Introduction

Reflex is an open-source framework for quickly building beautiful, interactive web applications in **pure Python**.

Goals

Pure Python

Use Python for everything. Don't worry about learning a new language.

Easy to Learn

Build and share your first app in minutes. No web development experience required.

Full Flexibility

Remain as flexible as traditional web frameworks. Reflex is easy to use, yet allows for advanced use cases.

Build anything from small data science apps to large, multi-page websites. **This entire site was built and deployed with Reflex!**

Batteries Included

No need to reach for a bunch of different tools. Reflex handles the user interface, server-side logic, and deployment of your app.

An example: Make it count

Decrement 0 Increment

Here is the full code for this example:

```python
import reflex as rx


class State(rx.State):
    count: int = 0

    def increment(self):
        self.count += 1

    def decrement(self):
        self.count -= 1


def index():
    return rx.hstack(
        rx.button(
            "Decrement",
            color_scheme="ruby",
            on_click=State.decrement,
        ),
        rx.heading(State.count, font_size="2em"),
        rx.button(
            "Increment",
            color_scheme="grass",
            on_click=State.increment,
        ),
        spacing="4",
    )


app = rx.App()
```

ırn
mponents
l Reference
ding
oduction
allation
ect Structure
figuration
ew
ew

The Structure of a Reflex App

Let's break this example down.

Import

```
import reflex as rx
```

We begin by importing the `reflex` package (aliased to `rx`). We reference Reflex objects as `rx.*` by convention.

State

```
class State(rx.State):
    count: int = 0
```

The state defines all the variables (called **vars**) in an app that can change, as well as the functions (called **event_handlers**) that change them.

Here our state has a single var, `count`, which holds the current value of the counter. We initialize it to `0`.

Event Handlers

```
def increment(self):
    self.count += 1
```

Goa
An e
The
Imp
Stat
Ever
User
Add
Nex

Within the state, we define functions, called **event handlers**, that change the state vars.

Event handlers are the only way that we can modify the state in Reflex. They can be called in response to user actions, such as clicking a button or typing in a text box. These actions are called **events**.

Our counter app has two event handlers, `increment` and `decrement`.

User Interface (UI)

```
def index():
    return rx.hstack(
        rx.button(
            "Decrement",
            color_scheme="ruby",
            on_click=State.decrement,
        ),
        rx.heading(State.count, font_size="2em"),
        rx.button(
            "Increment",
            color_scheme="grass",
            on_click=State.increment,
        ),
        spacing="4",
    )
```

This function defines the app's user interface.

We use different components such as `rx.hstack`, `rx.button`, and `rx.heading` to build the frontend. Components can be

started. We are actively adding more components. Also, it's easy to wrap your own React components .

```
rx.heading(State.count, font_size="2em"),
```

Components can reference the app's state vars. The `rx.heading` component displays the current value of the counter by referencing `State.count` . All components that reference state will reactively update whenever the state changes.

```
rx.button(
    "Decrement",
    color_scheme="ruby",
    on_click=State.decrement,
),
```

Components interact with the state by binding events triggers to event handlers. For example, `on_click` is an event that is triggered when a user clicks a component.

The first button in our app binds its `on_click` event to the `State.decrement` event handler. Similarly the second button binds `on_click` to `State.increment` .

In other words, the sequence goes like this:

- User clicks "increment" on the UI.
- `on_click` event is triggered.
- Event handler `State.increment` is called.
- `State.count` is incremented.
- UI updates to reflect the new value of `State.count` .

base route.

```
app = rx.App()
app.add_page(index)
```

Next Steps

🎉 And that's it!

We've created a simple, yet fully interactive web app in pure Python.

By continuing with our documentation, you will learn how to building awesome apps with Reflex.

For a glimpse of the possibilities, check out these resources:

- For a more real-world example, check out the tutorial .

Installation →