



AGENT 41

Agent41 Course
Apache OpenServerless

Lesson 1
Vibe Coding with Cursor



The AI Code Editor

Built to make you extraordinarily productive, Cursor is the best way to code with AI.



DOWNLOAD FOR MACOS

ALL DOWNLOADS

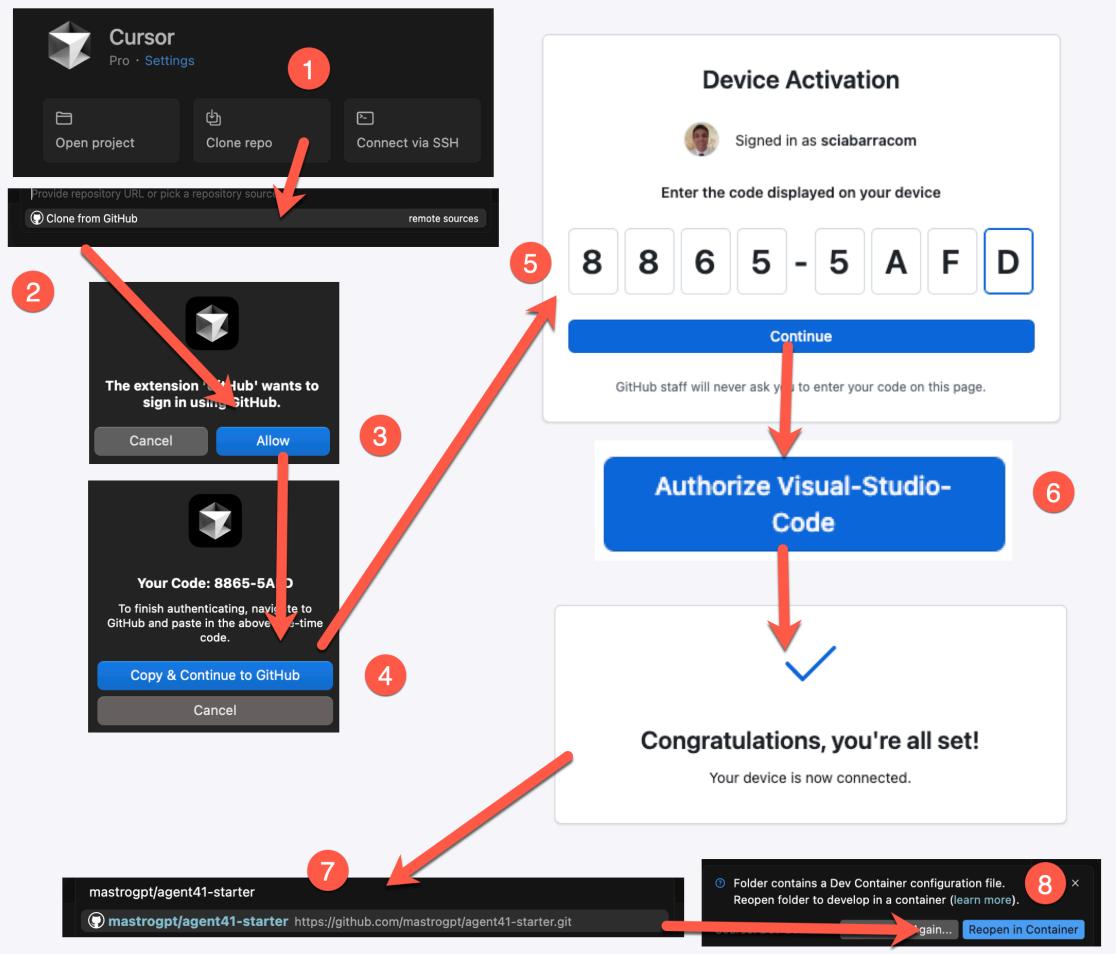
The screenshot shows the Cursor application window. At the top, there's a toolbar with standard Mac OS icons. Below it is a tab bar with 'mod.rs' selected. The main area contains a code editor with the following text:

```
72
73
74
75
76 // listeners sent from the old process for graceful upgrade
77 #[cfg(unix)]
78 upgrade_listeners: Option<ListenFds>,
79 upgrade_listeners: ListenFds,
80 }
```

A tooltip above the code editor says: "Implement the cleanup function for the transport stack. Do not make the upgrade listeners optional." To the right of the code editor is a sidebar with tabs for 'CHAT' and 'COMPOSER'. The 'CHAT' tab shows a message: "mod.rs Current File Could you make it easier to switch certificates in the transport listeners?". At the bottom right, there are buttons for 'Ask', 'Copy', and 'Apply'. The overall interface has a clean, modern look with a dark theme.

Install Cursor: <https://cursor.com>

Setup



1. Clone repo
2. Clone from GitHub
3. Allow GitHub access
4. Copy Device Code
5. Paste Device Code
6. Authorize VSCODE
7. Clone
8. Open Devcontainer

`mastrogpt/agent41-starter`

1. Open AI Chat

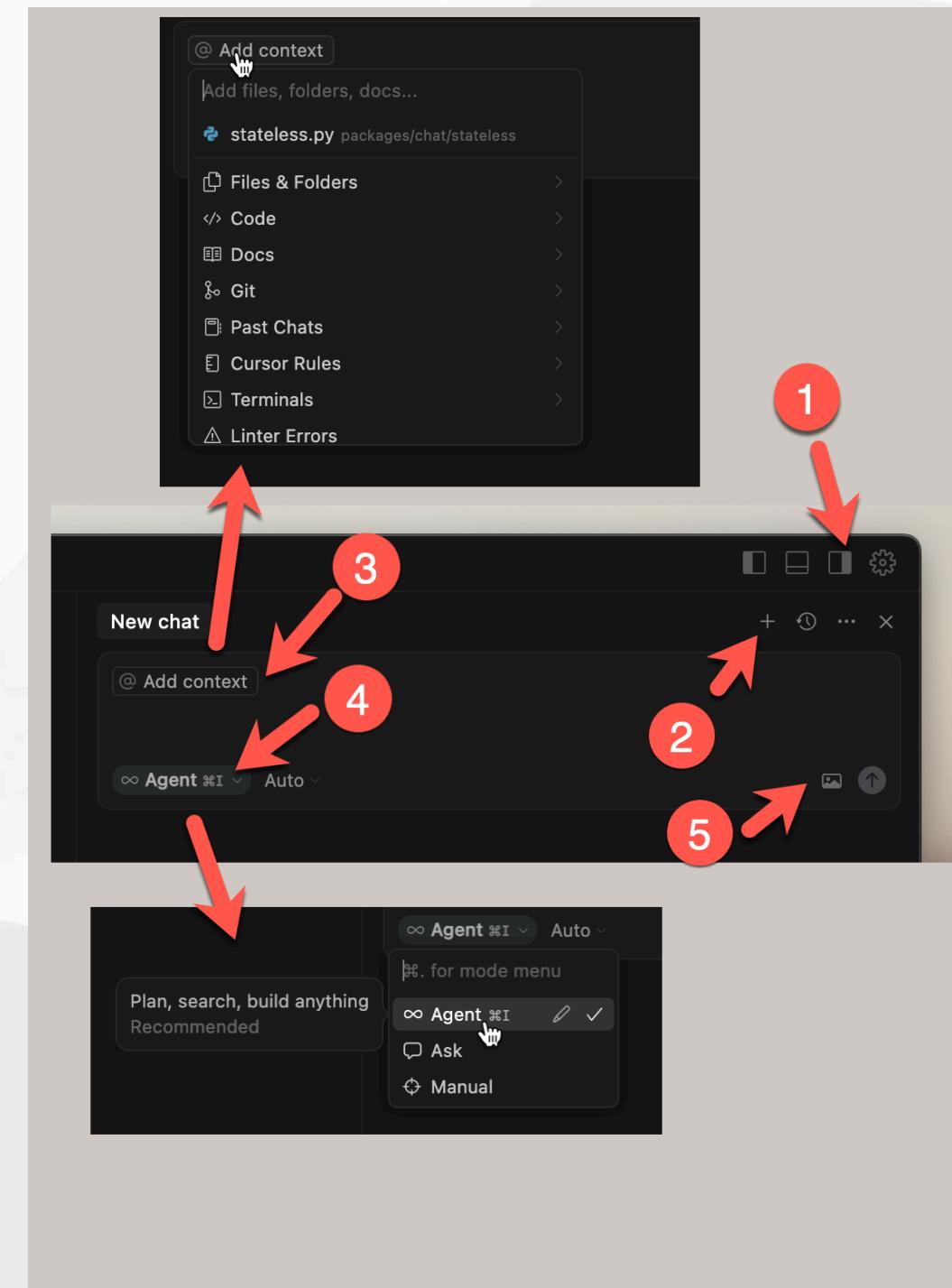
2. Create a New Chat

3. Select the context

- File & Folders
- Code
- More...

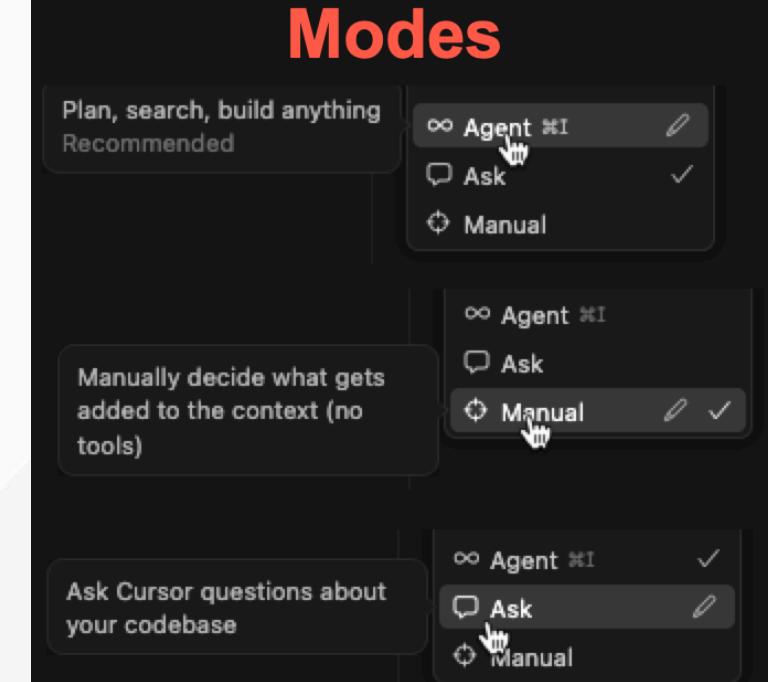
4. Select the Mode

- Agent
- Ask
- Manual

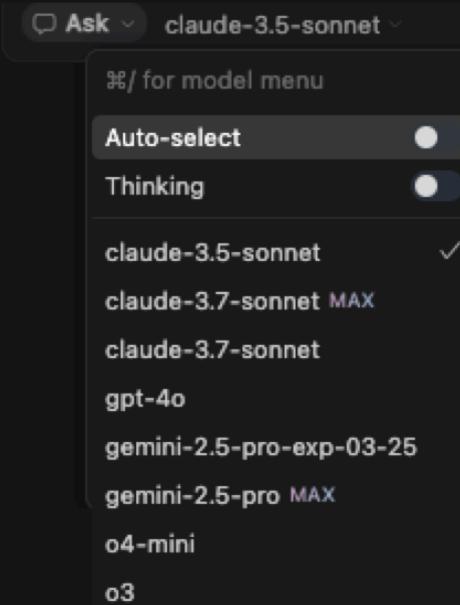


Modes

- **Agent:** fully automated
 - Find the context
 - Propose solution
 - Implements them
- **Ask**
 - As agent but do not act
- **Manual**
 - You have to provide the context

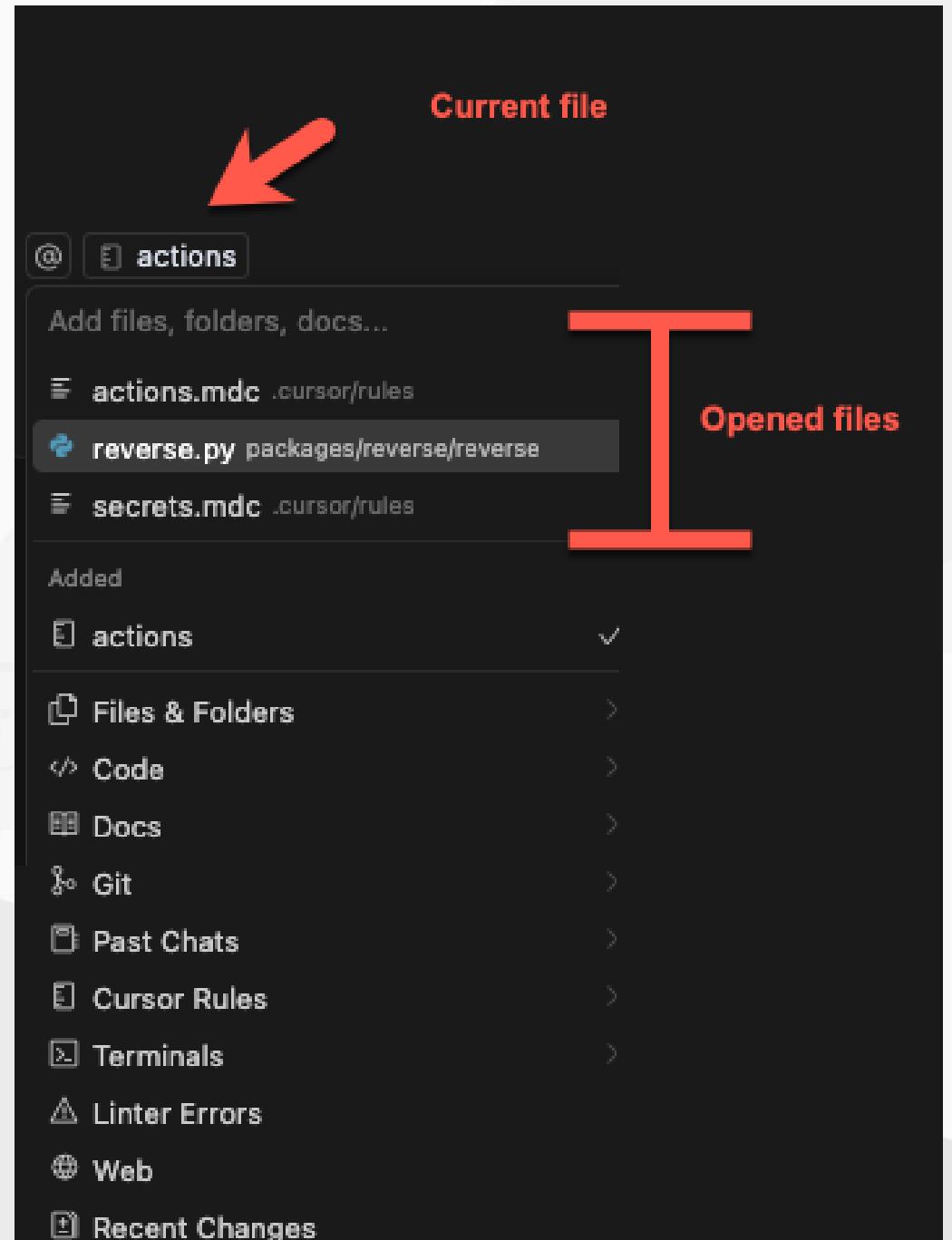


Models



Contexts

- Files and Folders
- Code (snippets)
- Docs
- Past git revisions
- Past chats
- Terminal commands
- Errors
- Web
- Cursor Rules



OPEN EDITORS

X D secrets.mdc U

AGENT41-STARTER [DEV CONTAI... ●

✓ .cursor/rules ●

✗ actions.mdc U

✗ secrets.mdc U

> .devcontainer ●

> packages ●

> site ●

> tests ●

> web ●

❖ .gitignore

✗ pytest.ini

Cursor Rules

.cursor > rules > ✎ secrets.mdc

Always

✓ Auto Attached

Agent Requested

Manual

File pattern matches ⓘ

packages/*/*.py ×

: redis you need the following secrets:

- REDIS_PREFIX

When you use OLLAMA you need the following secrets:

- OLLAMA_HOST

- AUTH

When you use S3 storage you need the following secrets:

- S3_HOST

- S3_PORT

- S3_ACCESS_KEY

- S3_SECRET_KEY

- S3_BUCKET_DATA

- S3_API_URL

Create Weather Tool

- create a tool named weather

- use a rule

- access to a service to get the weather in a location

- asks a question

- use openweather

- my answer

The screenshot shows a terminal window titled "Weather Tool Creation". The command entered is "create a tool named weather". The terminal output details the creation of a tool named "weather" using a rule. It shows environment variables (NEW_POSTGRES, NEW_S3, NEW_REDIS, NEW_MILVUS) set to false, Python version (3.12.4), virtual environment creation (.venv), and the creation of files including __main__.py, weather.py, and test_weather.py. A message at the bottom states that the tool "weather" has been created and includes an entry point, main logic, and a test file.

```
Weather Tool Creation
+
create a tool named weather

Rule: create
when required to create a tool, an action, a prompt or a resource
When requested to create a tool, a resource or a prompt ask or
deduct the <name> and optionally the <package> (assume
<package> = <name> if not specified).
If they ask for a generic action consider it a tool.
Do not use pip import or a requirements.txt, assume you have
the required libraries
Use the command
ops a41 new <packagename> <description> (--prompt=<name>) --tool=
```

```
ops a41 new weather --tool=weather
node@36cc465a8d46:/workspaces/agent41-starter$ ops a41 new weather
NEW_POSTGRES=false
NEW_S3=false
NEW_REDIS=false
NEW_MILVUS=false
Using CPython 3.12.4 interpreter at: /home/node/.ops/linux-x64/bin/python
Creating virtual environment at: .venv
Type: tool
Package: weather
Name: weather
Description:
Options: {'redis': False, 'postgres': False, 'milvus': False}
Creating file packages/weather/weather/__main__.py
Creating file packages/weather/weather/weather.py
Creating file tests/weather/test_weather.py
node@36cc465a8d46:/workspaces/agent41-starter$
```

Ask every time Move to background

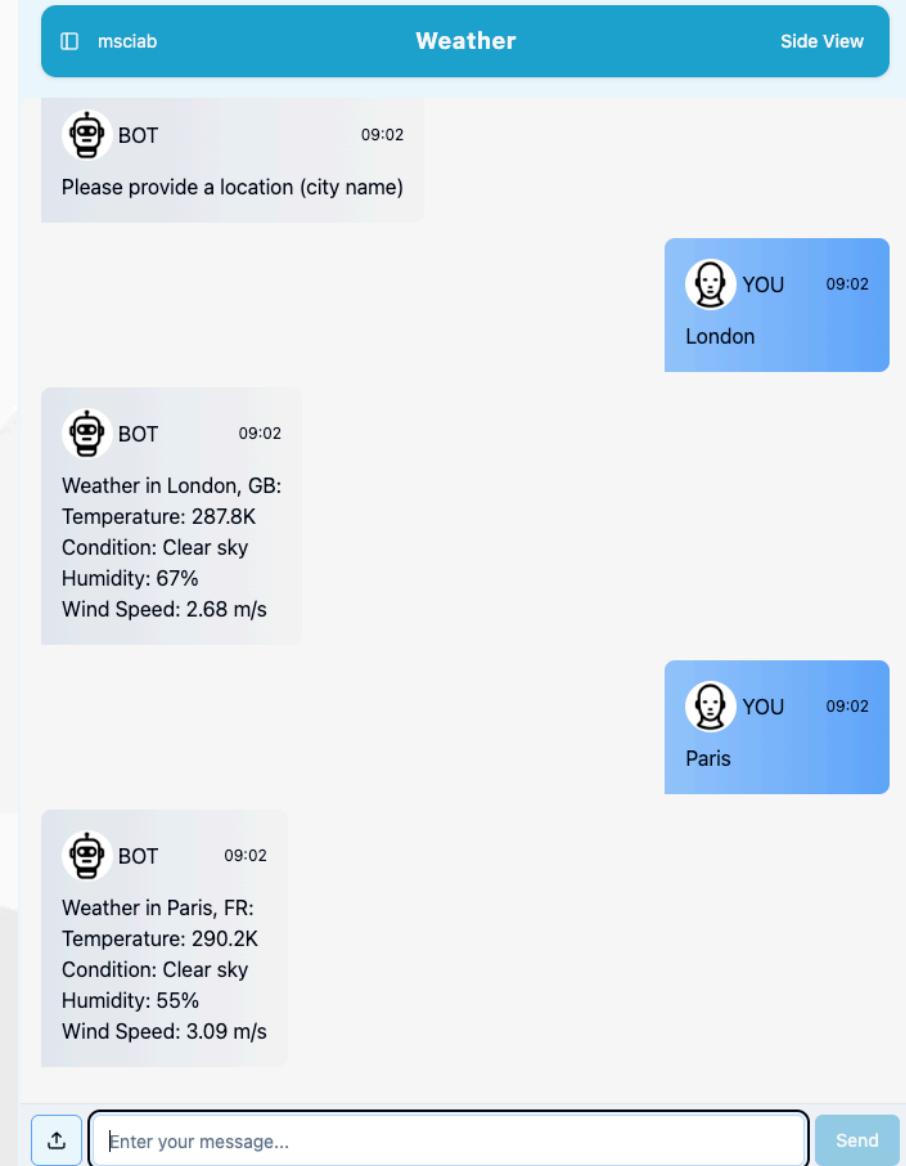
Read __main__.py
Read weather.py
Read test_weather.py

The tool named `weather` has been created. It includes:

- An entry point at `packages/weather/weather/__main__.py`
- The main logic in `packages/weather/weather/weather.py`
- A test file at `tests/weather/test_weather.py`

Implement Weather Tool

- read the api key
 - make a request to openweathermap and retrieve the weather for a given location
- add to the index
- test and deploy



Use Ollama to Parse Time

- write a function `ask_ollama` to ask using model `llama3.1:8b` and return its answer

- it should select or use the `ollama` rule

- write `parse_time_and_location` invoking `ask_ollama` assuming it is a request for a weather prediction at a given location and optionally a time. Return the answer as a tuple with 3 values: the location of the request, the time of the request as a number of hours in advance from now, and the number of days in advance. Default to 0 for both if not specified

Parse Time fixes and tests

- LLM does not know the current day and time we have to specify it
 - provide the current time and date to the prompt
- Sometimes the output is not a json object but is embedded:
 - check the response and extract a json object if it is embedded
 - in the answer
- We need to add tests:
 - write tests to check form London, London tomorrow moning,
 - Paris next Week and Moscow in the night

Close the loop fixing the `weather` function

Change the `weather` function to use the `parse_and_time` location and use always a forecast. In the result answer specify the date and time for the forecast in the local time.

write tests for London, London tomorrow, Paris next week