

Marimo: An Alternative to Jupyter Notebooks

Tech Talk | Feb 19, 2025

Presented by Chen Zhang



What is Marimo

- Marimo is a **reactive** Python notebook.
- Pure **Python** file (**git friendly**).
- UI components shipped with library (battery included).
- Easy deploy as a web app, similar to **streamlit** , **gradio** .

The screenshot shows the Marimo Notebook interface. On the left, there's a file tree with files like .pixi, InclusiveMinutes, layouts, TechTalk, .gitattributes, .gitignore, marimo_demo.py, pixi.lock, pixi.toml, and view_demo.py. The main area has two code cells at the top:

```
1 import marimo as m0
1 import numpy as np
1 import matplotlib.pyplot as plt
```

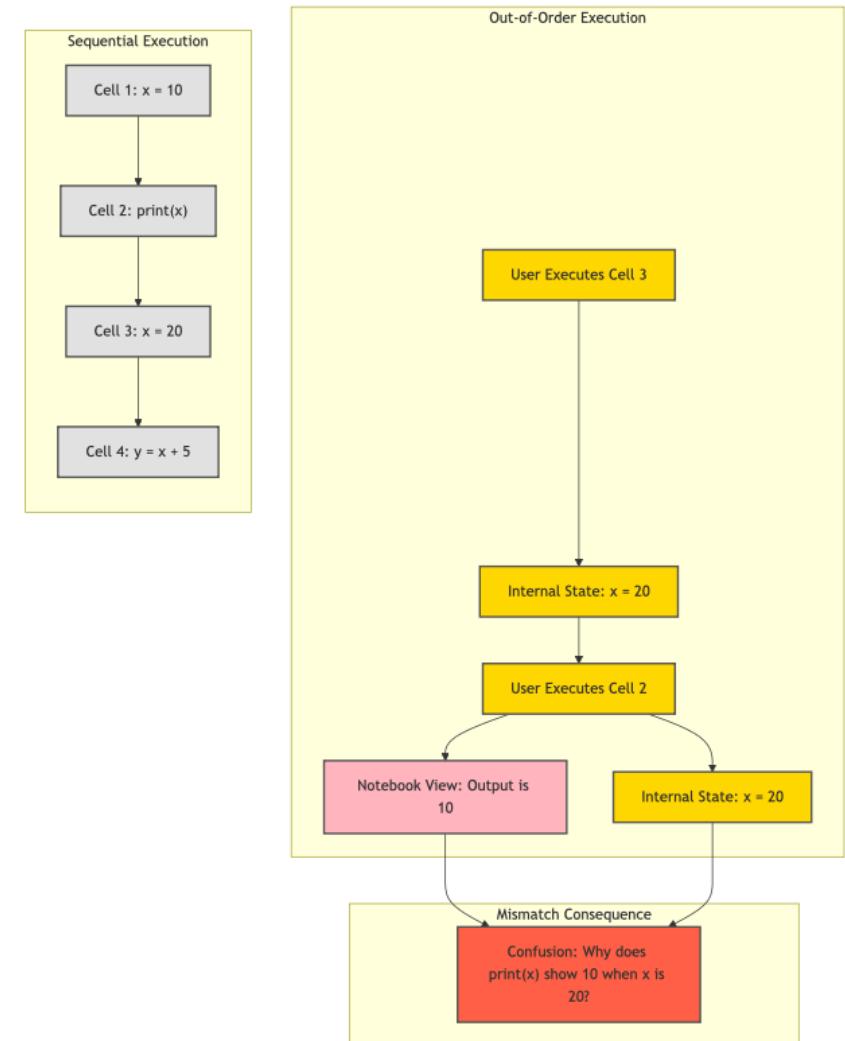
Below the cells is a text input field with placeholder text: "Generate a matplotlib plot of a sin wave with 100 data points, ranging from -pi to pi, return the figure object at the end". A large code block follows:1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def plot_sine_wave() -> plt.Figure:
5 """
6 Generate and plot a sine wave with 100 data points ranging
7 from -pi to pi.
8
9 Returns
10 -----
11 fig : plt.Figure
12 The figure object containing the plot.
13
14 Example
15 -----
16 >>> fig = plot_sine_wave()
17 >>> plt.gca().set_title("Sine Wave")
18
19 x = np.linspace(-np.pi, np.pi, 100)
20 y = np.sin(x)
21
22 fig, ax = plt.subplots()
23 ax.plot(x, y)
24
25 return fig
26
27 # Example usage:
28 # fig = plot_sine_wave()
29 # plt.gca().set_title("Sine Wave")

At the bottom, there are tabs for PYTHON, MARKDOWN, SQL, and GENERATE WITH AI, along with some status indicators.

Why Marimo?

Jupyter notebooks are great,
except for

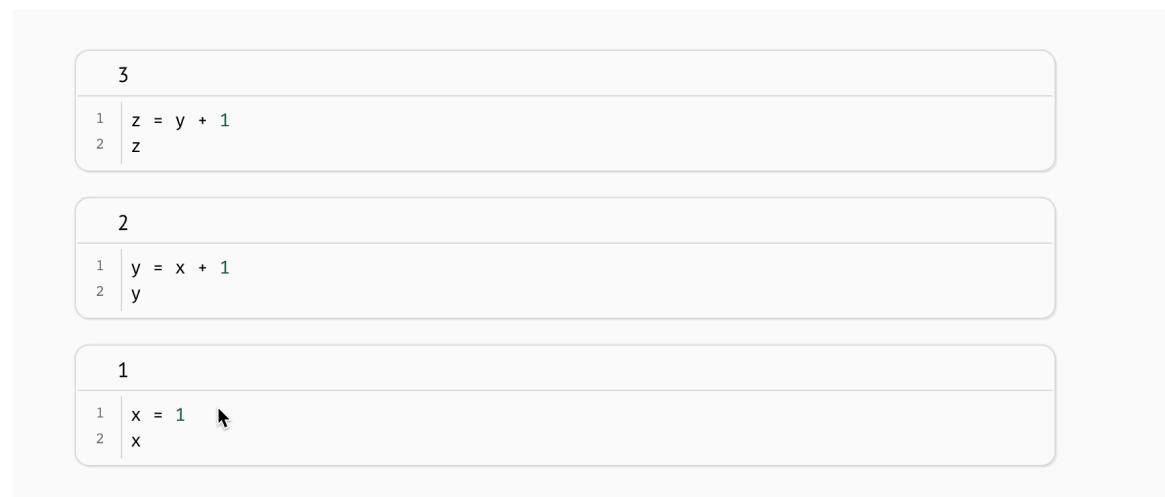
- Runtime/Display Mismatch
- Plugins management
- Not Git friendly



How Marimo helps

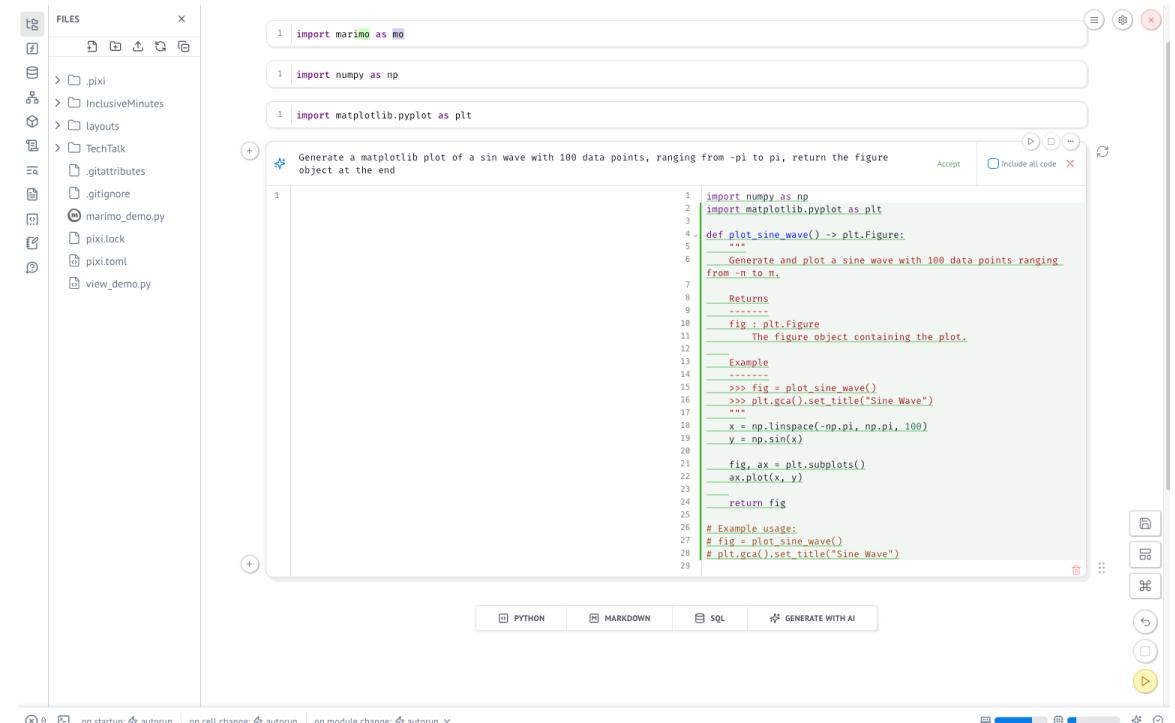
Reactive by design

- Each **cell** is a **function**, and dependencies between cells are automatically tracked and update via Directed Acyclic Graph (DAG) analysis.
- Global and local variables are managed by naming convention, all variables are global by default unless leading with `_`.



Marimo comes with UI components

- Common UI elements shipped with `marimo` library.
- Easy to deploy the notebook as web app built with `marimo` UI components.



Marimo is Git friendly

- Marimo notebook is a pure Python file.

```
import marimo

__generated_with = "0.10.13"
app = marimo.App(width="medium", layout_file="layouts/marimo_demo.slides.json")

@app.cell
def _():
    import marimo as mo
    return (mo,)

@app.cell
def _():
    import pandas as pd

    # Create a fake dataframe with three features
    data = {'Name': ['Alice', 'Bob', 'Charlie'],
            'Age': [24, 35, 42],
            'Gender': ['Female', 'Male', 'Non-binary']}
    df = pd.DataFrame(data)
    return data, df, pd

@app.cell
def _():
    return

if __name__ == "__main__":
    app.run()
```

Live demo

Setup Marimo

```
# init the project with pixi
pixi init
# install python (required)
pixi add python
# install node.js (required)
pixi add nodejs
# install marimo, need to pull from PyPI to enable the AI feature
pixi add --pypi "marimo[recommended]"
```

Given NDP is switching to `pixi` for Python environment management, we will be using `pixi` to manage the project.

AI integration with Marimo

- marimo has native support for Github's copilot and Codium AI for code completion.
- marimo can use both commercial LLM service (OpenAI, Claude, Gemini etc.) and local LLM services (e.g. Ollama, LMStudio, etc.)

```
# Install Ollama and pull a model from huggingface [skipped]
# Start the Ollama service [skipped, Ollama server is started in the background on this machine]
# Start the Model server
ollama run phi4
```

Once the prompt returns, we can start the marimo notebook with

```
pixi run marimo edit demo.py
```

marimo has two modes: edit mode is similar to Jupyter notebook where one interactively develop the notebook; run mode is starting the notebook as a webapp, similar to streamlit .

Setup example case

Prompt to use

- Generate Python code to create a pandas DataFrame with 100 rows and 10 features. The features should include a mix of numeric, categorical, and datetime types. Include realistic data values and randomization for variability.
- Generate a cumulative distribution plot for `@df Numeric1`, using `matplotlib`, return the figure object at the end.
 - When a library is missing, `marimo` will ask to install the library on your behalf, restart the server and rerun the entire notebook.
 - `marimo` requires an interactive object handle to display, therefore we need to add `plot.show()` at the end to view the graph.

Adding more code

- Add a new feature to @df to update it, new feature is a combination (add) of Numeric2 and Numeric3, and name it as `2+3`.

Local model generally do not have the hidden agent like ChatGPT and Claude.ai, therefore you will need to adjust the code to make it work.

Global var v.s. Local var

- marimo uses DAG to track var update and dependencies, therefore unique name is required for all global vars to avoid conflict.
- Local vars are identified by leading `_`, and are not tracked by DAG.

```
# cell 1
a = 1
_b = 2
# cell 2
a = 3 # this will lead to error
_b = 4 # this is fine
```

This check only works for native Python vars, if you are using a `dataframe`, which can be modified in-place even passed as a function argument, you are responsible for tracking the changes.

Example UI elements

Most of the LLM services have not learned the `marimo` UI elements, therefore you will need to manually write the code by referring to the [official documentation](#).

```
# in one cell
slider = mo.ui.slider(start=0, stop=10)

# in another cell
mo.vstack([
    slider,
    mo.md(f'Hello, islands! {"🌴" * int(slider.value)}'),
])

```

Q&A

