



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

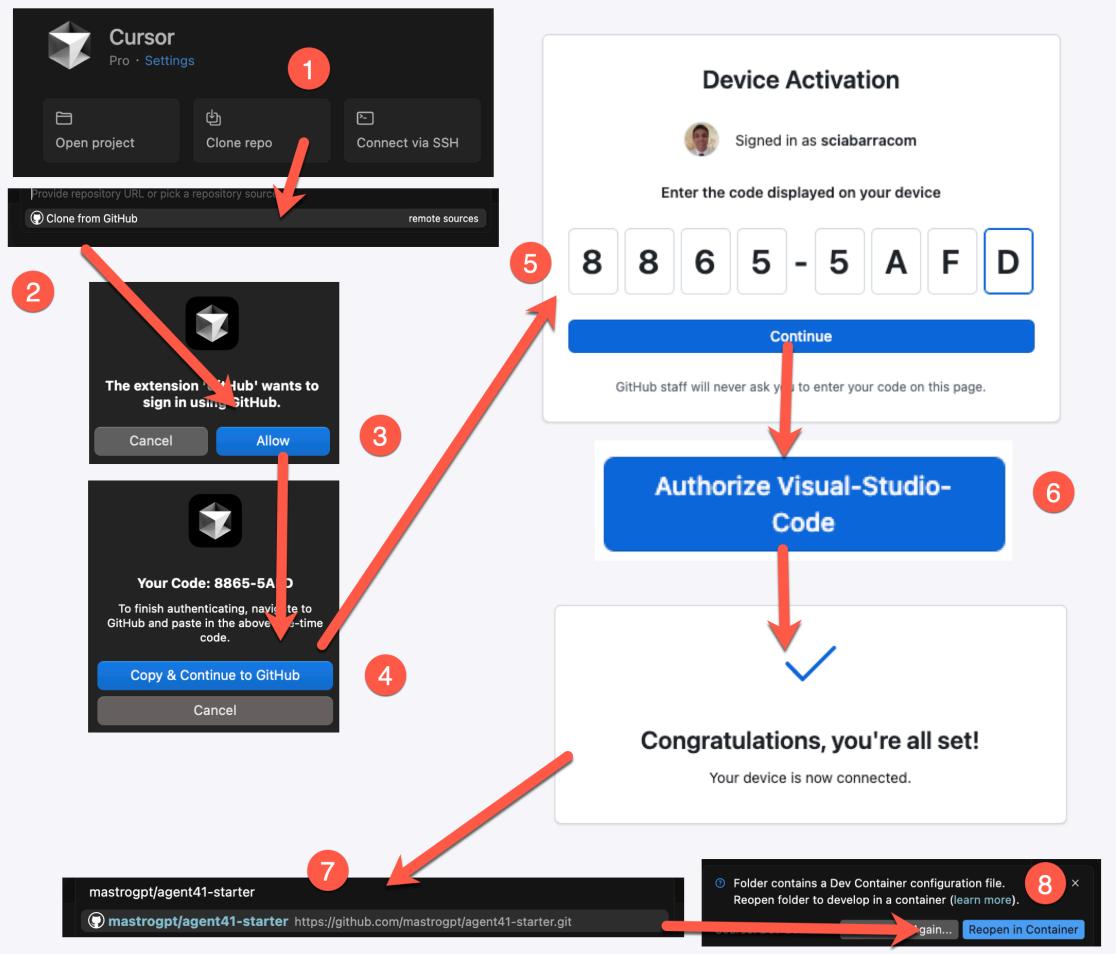
ALL DOWNLOADS

A screenshot of the Cursor AI Code Editor. The interface has a dark theme. At the top, there's a navigation bar with three dots, a title bar showing 'mod.rs' and a close button, and system status icons. Below the title bar is a toolbar with 'CHAT' and 'COMPOSER' buttons. The main area contains a code editor with some Rust code, a 'mod.rs' tab, and a message from the AI: 'Implement the cleanup function for the transport stack. Do not make the upgrade listeners optional.' To the right of the code editor is a 'mod.rs Current File' panel with another message: 'Could you make it easier to switch certificates in the transport listeners?'. At the bottom, there's a footer with the text 'Install Cursor: <https://cursor.com>' and a small '2' in the bottom right corner.

```
72
73
74
75
76 // [cfg(unix)]
77 upgrade_listeners: Option<ListenFds>,
78 upgrade_listeners: ListenFds,
79 }
80 impl TransportStack {
```

```
// ... existing code ...
/// The APIs to customize things like certificate during TLS ser
#[async_trait]
```

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`



The screenshot shows a VS Code interface with a file named `index.html` open. The code contains a script block that sends a message to a content window:

```
    type: 'metadata',
    data: { namespace: data.username }
};

target.contentWindow.postMessage(msg, targetOrigin);
```

Below the script, there's an `<iframe>` tag with a source URL:

```
<body>
  <iframe id="target" src="https://pinocchio.nuvolaris.org"></iframe>
</body>
```

A red arrow points from the URL in the `src` attribute of the `<iframe>` tag to the URL in the browser's login form.

The browser window on the right displays the Apache OpenServerless login page. It has fields for username ('msciab') and password ('.....'), and a highlighted URL field containing `https://nuvolaris.org`. A red arrow points from this URL field to the URL in the `src` attribute of the `<iframe>` tag in the VS Code editor.

Must Be the Same

The terminal window shows the following output:

```
Launching login script...
*** Configuring Access to OpenServerless ***
apihost=https://nuvolaris.org username=msciab
Logging in https://nuvolaris.org as msciab
Successfully logged in as msciab.
ok: whisk auth set. Run 'wsk property get --auth' to see the new value.
ok: whisk API host set to https://nuvolaris.org
OpenServerless host and auth set successfully. You are now ready to use ops!
```

1. Open AI Chat

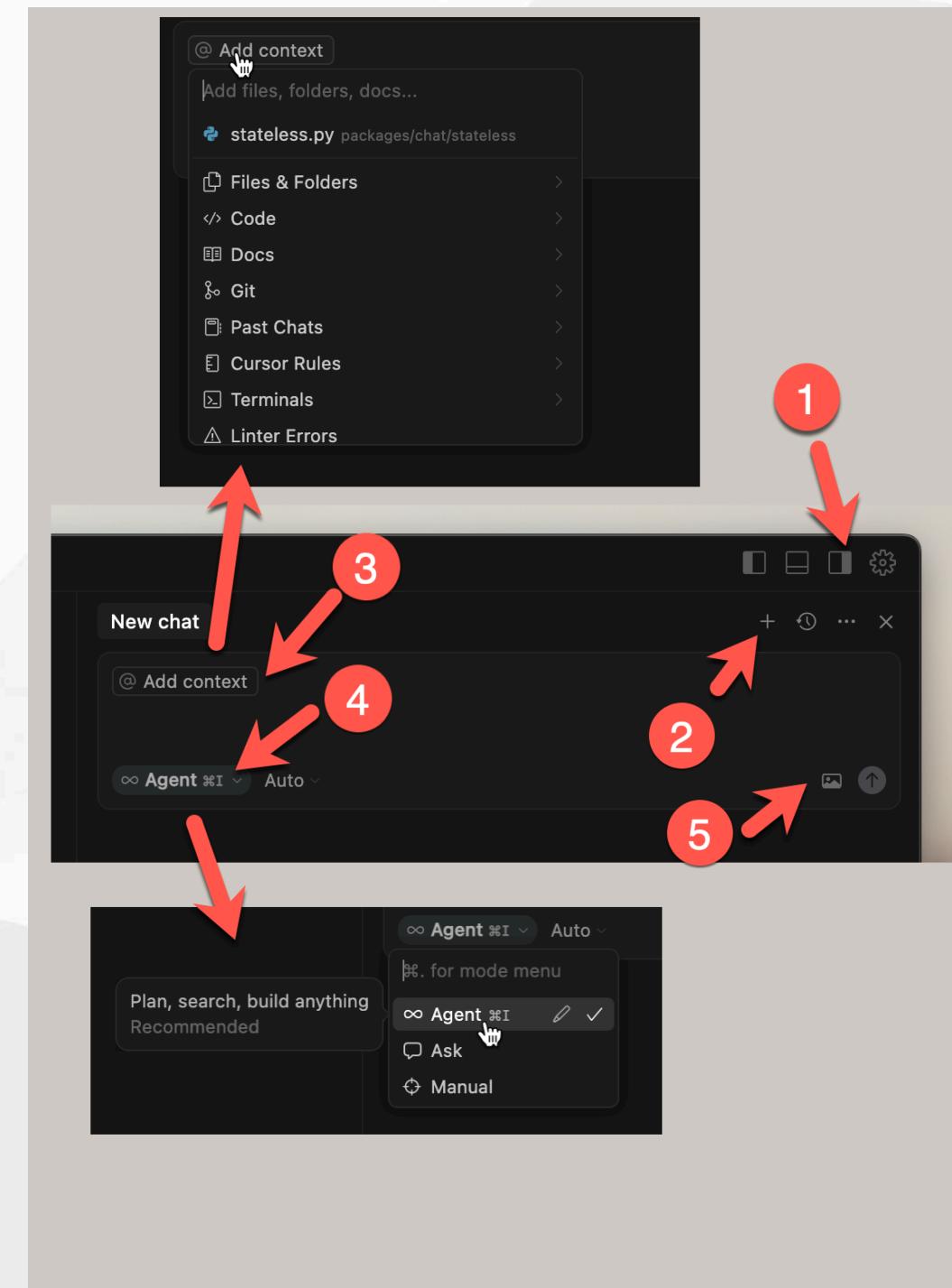
2. Create a New Chat

3. Select the context

- File & Folders
- Code
- More...

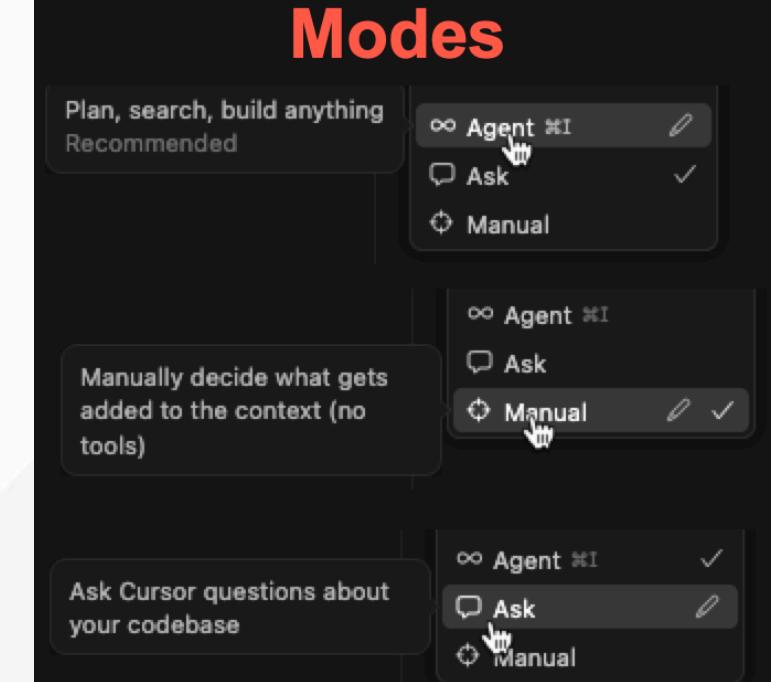
4. Select the Mode

- Agent
- Ask
- Manual

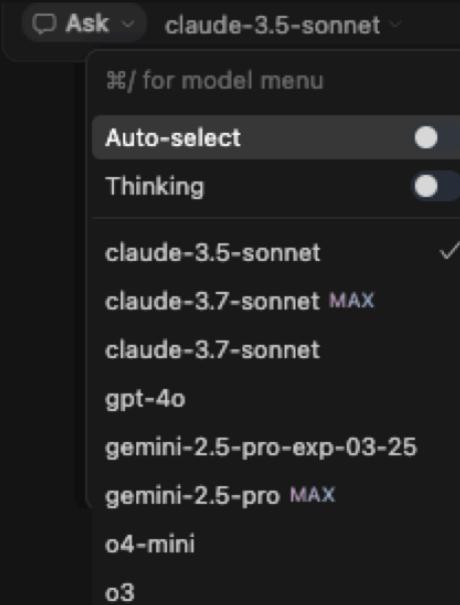


Modes

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

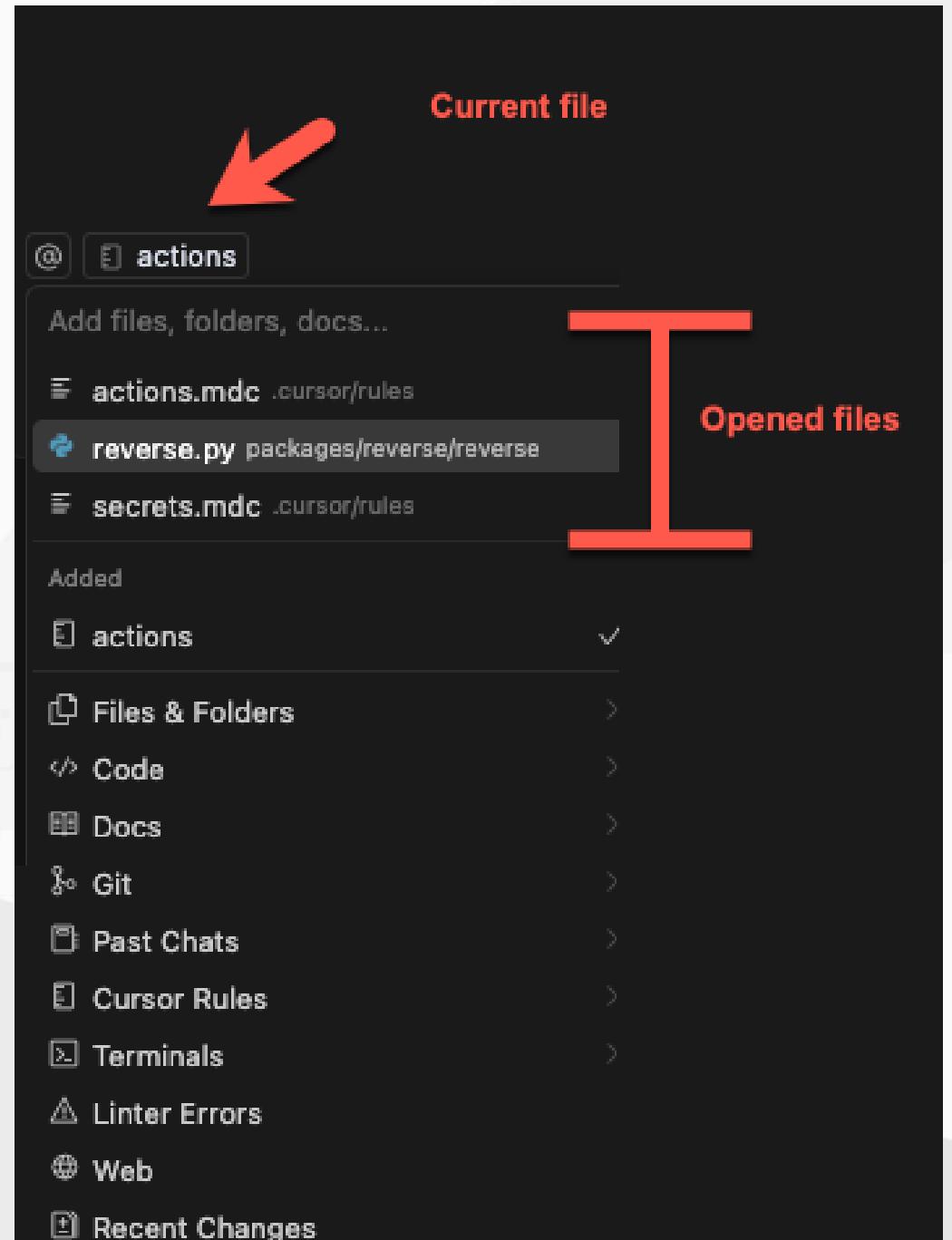


Models



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
| E actions.mdc U
| E secrets.mdc U
> .devcontainer ●
> packages ●
> site ●
> tests ●
> web
◆ .gitignore
E pytest.ini

Cursor Rules

.cursor > rules > E 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

```
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 <name> <description> (--prompt=<name>) --tool=weather

ops a41 new weather --tool=weather
node@36cc465a8d46:/workspaces/agent41-starter$ ops a41 new weather --tool=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

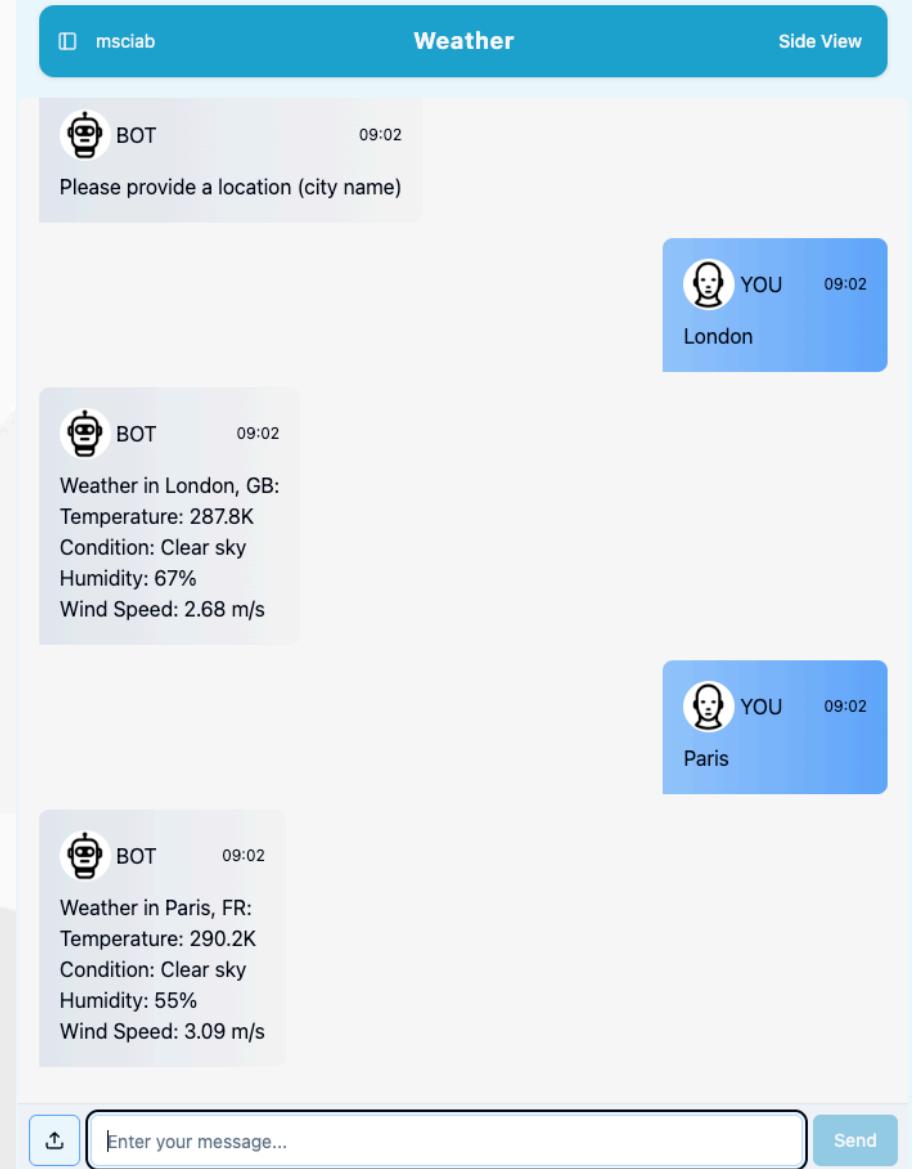
Read __main__.py
Read weather.py
Read test_weather.py

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