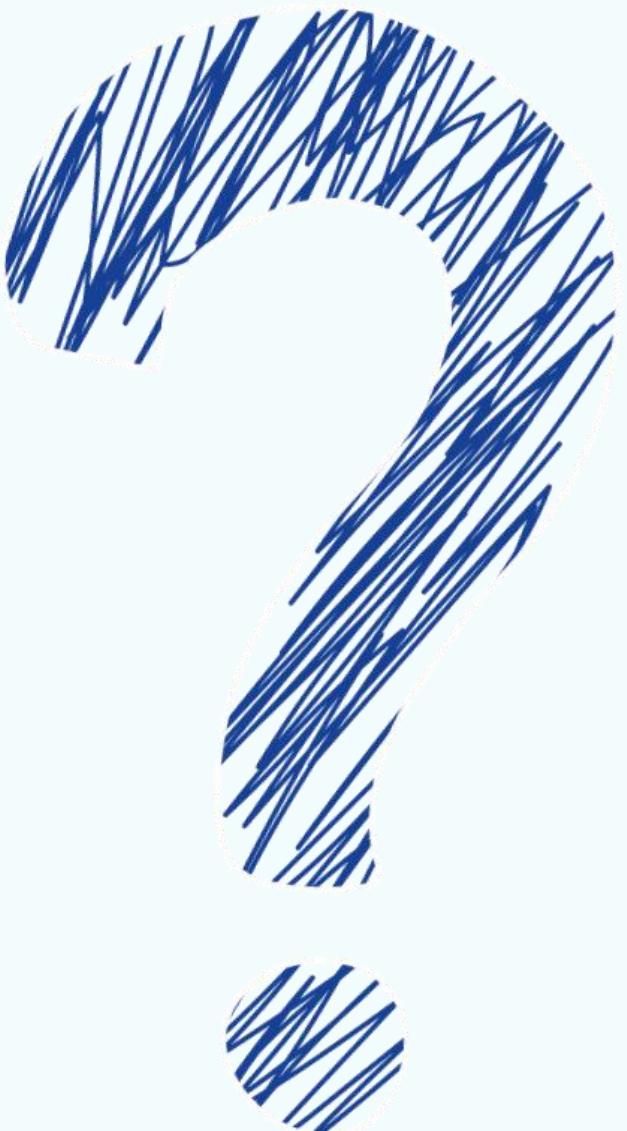
- Sometimes we want to find the root of a function.
  - I.e., when it crosses zero.
- Module `scipy.optimize` has function `fsolve` that does this.
  - You provide the function, initial guess, other things.
  - It iterates:
    - Invoke the function at initial guess
    - Figures out next guess
    - Keeps invoking function until guess is “close enough” per prescribed tolerance



- Why demo provided:
  - Similar in structure to `scipy.integrate.solve_ivp`, which is on homework.

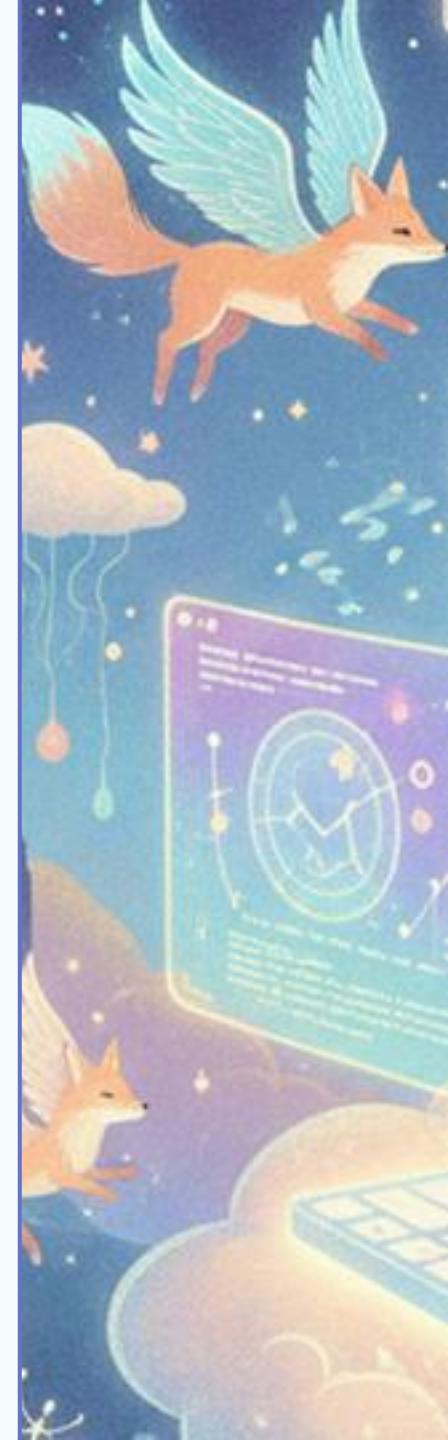


# Questions?



# Homework for this week

What better way to get better at something than to practice?



# First: some information.

“Final” codecamp project, due before Week 6.

- Details/peer-feedback rubric in week06 subfolder.
- Let’s go through it together.
- If you finish this week’s homework quickly and want to move onto final project:
  - Discuss in team method of attack. Code diagrams, feature branches, etc.
  - Ideal: meet with no laptops, just a whiteboard.
  - If plan is clear, may begin working on feature branches.

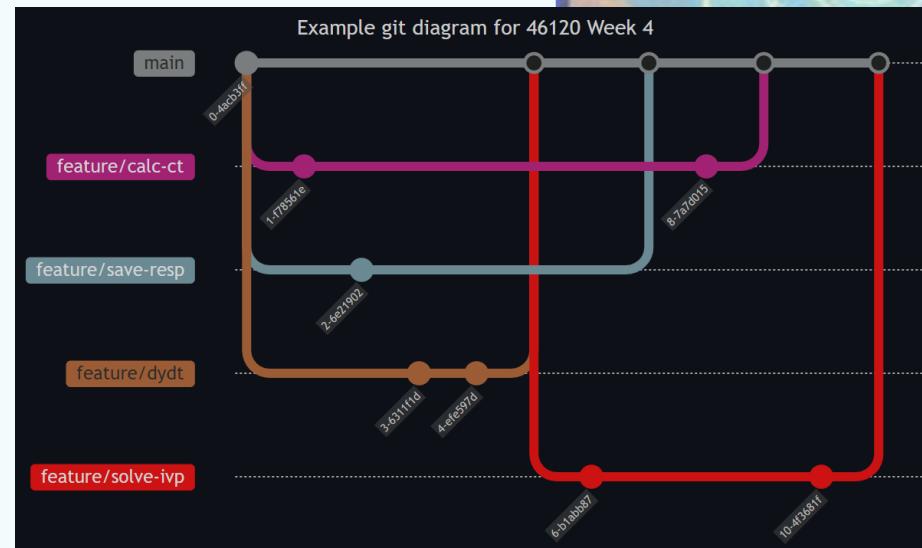


# Overview of homework.

- Objectives:
  - A. Load a look-up-table from file and use it to calculate  $C_t$ , given a wind time series.
  - B. Create a function that evaluates  $dy/dt = f(t, y)$  for both homogeneous and forced response.
  - C. Use `scipy.integrate.solve_ivp` to generate Turbie's time-marching response.
  - D. Save a response time series to file.
- Example git workflow at right.
- Recommended: begin planning/development for CodeCamp final code, if time.
- More details on 46120 GitHub.

heaviest

requires Part B



*Remember, you're expected to work about 6 hours outside of class. Schedule accordingly.*

# Last words.

- Homework details on the course GitHub repo.

Complete Part 1 in class, move on as agreed with your team.

If time this week: start development for final code. (see details page)

Online: we will open self-navigable BORs.

- **To get help during class:** Post in Slack / #debugging if you want a TA to enter your BOR or come find your group.
- NB: We may close the Zoom meeting without warning at 12:00. Be ready with a backup plan.

## Any questions?



# Tutorials.

1. Functions and passing functions [1.11. Defining Functions of your Own — Hands-on Python Tutorial for Python 3 \(luc.edu\)](#)

