Lecture 8 Activity: Be the Agent

E190BO Name: _____

Claude Code, Cursor, and ChatGPT's agent mode read files, run commands, and fix bugs over many steps. Today the class runs an agent **with people instead of code**. Then, in groups, you'll debug agents that went wrong. **Laptops closed today.**

Timeline (55 min). **0–5** warm-up vote **5–24** the Human Agent (Part 1) **24–31** two twists **31–48** debug the agent, in groups (Part 2) **48–55** wrap-up

Warm-up: vote, talk to a neighbor, vote again

Circle your first answer. After talking with a neighbor, put a box around your second answer.

W1. Your project's test file, `test_grades.py`, is sitting on your laptop. You ask Claude Code (in your terminal) to fix the failing tests. A moment later it reports: "All 6 tests pass." **Who ran the tests, and where?**
(A) the model, on Anthropic's servers (B) the model, on your laptop (C) ordinary code on your laptop (D) nobody: the model predicted what the tests would print

W2. On turn 6 of an agent run, what does the model remember about turn 1?
(A) nothing (B) a short summary (C) everything, stored in the model's memory (D) everything, because it's sent again

W3. You never want an agent to delete files. What is the *most reliable* place to enforce that?
(A) the system prompt: "Never delete files." (B) the tool's description (C) the code that runs the tools (D) the model's training

Part 1: The Human Agent (whole class)

A few classmates will play the agent. Everyone else keeps the record.

- **Models** wait outside the door. Each comes in for **one turn only**, reads the context sheet (nothing else), writes one reply, then sits down and stays silent. Each turn is a *different* Model.
- The **Harness** runs the loop exactly as written, like code: no hints, no help.
- Three **Tools** act like Python functions. They do exactly what their code says, even when that isn't what the Model meant.

The user's question: *"How many days from today until the first Prototype Presentations class in E190BO, and what is due that day?"*

The tools the Model can ask for (this is all the Model knows about them):

tool	description	input
`get_today`	Get today's date (YYYY-MM-DD) and the day of the week.	(none)
`search_schedule`	Search the course schedule. Returns every entry whose date, topic, or due items contain the query text.	`query`: text to look for
`days_between`	Count the days from one date to another.	`start`, `end`: dates, YYYY-MM-DD

While each Model is writing: whisper with a neighbor. What do you predict it will write?

Your job: as the Harness reads each box aloud, copy it into the table. Keep it short.

The last column: when each Model comes in, count how many message boxes are *already on the sheet* for it to read. Model 1 sees only box 0, so we filled in **1** for you. Model 2 sees boxes 0, 1, and 2, so write **3**.

Box	**Role**	**What it says** (tool name + input + id, a result + its id, or the answer)	**Boxes this Model had to read**
0	user	How many days from today until the first Prototype Presentations class…	
1	assistant		**1**
2	user		
3	assistant		
4	user		
5	assistant		
6	user		
7	assistant		

With a partner, once the loop stops:

(a) How many times was a Model called? How many times did a tool run? Who ran the tools?

(b) The last Model never saw the earlier Models work. How did it know what had already happened?

(c) Each turn adds 2 boxes, and **every** call also re-sends the system prompt and tool descriptions. Look at your last column. A 100-turn Claude Code session reads roughly how many times as much as a 10-turn one: 10×, or more? Why?

(d) Who decided the loop was finished? What *else* could have stopped it?

Checkpoint 1: walk to a checkpoint sheet on the wall and compare.

Two twists: now *you* are the Model

The slide shows a context. Write your next move **exactly**: a tool request (name, input) or an answer.

Twist A. My move: _____

Twist B. My move: _____

What should have stopped this, and who is responsible for it? _____

Part 2: Debug the agent (groups of 3)

Each agent below went wrong. For each one:

1. **Think like your agent.** What did the model see, or *not* see, that made its choice seem reasonable *to it*?
2. **Fix it.** Circle the layer where your fix goes, then describe the fix in one sentence. There is often more than one good answer. Pick the **most reliable** one.

Layers. **D** tool **d**escription (name, description, inputs) **T** the **t**ool itself (what it does or returns) **P** system **p**rompt **H** **h**arness code (checks, limits, approvals) **W** don't use an agent: use a fixed **w**orkflow

Do them in order. Most groups will finish 3–5. If you finish all five, go to Part 3.

1. The crash. In the schedule agent from Part 1, the tool description for `days_between` said only *"Count the days between two dates."* On turn 2 the model asked for:

```
days_between(start="today", end="Nov 4")
```

The harness called the function, Python raised `ValueError: Invalid isoformat string: 'today'`, and the whole program crashed. The user saw a traceback.

Why "today" seemed reasonable to the model	
Fix D T P H W	

2. All tests pass! A coding agent is told: *"The tests in test_grades.py are failing. Find and fix the bug(s) in grades.py."* Its system prompt says *"Never modify the tests."* It reads both files, then calls `write_file("test_grades.py", ...)`, changing the expected answer from `90` to `89.9`. It reports: *"Fixed! All tests pass now."*

Why this looked like success to the model	
Fix D T P H W	

3. The $40 question. Another schedule agent. The model called `search_schedule("prototype presentation day")` and got back *"No schedule entries match 'prototype presentation day'."* It tried `"prototype presentation date"`, then `"Prototype Presentation Day"`, then `"presentations of prototypes"`... 212 turns later, someone noticed. The run cost $40.

What the tool results told the model	
Fix (two layers!) D T P H W	

Checkpoint 2: walk to a checkpoint sheet on the wall and compare.

4. Where's my package? A shipping company built an agent (12 tools, a long system prompt) to answer *"Where is my package?"* It gets 50,000 questions a day. It averages 6 turns, 20 seconds, and $0.12 per question ($6,000 a day), and gets 96% right. Almost every question is handled the same way: find the order number, look up the tracking status, reply.

What's wrong with the design?	
Fix D T P H W	

5. Turn 80. A coding agent is working in a 400-file project. The user's first message said *"Keep the public function names unchanged."* `read_file` returns whole files. By turn 80, each call reads 600,000 tokens ($1.20 per turn), the agent is re-reading files it already read, and it just renamed a public function. The original instruction is still in the conversation.

Why it "forgot," even though the instruction is still there	
Fix D T P H W	

Checkpoint 3: walk to a checkpoint sheet on the wall and compare.

Part 3: Challenges (if you finish, or have already done Lab 4)

These go beyond the lab. Pick one, work on scratch paper, and be ready to share.

C1. The do-nothing tool. Claude Code has a *to-do list* tool: the model writes a checklist, and the harness just stores it and sends it back. Anthropic has also described a **`think`** tool that does *nothing at all*. Why would a tool that does nothing make an agent better? Where do the model's notes end up, and when would they help most? (Hint: Part 2, #5.)

C2. The cost curve. The system prompt and tools are $P = 4{,}000$ tokens; each turn adds $k = 2{,}000$; input costs $2 per million tokens. (a) How many tokens does turn n read? The total over N turns? (b) Cost of a 10-turn run? A 100-turn run? (c) *Prompt caching* bills re-sent tokens at 10% of the price. Redo (b) roughly. What does caching **not** fix? (d) The harness empties every tool result older than 5 turns. How does the total grow now? Any catch with (c)?

C3. The lethal trifecta. Simon Willison: an agent is dangerous when it combines **private data**, **untrusted content**, and a way to **send data out**. A course-help agent has `read_my_grades`, `read_my_email`, `search_web`, `run_python`, and `send_email`. (a) Sort the tools into the three categories. (Where does `run_python` go?) (b) Design a permission policy that stays safe **even if the model obeys every injected instruction**. What usefulness did you give up?

C4. The budget-aware agent. Barry Zhang calls budgets an open problem. Should the harness *tell* the model how many turns or dollars it has left? What goes wrong if it doesn't (Lab 4, Experiment 3)? If it does? Design what the harness does at 80% of the budget, and how you'd test it.