Getting started with marimo

Python notebooks. A notebook is a type of program in which code is organized in a sequence of code blocks, called *cells*. A Python notebook is a notebook in which each cell consists of Python code.

Cells are typically displayed in a graphical user interface, or editor, that lets you edit their code and execute them interactively. Unlike traditional Python scripts, which run from start to finish without human intervention, notebooks allow you to execute a section of code and inspect the execution results, at which point you may choose to execute another cell, modify a cell's code (including the one you just executed), or otherwise author entirely new cells. In this way notebooks allow you to author programs incrementally, using intermediate execution results as feedback for what to author next, which can be of great help when working with data.

Outputs. The last expression of a cell is its *output*, which is visualized by the editor below the cell. Outputs can be arbitrary Python objects. When working through this companion, you will frequently output plots, matrices, and other data structures.

For example, to visualize a matplotlib plot, write

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> plt.plot(np.array([0, 1, 2]))
>>> # get the current axis and output it
>>> plt.gca()
```

Markdown. In addition to Python, notebooks allow you to document your work with *Markdown*. Markdown is a simple markup language for creating formatted text. It is common to intersperse cells with expository Markdown that explains work or demarcates sections. Markdown is very widely used, and there are many free tutorials available online that teach its syntax. Nearly all Markdown implementations in notebooks also allow you to write mathematical notation with LaTeX.

Documents. A notebook, consisting of cells, outputs, and Markdown, can be interpreted as a *document*, in addition to a program. Notebook editors allow you

to export your notebook as a standalone HTML or PDF file that can be shared with others.

The marimo notebook

In this companion we recommend using the marimo notebook. Marimo is an open-source notebook designed specifically for the Python language. Similar to other modern language-specific notebooks, like Pluto.jl for the Julia programming language (from which marimo draws significant inspiration), marimo provides immediate feedback when cells are run through *reactive execution* while also solving key problems with traditional notebooks like Jupyter. Roughly speaking, reactive execution means that marimo's execution model is more similar to a spreadsheet than a read-eval-print loop.

Marimo and its documentation are freely available at

https://docs.marimo.io

The online documentation contains a quickstart, comprehensive user guides, and several examples.

Setup

Installation. To install marimo in your Python project, from your terminal and in your project directory, run

```
uv add marimo
```

To upgrade marimo to the latest version, run

```
uv lock --upgrade-package marimo
```

To upgrade marimo to a specific version, run

```
uv lock --upgrade-package marimo==<version>
```

VS Code extension. You can edit and run marimo notebooks in VS Code using the official marimo extension. Install the extension by opening the Extensions view (`Ctrl/Cmd+Shift+X`) and searching for `marimo`. This extension is experimental but under very active development.

To create a new notebook, open the command palette with `Ctrl+Shift+P`, then type 'new marimo notebook' to find the command. Make sure to save the notebook after you create it. Marimo notebooks are just Python programs, and are saved with the `.py` extension.

To open an existing notebook, select the Python file from the VS Code file browser and use the 'Open as marimo notebook' command palette action.

Command-line interface and browser-based editor. An alternative to the VS Code extension is marimo's browser-based editor, which can be opened through the command line. The browser-based editor is more fully featured than the VS Code extension, but has the drawback of not being accessible through VS Code. Which editor you choose comes down to personal preference.

To open the browser-based editor, in your project directory run

```
uv run marimo edit
```

This opens a tab in your browser that lets you create new notebooks or open existing ones.

Cloud-hosted notebooks. As an alternative to running marimo locally, you may optionally choose to use marimo's free cloud-hosted notebook service, available at

https://molab.marimo.io/

Notebooks created on molab are public but not discoverable by default, and can be shared with others by URL.

Built-in tutorials. Marimo comes packaged with interactive tutorials. To open the introductory tutorial, run

```
uv run marimo tutorial intro
```

from the command line.

To see a list of all tutorials, run

```
uv run marimo tutorial --help
```

Concepts

Reactive execution. When a cell is run, marimo reacts by automatically running all other cells that reference any of the variables it defines. This reactive execution keeps code and outputs in sync, eliminating a large class of silent bugs associated with traditional notebooks while also facilitating rapid data exploration. (For notebooks with long-running cells, you may choose to configure marimo to run cells lazily, marking them as stale instead of automatically running them, through the notebook settings.)

Unique variable definitions. In order to implement reactive execution, marimo imposes one constraint on your code: each variable must be defined in just one cell. If you accidentally define a variable in two (or more) cells, marimo will raise an error. For this reason, when using marimo you should define global variables sparingly, wrapping intermediate variable definitions in functions when possible. This also results in better code.

Marimo's rule on variable naming has one exception. Variables that begin with an underscore are made *local* to a cell, and as such their names can be bound in multiple cells. This is helpful for loop variables, which for example may be written as

```
>>> for _i in range(10):
...   print(_i)
```

User interface elements. Marimo comes packaged with user interface (UI) elements like numerical sliders, dropdowns, and selectable scatterplots, and more. To create a UI element, you must first import the `marimo` library into your notebook.

```
>>> import marimo as mo
```

UI elements are exposed through `marimo`'s `ui` module. Create a UI element assign it to a variable, and output it in the same cell.

```
>>> x = mo.ui.slider(start=1, stop=10, step=1, label="$x$")
>>> x
```

Access its associated value through the `value` attribute in any cell other than the one defining the UI element.

```
>>> x.value
```

When you interact with a UI element (for example, sliding the knob of a slider), all other cells that reference the UI element will run automatically, thanks to reactive execution.