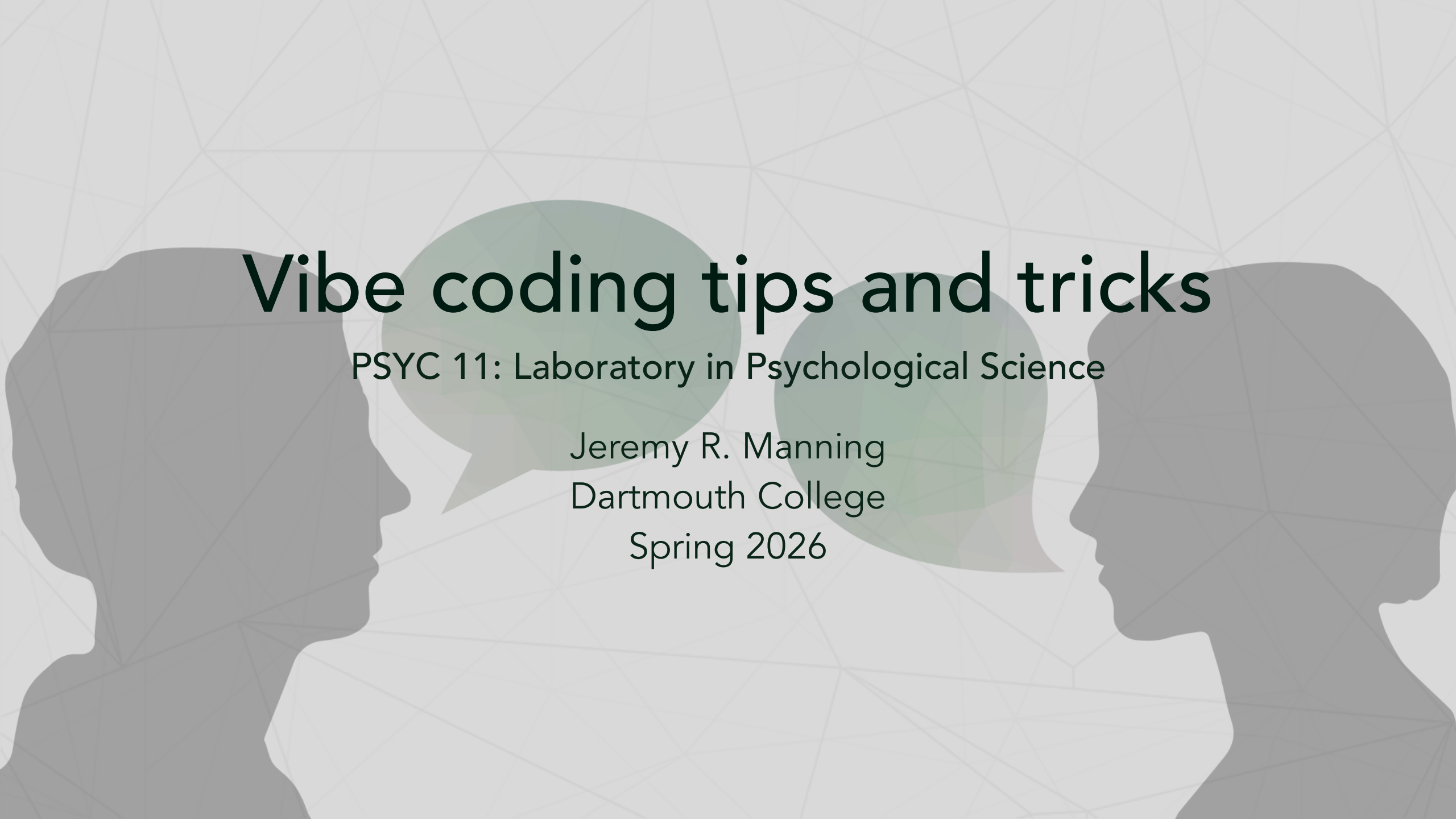


Vibe coding tips and tricks

PSYC 11: Laboratory in Psychological Science

Jeremy R. Manning
Dartmouth College
Spring 2026

Today's agenda

What is vibe coding?

Vibe coding means using AI coding agents to rapidly prototype and implement software by describing what you want in natural language, then iterating on the output.

Topics we'll cover

1. **Free AI coding tools for students:** GitHub Copilot, Google Gemini, claude.dartmouth.edu
2. **Setting up your environment:** VS Code, Claude Code, spec-kit
3. **The spec-kit workflow:** from design docs to implementation
4. **Live demo:** build something together! (*If there's time!*)

Follow along!

Install the tools (and *try using them*) as we go— and ask questions as they arise!

Free coding models

- **GitHub Copilot** (free for students): great at code completion, chat assistance
- **Google Gemini** (free for students): long context, reasoning-heavy tasks
- **Dartmouth GenAI** (chat.dartmouth.edu): free access to many models
- **Dartmouth Claude** (claude.dartmouth.edu): powerful coding model, free for Dartmouth students, faculty, and staff
- **Ollama** and **LM Studio**: run LLMs locally
- **Hugging Face**: open models, useful for integrating into projects

My favorite paid options

- **Anthropic Claude**: fantastic coding model (what I use most!)
- **OpenAI ChatGPT**: powerful, different feel; sometimes when one struggles the other helps

> **Student discounts**
>
> Check for student discounts and free tiers; most AI providers offer generous free usage for students with a `.edu` email address.

Setting up your environment: two(ish) options

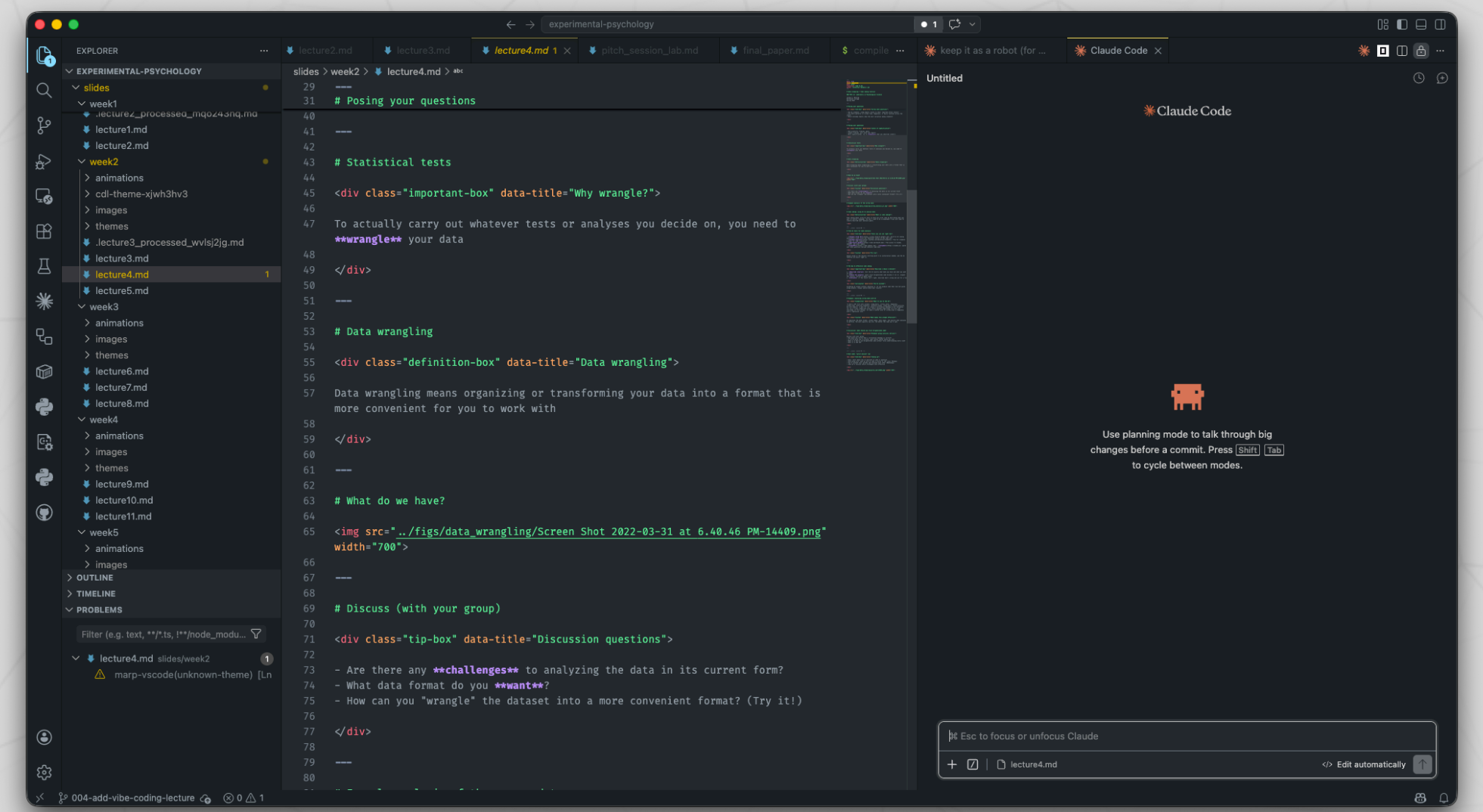
Two main options:

1. **Integrated Development Environment (IDE):** full-featured environment with syntax highlighting, debugging, Git integration, extensions (e.g., VS Code, PyCharm)
2. **Terminal-based coding agent:** lightweight, fast, scriptable (e.g., Claude Code, ChatGPT Codex CLI, OpenCode)

Some other options to try out

- Claude, OpenAI, and OpenCode all have native desktop apps that combine terminal-based coding agents with IDE-like features (file browsing, syntax highlighting, etc.)
- Some IDEs are explicitly designed for AI coding (e.g., Antigravity, Cursor)
- Google Colab now builds in AI coding assistance directly into notebooks (similar to VS Code's Copilot extension); no installation required!

Setting up VS Code



Setting up VS Code (details)

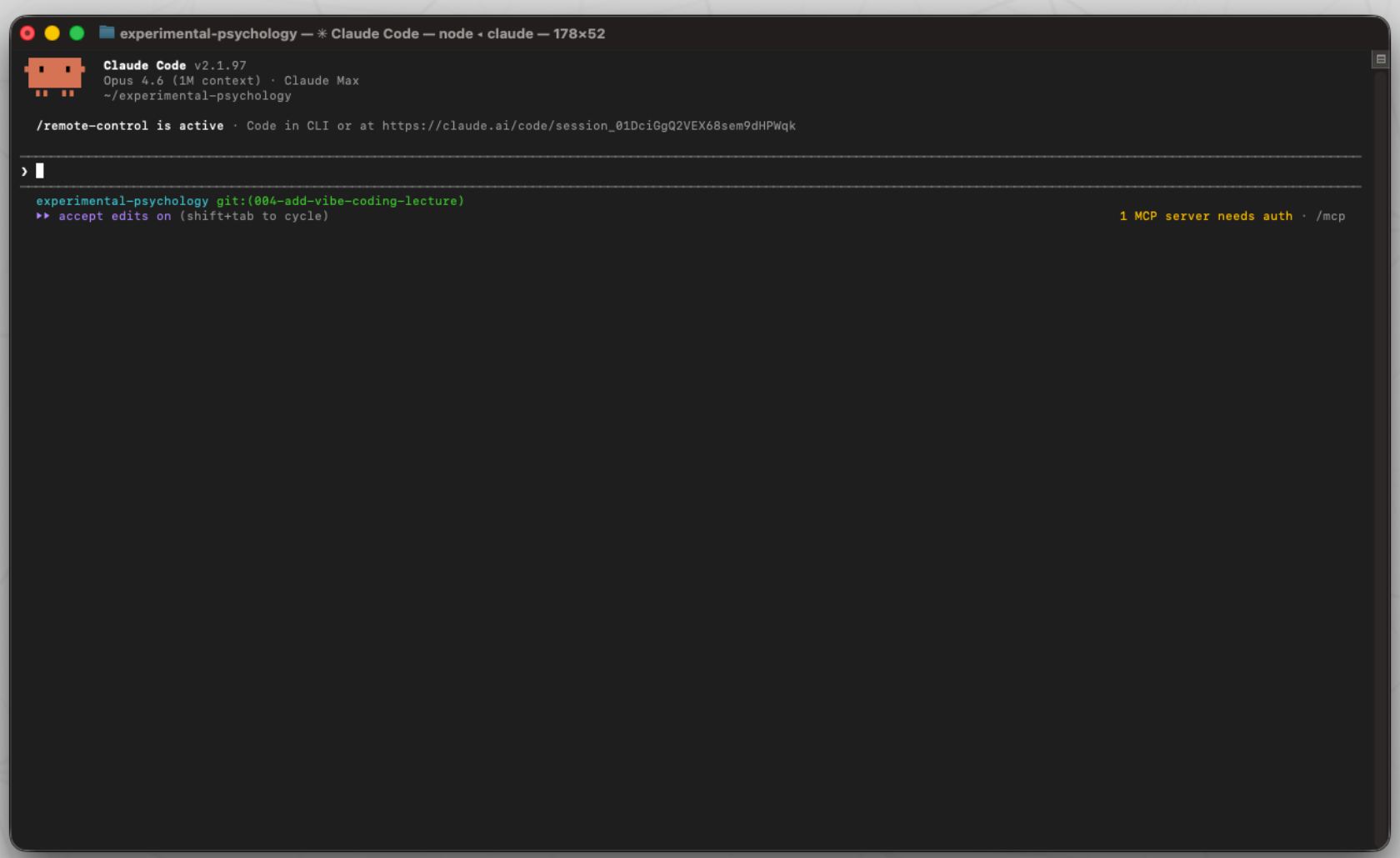
- Download and install from code.visualstudio.com
- Install essential extensions:
  - GitHub Copilot
  - Jupyter
  - Python
  - Claude Code
- Activate Copilot with your GitHub account (click the Accounts icon in bottom left)

Setting up Claude Code (in Terminal)

Install command

```
1 npm install -g @anthropic-ai/claude-code
```

Launch Claude Code



Claude Code configuration

- Use `/model` to switch between models (Claude Sonnet, Opus, Haiku)
- Connect with your Anthropic account, or use GitHub Copilot models
- Claude Code runs in your terminal inside your project directory
- It can read files, write code, run commands, and browse the web

Spec-kit: a framework for AI-assisted software development

What is spec-kit?

A "spec" is a detailed, unambiguous description of what a software project should do. Spec-kit is a workflow for using AI to go from specification to implementation in a structured way.

Install it!

```
# Install the spec-kit CLI tool globally
uv tool install specify-cli --from git+https://github.com/github/spec-kit.git

# Set up spec-kit in your project (from within your project folder in Terminal)
specify init --here --ai claude
```

Orienting Claude Code to your project

Start here!

Clone (download) your repo and `cd` into it.

- Launch Claude Code (run `claude` inside the project folder)
- Run `/init` to tell Claude to explore your project folder
- It maintains a `CLAUDE.md` file to help future sessions understand the project

The four-step vibe coding workflow

1. Detailed description

Specify inputs/outputs, edge cases, and examples

2. Technical design doc

Have AI draft the architecture, iterate on it, produce skeleton code

3. Implementation plan

Break into small tasks with verification steps; mind context limits

4. Implement and verify

Let AI write code, test each piece, stress test the final product

The spec-kit workflow

What is spec-kit?

Instead of writing a technical design doc and implementation plan from scratch, you write a specification first. The spec becomes the executable source of truth.

The 6 steps

1. **Constitution** — establish inviolable project principles
2. **Specify** — describe what you want built
3. **Clarify** — resolve ambiguities interactively
4. **Plan** — generate an architecture and design doc
5. **Tasks** — break the plan into ordered, actionable tasks
6. **Implement** — execute tasks with verification at each step

Writing a good spec

The golden rule

Focus on **WHAT** and **WHY**, not **HOW**.

Example prompt: build a game that tests my reaction time

Spec out a game that tests the user's reaction time. It should display a stimulus (e.g., a circle) at random intervals, and the user should click as quickly as possible when they see it. The game should record the reaction time for each trial and display the average reaction time at the end of the session. It should run in a web browser and be visually engaging. Support desktop and mobile devices, and all major platforms and browsers. Store results locally without sending data to any servers. Include instructions and feedback for the user. Make it fun!

Writing a good spec

The golden rule

Focus on **WHAT** and **WHY**, not **HOW**.

Notice: what are we building?

Spec out a game that tests the user's reaction time. It should display a stimulus (e.g., a circle) at random intervals, and the user should click as quickly as possible when they see it. The game should record the reaction time for each trial and display the average reaction time at the end of the session. It should run in a web browser and be visually engaging. Support desktop and mobile devices, and all major platforms and browsers. Store results locally without sending data to any servers. Include instructions and feedback for the user. Make it fun!

Writing a good spec

The golden rule

Focus on **WHAT** and **WHY**, not **HOW**.

Notice: how should it work?

Spec out a game that tests the user's reaction time. **It should display a stimulus (e.g., a circle) at random intervals, and the user should click as quickly as possible when they see it. The game should record the reaction time for each trial and display the average reaction time at the end of the session.** It should run in a web browser and be visually engaging. Support desktop and mobile devices, and all major platforms and browsers. Store results locally without sending data to any servers. **Include instructions and feedback for the user.** Make it fun!

Writing a good spec

The golden rule

Focus on **WHAT** and **WHY**, not **HOW**.

Notice: what are the design constraints?

Spec out a game that tests the user's reaction time. It should display a stimulus (e.g., a circle) at random intervals, and the user should click as quickly as possible when they see it. The game should record the reaction time for each trial and display the average reaction time at the end of the session. **It should run in a web browser and be visually engaging. Support desktop and mobile devices, and all major platforms and browsers. Store results locally without sending data to any servers.** Include instructions and feedback for the user. **Make it fun!**

Example: Gamified cognitive testing battery

What we're building

A single HTML file that runs participants through a battery of quick gamified cognitive tests (Stroop, free recall, go/no-go, N-back, digit span, flanker, visual search, mental rotation) and displays a bar chart of performance with a class leaderboard.

Real research!

You could use something like this in part 2 of this course!

The spec-kit workflow for our demo

Documents at each step

Each step produces a document that becomes the source of truth.

Why this matters

The spec-kit workflow keeps everything grounded in a clear, unambiguous specification.

Constitution

Prompt: /speckit.constitution

Create a constitution for this project with these principles:

- Single HTML file (nothing to install or download, runs in any browser)
- Results are stored locally (no data sent to servers)
- User delight: smooth animations, clean design, fun feedback
- Privacy and security: no personally identifiable information collected
- Leaderboard uses three-letter user-chosen codes
- Must work on mobile and desktop browsers
- Accessible design (WCAG 2.1 AA)

Output

A `constitution.md` file with inviolable rules.

Specify

Prompt: /speckit.specify

Build a gamified cognitive testing battery. Single HTML file.

User Journey:

1. Student enters their anonymous class code
2. Sees a menu of 8 cognitive tasks (Stroop, free recall, go/no-go, N-back, digit span, flanker, visual search, mental rotation)
3. Completes each task with animated instructions and feedback
4. After all tasks, sees a bar chart of their scores
5. Can view a class leaderboard comparing anonymous scores

Output

A `spec.md` with user stories, acceptance criteria.

Clarify

Prompt: /speckit.clarify

The agent might ask...

1. How long should each task take?
2. What scoring metric for each task?
3. How to handle incomplete sessions?
4. Should tasks be randomized?
5. What happens if localStorage is full?

Output

An updated `spec.md` with notes about what was clarified or changed.

Plan and tasks

Prompt: /speckit.plan

Run this to generate a plan for implementing the project and define success criteria for each project milestone.

Prompt: /speckit.tasks

Run this to break the implementation plan into discrete, ordered tasks with clear acceptance criteria for each task.

Optional: /speckit.analyze

For complex projects, run this to identify any inconsistencies, gaps, or potential issues in the specification or plan before you start implementing.

Implement

Prompt: /speckit.implement

Running this command will launch an interactive coding session where the agent will execute the implementation plan task by task, generating code, running tests, and verifying outputs at each step.

Output

Whatever each task specifies: functions, classes, tests, documentation, etc. A task is "done" when all its acceptance criteria are met.

Babysitting

Claude will prompt you (often very frequently) to ask for permission to run code, execute commands, or change files. Read them carefully the first time you see them, and then once you get used to how it works you can usually just skim and click "yes" to keep things moving.

Guiding principles

Simplicity

"Simplicity is the art of maximizing the amount of work not done."

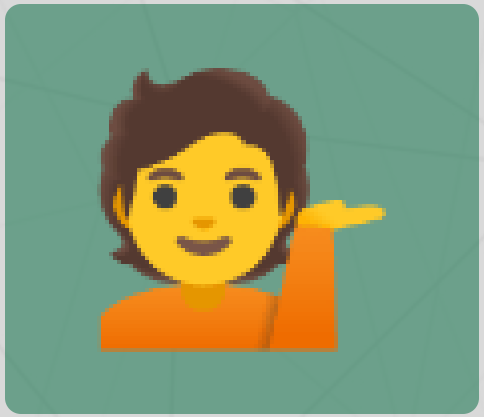
Five principles for effective vibe coding

1. **Start small** — get a working prototype before adding features
2. **Be specific** — vague prompts produce vague code
3. **Verify everything** — never trust AI output without checking it
4. **Iterate rapidly** — small changes, frequent testing
5. **Document as you go** — your future self (and your AI) will thank you

Questions? Want to chat more?

Email me

Join our Slack



Come to office hours

Up next this week...

Friday: data wrangling and how far can you get with stats?