UI Elements

UI Elements

One of marimo's most powerful features is its first-class support for interactive user interface (UI) elements: interacting with a UI element will automatically run cells that reference it.

`marimo.ui`

How interactions run cells

Whenever you interact with a UI element, its value is sent back to Python. When this happens, all cells that reference the global variable bound to the UI element, but don't define it, will run.

This simple rule lets you use UI elements to drive the execution of your program, letting you build interactive notebooks and tools for yourselves and others.

Simple elements

State

" " **Heads up!**

```
The rest of this tutorial covers state, an advanced topic. Feel free
to return here later, if or when you find yourself
limited in building interactive stateful apps.
"""
```

).callout(kind="warn

You can build powerful interactive notebooks and apps using just `mo.ui` and reactivity.

Sometimes, however, you might want interactions to mutate **state**. Maybe you're building a checklist, and you want to maintain a list of action items. Or maybe you want to tie two different UI elements, so that updating one updates the other.

For these and other cases, marimo provides the function `mo.state`, which creates state returns a getter function and a setter function. When you call the setter function in one cell, all other cells that reference the getter via a global variable are automatically run (similar to UI elements).

Creating state

`mo.state` takes an initial state value as its argument, and returns

- a function that returns the state value;
- a function that updates the state value.

For exaxmple,

```
get_counter, set_counter = mo.state(0)
```

Setting State

Set an element's state by calling its setter function.

- Call it with a new value: `set_counter(1)`
- Call it with a function that takes the current value and returns a new one: `set_counter(lambda count: count + 1)`

State updates are reactive. When you call a state's setter in one cell, *all other cells that reference the state getter via a global variable* are automatically run with the new state value. This is similar to how interacting with a UI element automatically runs all cells that use the element.

The `on_change` callback. Every UI element takes an optional `on_change` callback, a function that takes the new value of the element and does anything with it. You can use the setter function in an `on_change` callback to mutate state.

Try it! Click the button below and watch what happens.

Tied elements

Use state to tie two UI elements to the same value.

Example: Task list

Appendix

The remaining cells are helper data structures and functions. You can look at their code if you're curious how certain parts of this tutorial were implemented.