NIHR | Applied Research Collaboration South West Peninsula

@penCHORD_UoE
@peninsula_ARC

Phase 2: Materclass – Shiny for Python
Session on PyShiny
Elliott Coyne

"A Great Way to Fly"

HSMA 5

#hsma5isalive

What we’ll learn this morning...

- What is Shiny for Python and Why Use It?
- What is an interactive Dashboard/ App?
- Examples of Deployed PyShiny Apps
- Anatomy of a Shiny app
- Installing Shiny for Python with VSCode
- Principles of how it works?
- Decorators
- Differences with Streamlit?
- Examples and Exercise

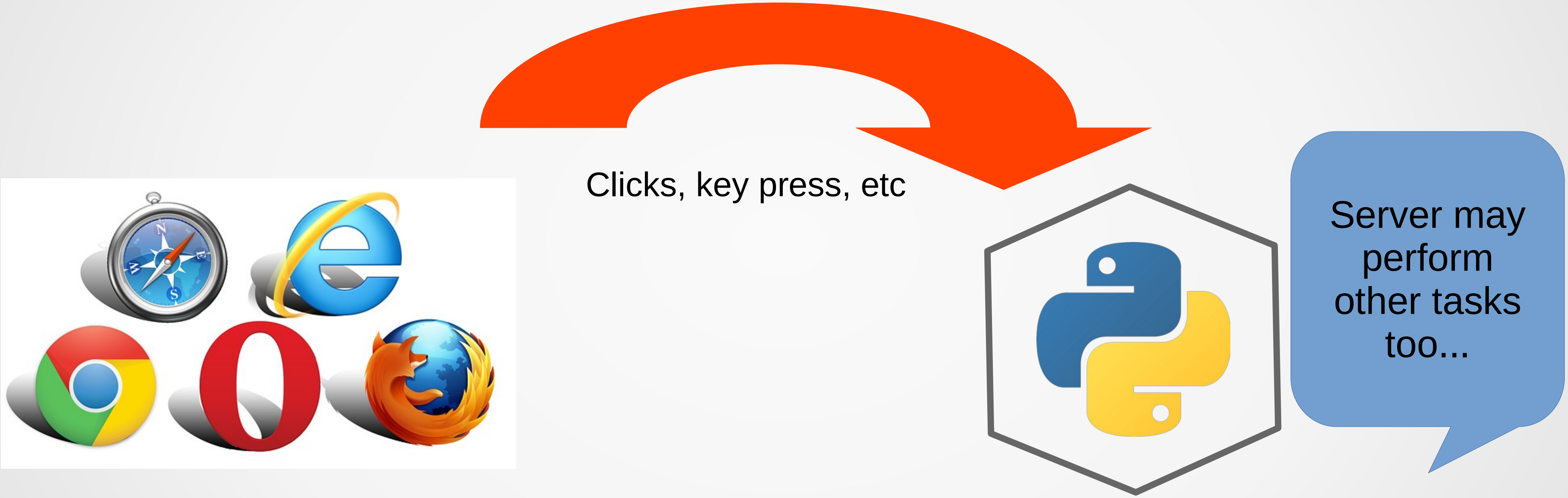
What is Shiny for Python?

“Interactive apps and dashboard made easy*ish*”
Joe Cheng, CTO, Posit (aka RStudio)

What is Shiny for Python and Why Use It?

- Shiny was originally released for R circa 2012
- Shiny for Python (aka PyShiny) released (as Alpha) in 2022
- Provides an framework for creating interactive apps and dashboard in Python
- No need for prior knowledge of HTML, CSS, or JavaScript!
  - However knowledge will allow you to enhance your apps
- Compatible with Pandas, NumPy and scikit-learn. Plus integrates with Matplotlib, Seaborn, Plotnine, Plotly, etc.
- Similarity with R Shiny (for those familiar) learn more **HERE**

What is an interactive Dashboard/ App?



Input

Clicks, key press, etc

Server may perform other tasks too...

Browser

Output

Server

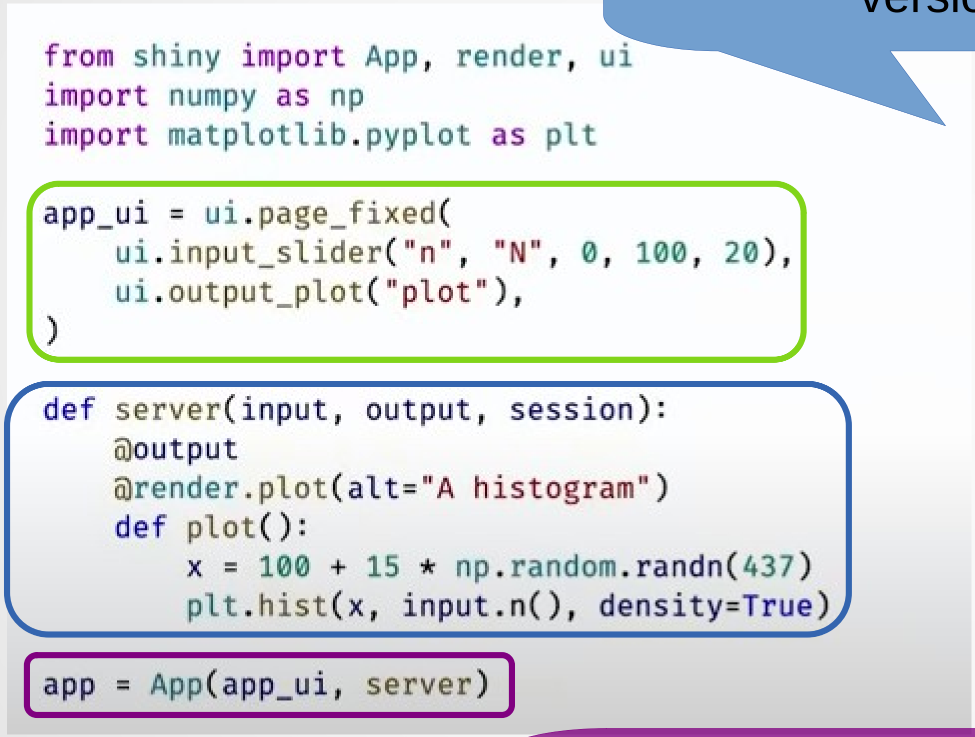
Plots, tables,
UI Change

Examples of Deployed PyShiny Apps

- Respiratory Disease Data (HERE)
- Simulate Data for a t-test (HERE)

Anatomy of a Shiny app

But what does this look like? To VS Code and web versions



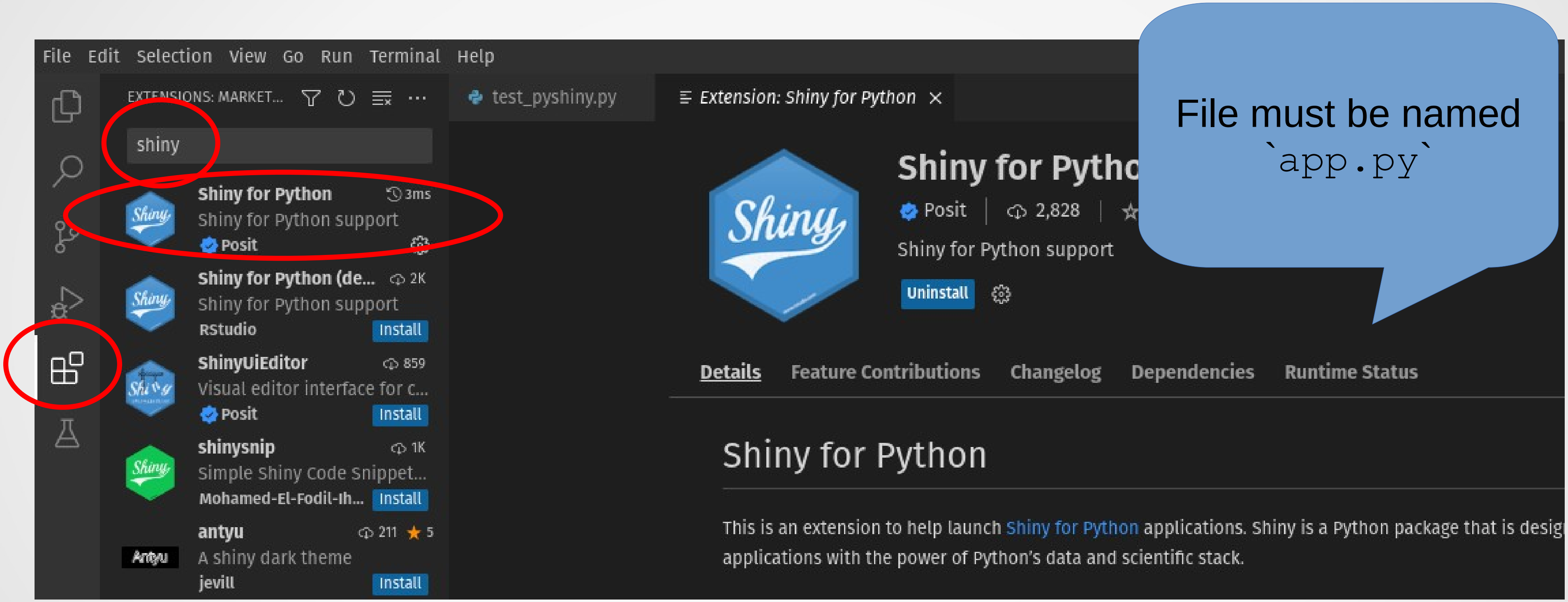
```
from shiny import App, render, ui
import numpy as np
import matplotlib.pyplot as plt

app_ui = ui.page_fixed(
    ui.input_slider("n", "N", 0, 100, 20),
    ui.output_plot("plot"),
)

def server(input, output, session):
    @output
    @render.plot(alt="A histogram")
    def plot():
        x = 100 + 15 * np.random.randn(437)
        plt.hist(x, input.n(), density=True)

app = App(app_ui, server)
```

User Interface
Generates HTML to send to a web-browser

Server Logic
Runs server, which provides interactivity

Union
Bringing together these 2 elements

Installing Shiny for Python with VSCode

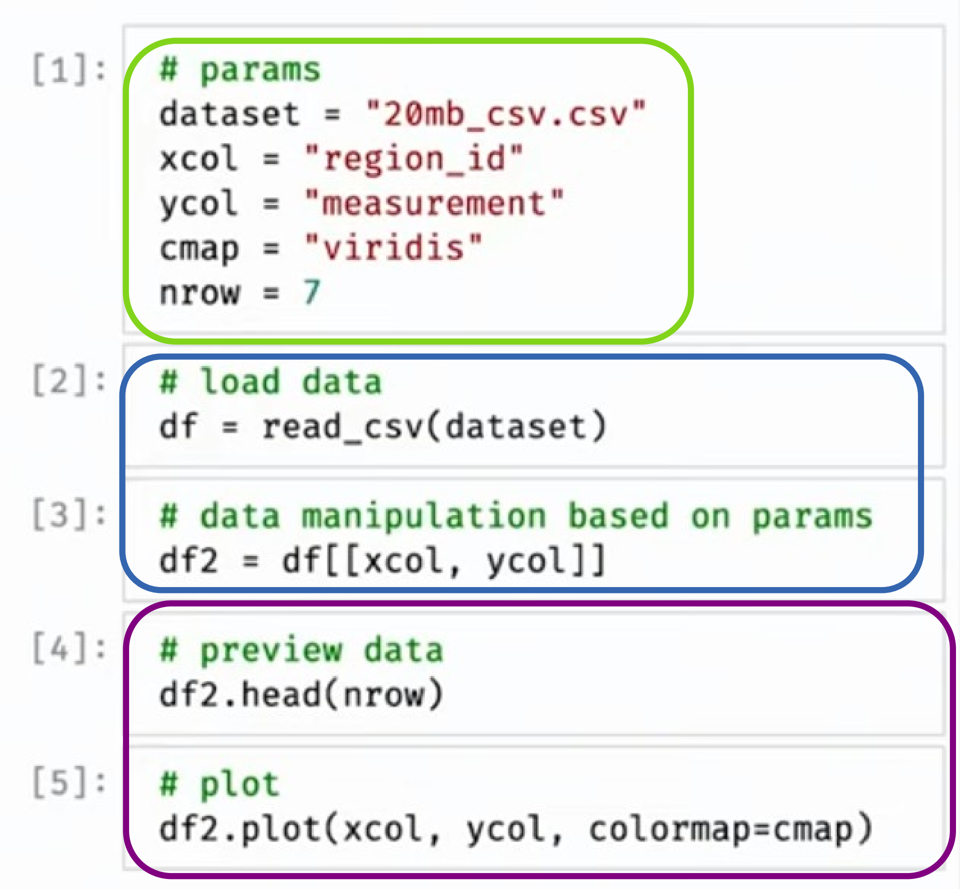


Shiny for Python

This is an extension to help launch Shiny for Python applications. Shiny is a Python package that is desig applications with the power of Python's data and scientific stack.

File must be named `app.py`

Principles of how Shiny for Python works?



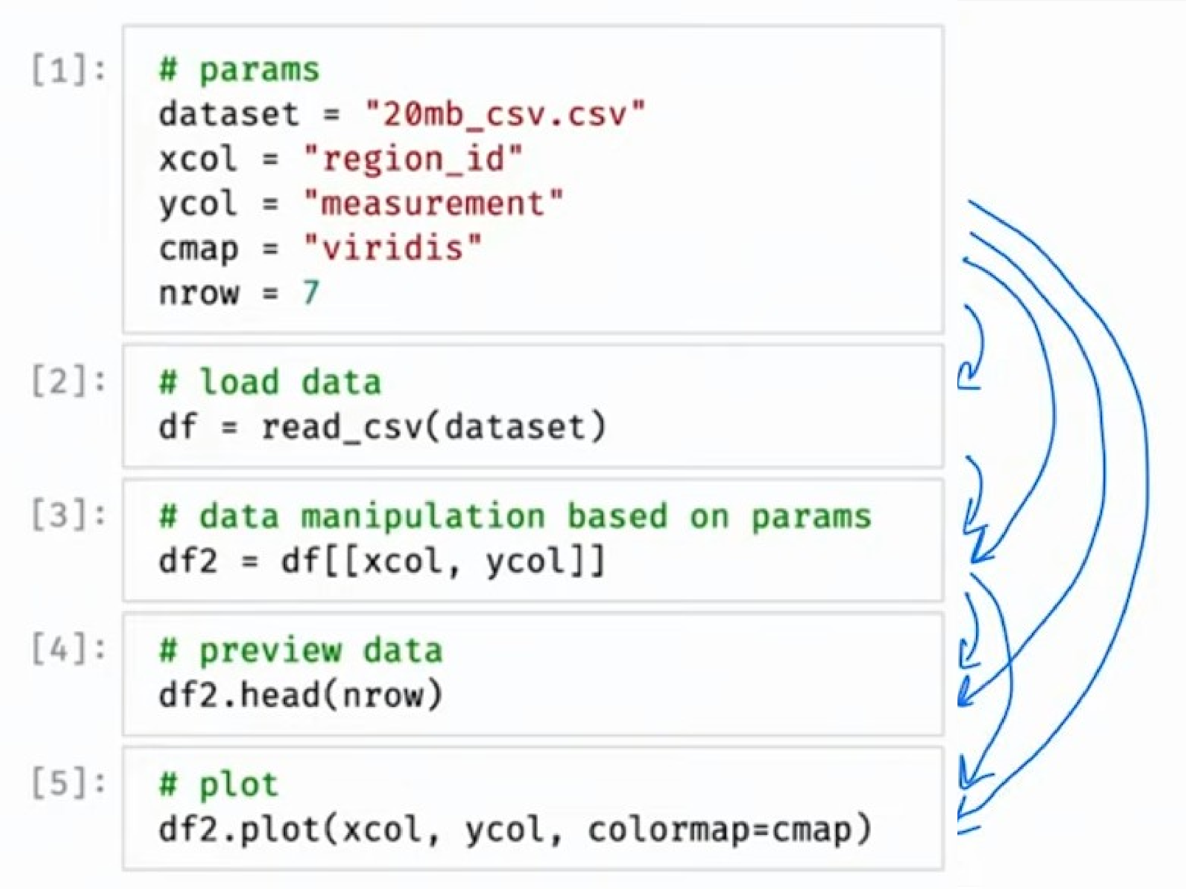
```
[1]: # params
     dataset = "20mb_csv.csv"
     xcol = "region_id"
     ycol = "measurement"
     cmap = "viridis"
     nrow = 7

[2]: # load data
     df = read_csv(dataset)

[3]: # data manipulation based on params
     df2 = df[[xcol, ycol]]

[4]: # preview data
     df2.head(nrow)

[5]: # plot
     df2.plot(xcol, ycol, colormap=cmap)
```

Inputs When these things change...

Intermediate values

Outputs … these outputs reflect the changes

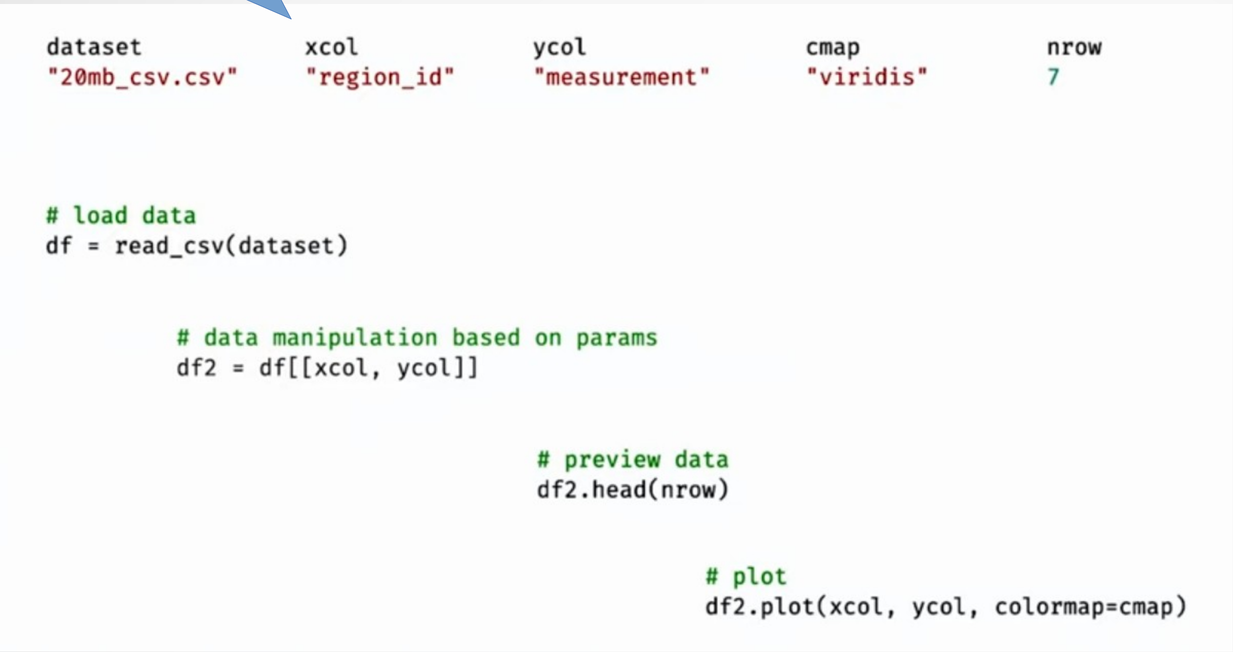
Principles of how it works?



```
[1]: # params
     dataset = "20mb_csv.csv"
     xcol = "region_id"
     ycol = "measurement"
     cmap = "viridis"
     nrow = 7

[2]: # load data
     df = read_csv(dataset)

[3]: # data manipulation based on params
     df2 = df[[xcol, ycol]]

[4]: # preview data
     df2.head(nrow)

[5]: # plot
     df2.plot(xcol, ycol, colormap=cmap)
```

Lets look at it another way...

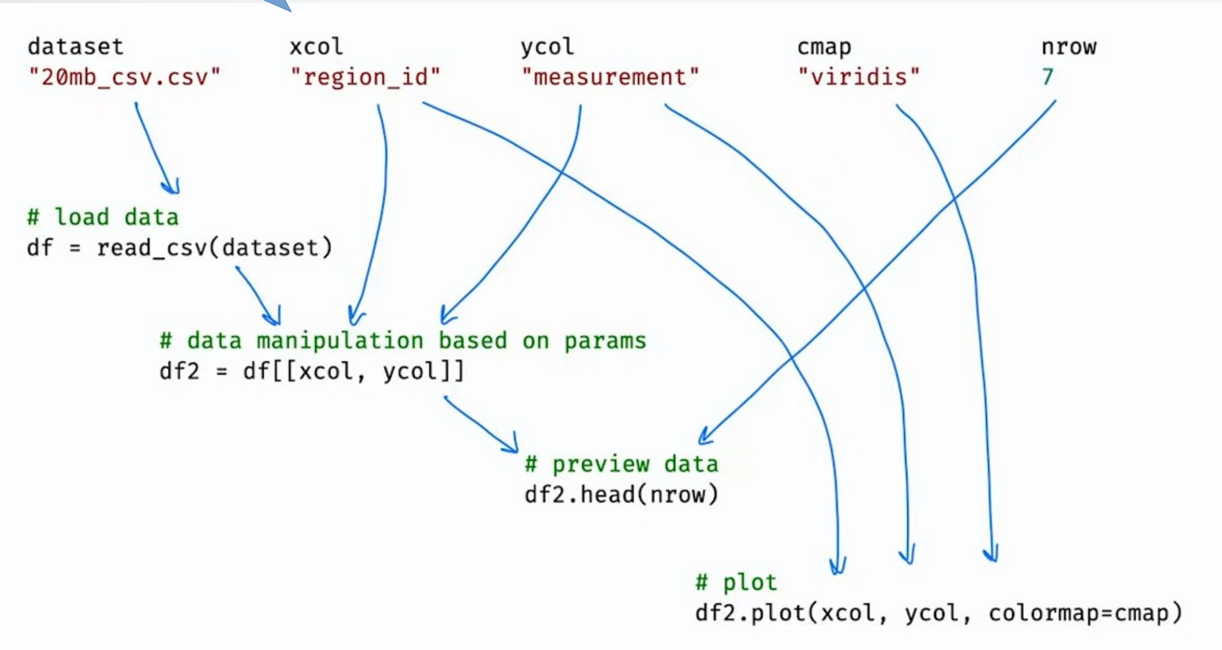
Params at the top...



```
dataset          xcol          ycol            cmap          nrow
"20mb_csv.csv"   "region_id"   "measurement"   "viridis"     7



# load data
df = read_csv(dataset)


         # data manipulation based on params
         df2 = df[[xcol, ycol]]


                                 # preview data
                                 df2.head(nrow)


                                             # plot
                                             df2.plot(xcol, ycol, colormap=cmap)
```

Lets look at it another way...

Params at the top...

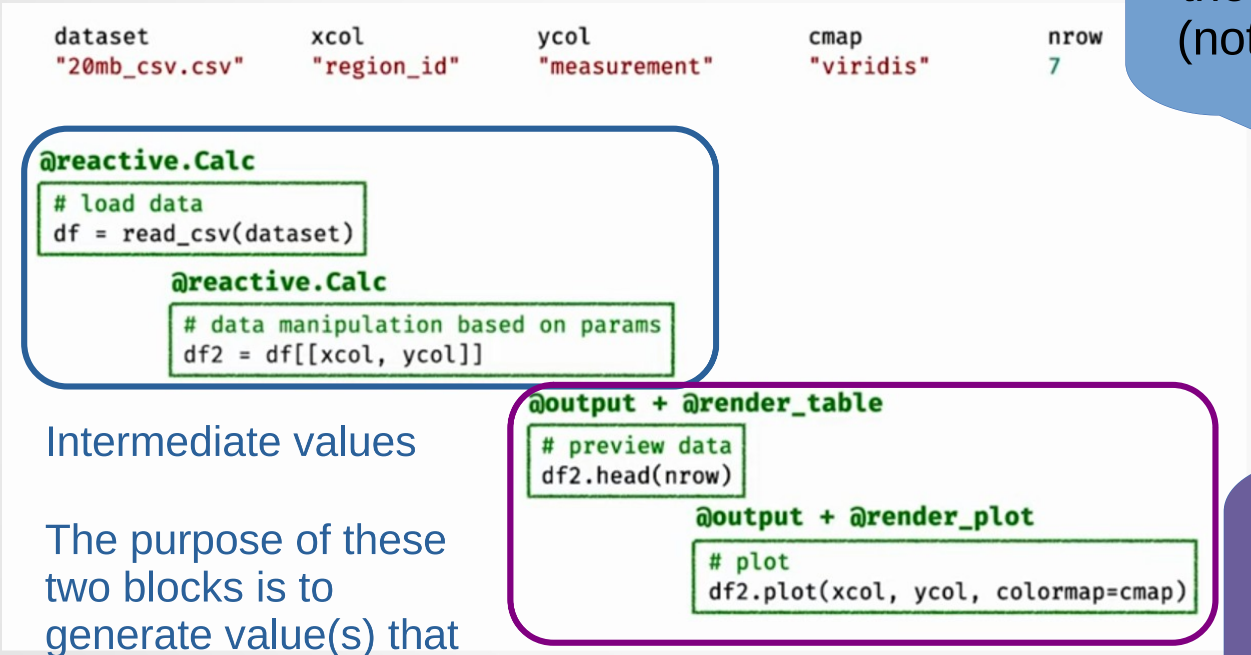


```
dataset            xcol            ycol                cmap              nrow
"20mb_csv.csv"     "region_id"     "measurement"       "viridis"         7

# load data
df = read_csv(dataset)

         # data manipulation based on params
         df2 = df[[xcol, ycol]]

                                    # preview data
                                    df2.head(nrow)

                                                # plot
                                                df2.plot(xcol, ycol, colormap=cmap)
```

Lets look at it another way...

With Shiny you just need to maintain these chunks of code (not the relationships)

Divide into chunk that makes sense...



```
dataset          xcol          ycol           cmap        nrow
"20mb_csv.csv"   "region_id"   "measurement"  "viridis"   7

@reactive.Calc
# load data
df = read_csv(dataset)
        @reactive.Calc
        # data manipulation based on params
        df2 = df[[xcol, ycol]]

                           @output + @render_table
                           # preview data
                           df2.head(nrow)
                                   @output + @render_plot
                                   # plot
                                   df2.plot(xcol, ycol, colormap=cmap)
```

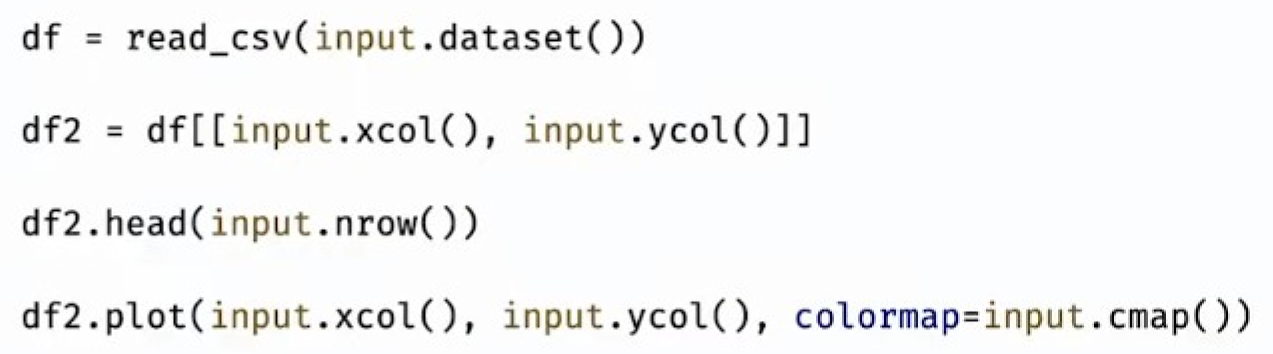
Intermediate values

The purpose of these two blocks is to generate value(s) that will be used elsewhere...

For each of these two outputs we need to say what kind of output they will be

Provide a decorator to explain the purpose of the chunk

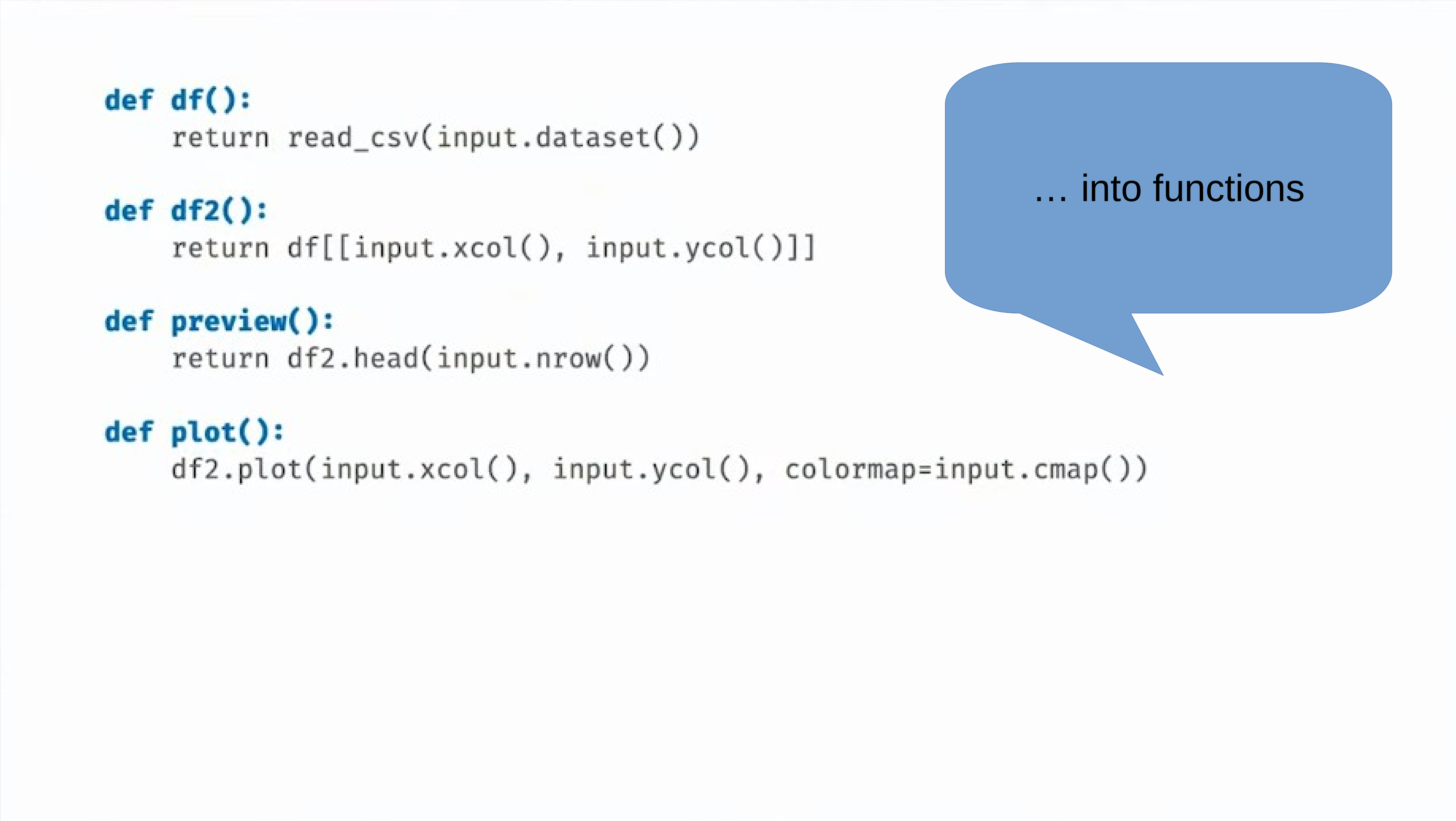
Lets get Shiny...



```
df = read_csv(input.dataset())

df2 = df[[input.xcol(), input.ycol()]]

df2.head(input.nrow())

df2.plot(input.xcol(), input.ycol(), colormap=input.cmap())
```

We’ll break the chunks of code down...

Create Functions for Each Step

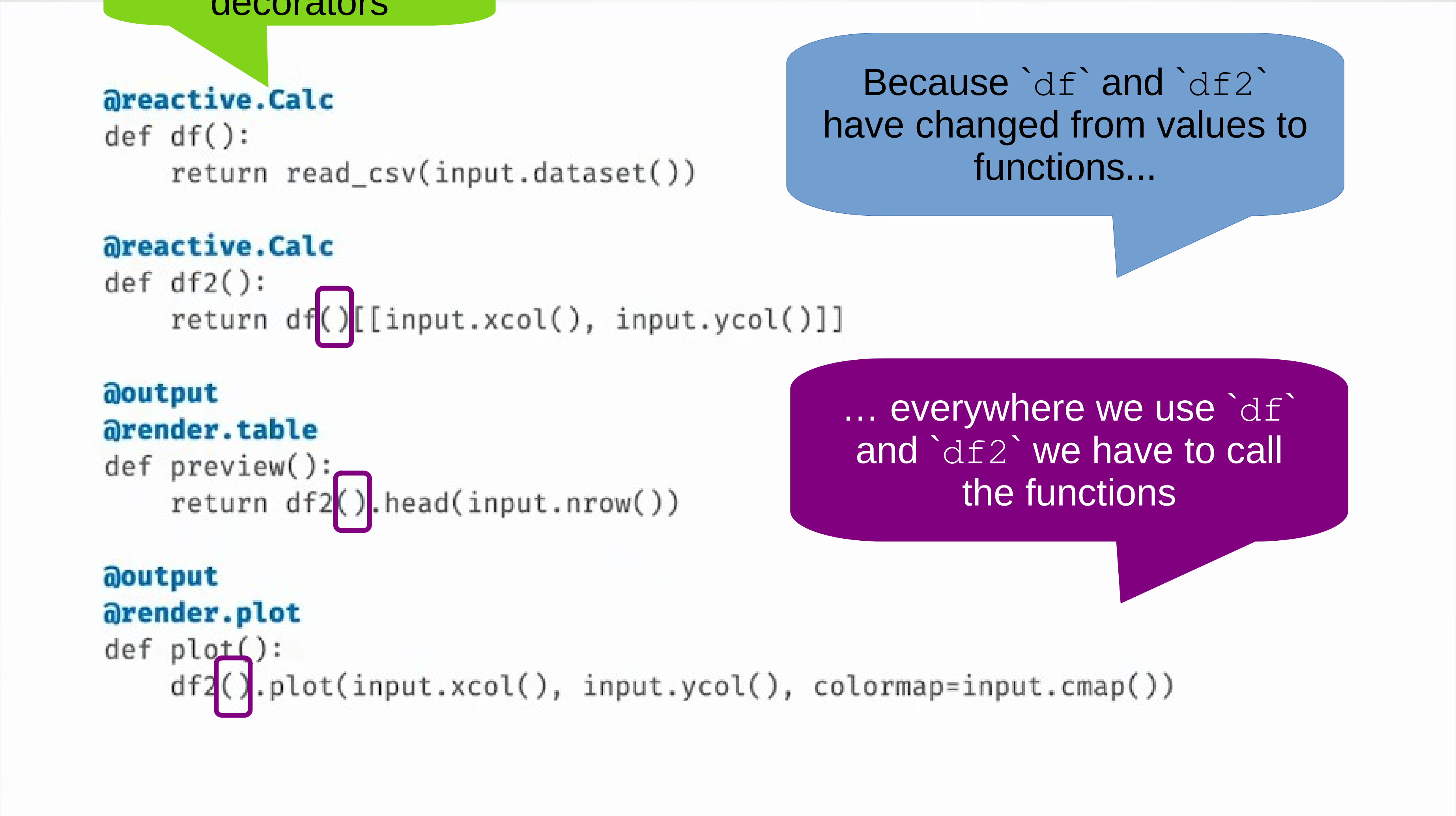


```
def df():
    return read_csv(input.dataset())

def df2():
    return df[[input.xcol(), input.ycol()]]

def preview():
    return df2.head(input.nrow())

def plot():
    df2.plot(input.xcol(), input.ycol(), colormap=input.cmap())
```

… into functions

Add Respective Decorators

… and add decorators



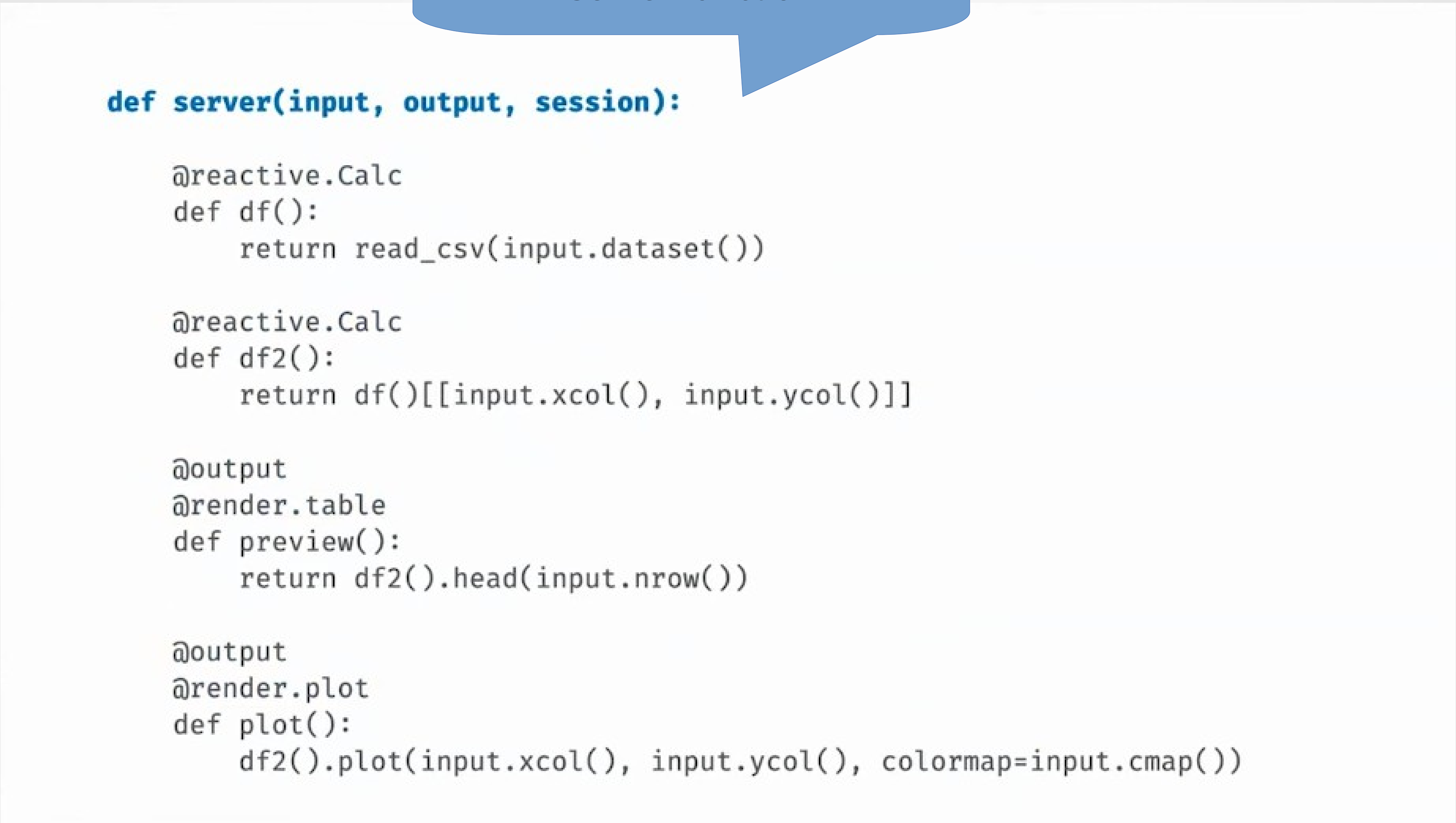
```
@reactive.Calc
def df():
    return read_csv(input.dataset())

@reactive.Calc
def df2():
    return df()[[input.xcol(), input.ycol()]]

@output
@render.table
def preview():
    return df2().head(input.nrow())

@output
@render.plot
def plot():
    df2().plot(input.xcol(), input.ycol(), colormap=input.cmap())
```

Because `df` and `df2` have changed from values to functions...

… everywhere we use `df` and `df2` we have to call the functions

Pull Together inside Server Function

Finally put these inside a server function



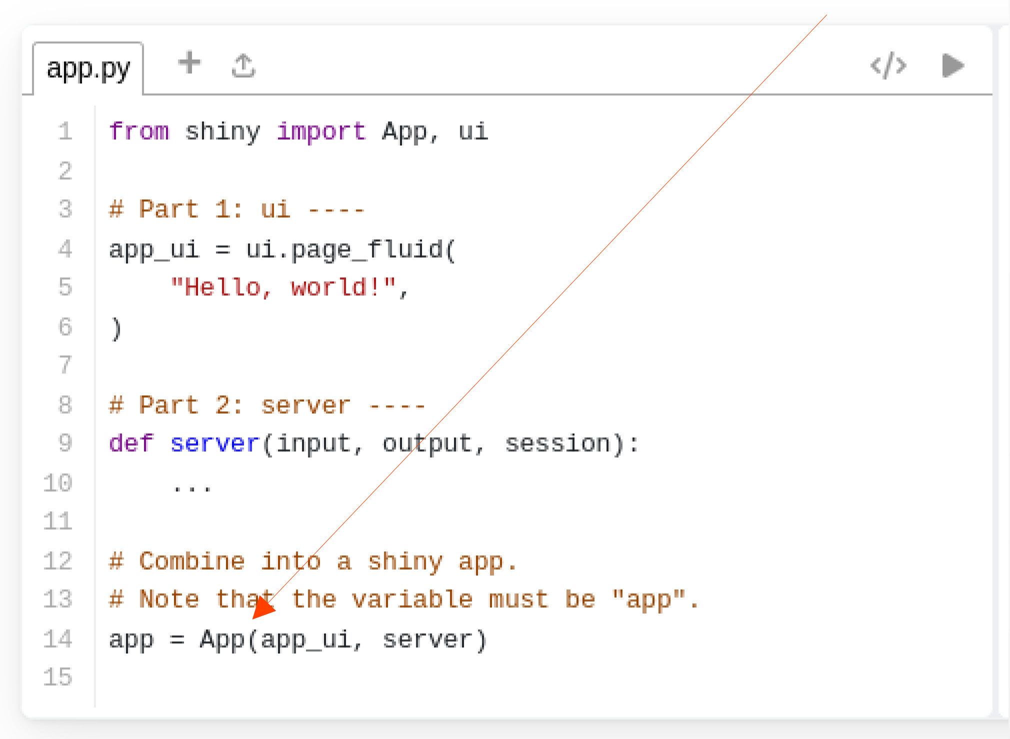
```
def server(input, output, session):

    @reactive.Calc
    def df():
        return read_csv(input.dataset())

    @reactive.Calc
    def df2():
        return df()[[input.xcol(), input.ycol()]]

    @output
    @render.table
    def preview():
        return df2().head(input.nrow())

    @output
    @render.plot
    def plot():
        df2().plot(input.xcol(), input.ycol(), colormap=input.cmap())
```

Decorators

- After a break, we’ll work through a Jupyter Notebook to help you get to grips with decorators
- See `` `01_decorators.ipynb` `` in code along folder

Your First Shiny App

- Shiny applications consist of two parts:
  - 1) The User Interface and
  - 2) The server function
- The are combined using a `` `shiny.App` `` object

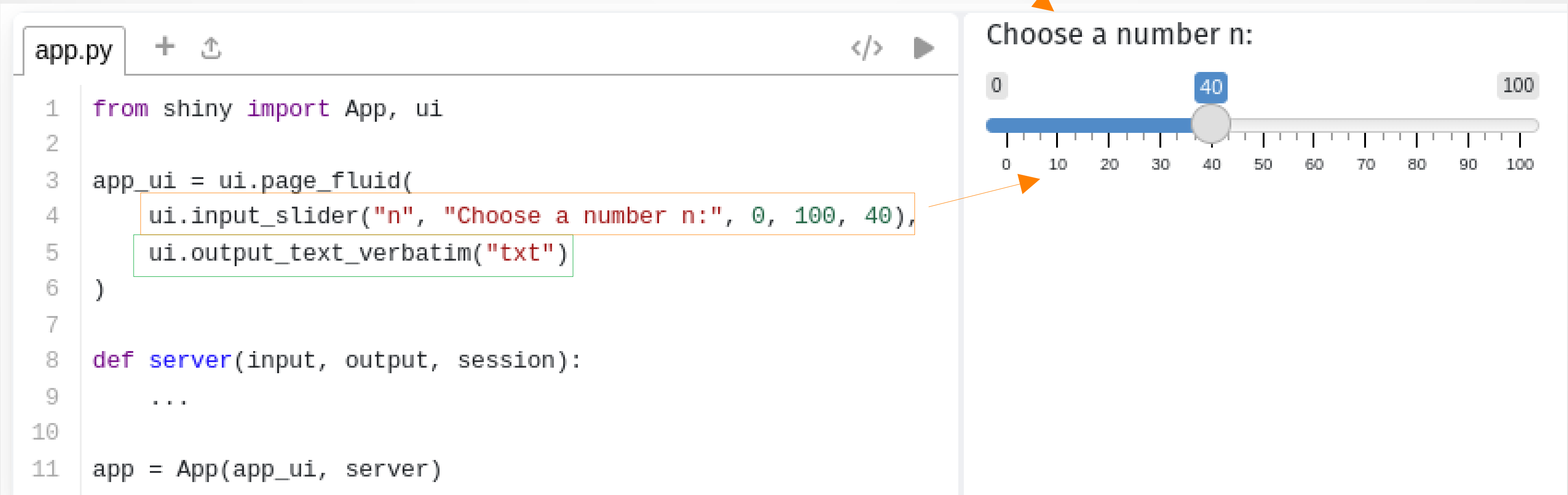


```
from shiny import App, ui

# Part 1: ui ----
app_ui = ui.page_fluid(
    "Hello, world!",
)

# Part 2: server ----
def server(input, output, session):
    ...

# Combine into a shiny app.
# Note that the variable must be "app".
app = App(app_ui, server)
```

- Dynamic parts of the app happen within the ‘`server`’ function
- Try for yourself – code in `` `02_my_first_shiny_app/app.py` `` within code along folder
- To use remotely, copy and paste the code from the above file and paste into HERE

Adding UI In/Outputs

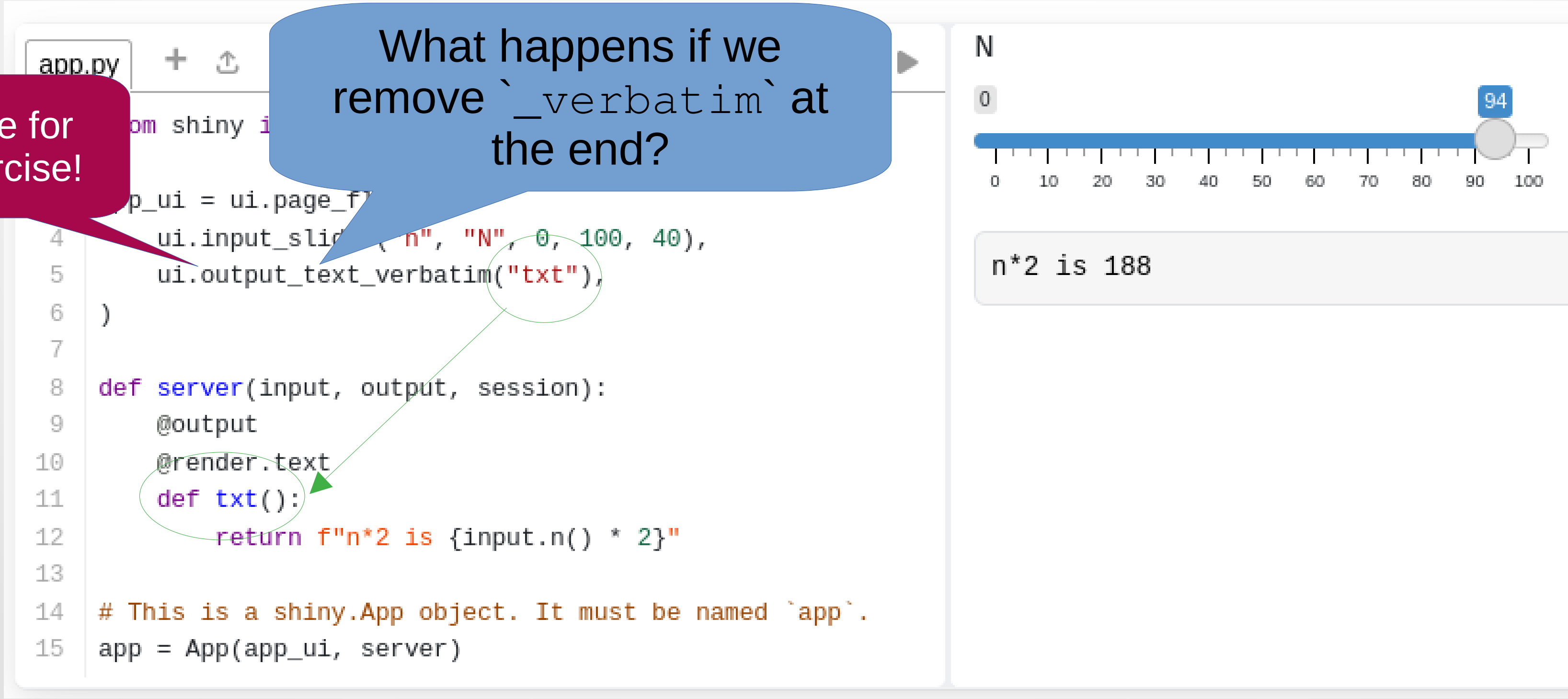
- Two new piece added in UI section:
- `` `input_slider()` `` - slider
- `` `output_text_verbatim()` `` - creates a field to display dynamically generated text (nothing there yet!)
- See `` `03_adding_ui_in_outputs/app.py` ``



```python
from shiny import App, ui

app_ui = ui.page_fluid(
    ui.input_slider("n", "Choose a number n:", 0, 100, 40),
    ui.output_text_verbatim("txt")
)

def server(input, output, session):
    ...

app = App(app_ui, server)
```

Getting Started – Adding Server Logic

- Now we can add to the server function.
- Inside of the server function, we'll define an output function named `txt`.
- This output function provides the content for the `output_text_verbatim("txt")` in the UI.
- Try for yourself with `` `04_server_logic/app.ay` ``



```
om shiny i

p_ui = ui.page_f
    ui.input_sli  "n", "N", 0, 100, 40),
    ui.output_text_verbatim("txt"),
)

def server(input, output, session):
    @output
    @render.text
    def txt():
        return f"n*2 is {input.n() * 2}"

# This is a shiny.App object. It must be named `app`.
app = App(app_ui, server)
```

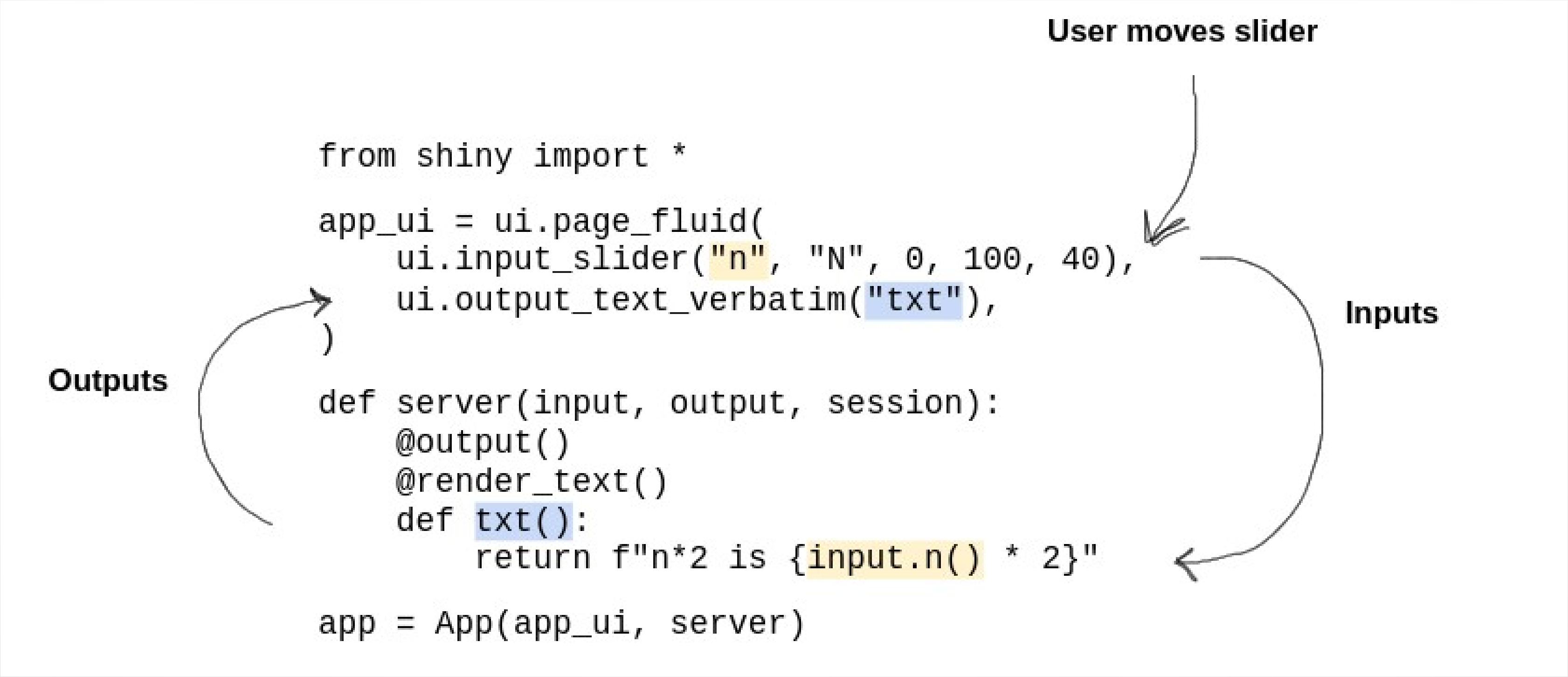
```
n*2 is 188
```

Clue for exercise!

What happens if we remove `` `_verbatim` `` at the end?

How do the different parts of the code interact?

- From the previous example, what’s going on when we move the slider?



```
from shiny import *

app_ui = ui.page_fluid(
    ui.input_slider("n", "N", 0, 100, 40),
    ui.output_text_verbatim("txt"),
)

def server(input, output, session):
    @output()
    @render_text()
    def txt():
        return f"n*2 is {input.n() * 2}"

app = App(app_ui, server)
```

Input Controls

Each input control is created by calling a Python function. They take the same first two string arguments:

- `id`: an identifier used to refer to input's value in the server code. For example, `id="x1"` corresponds with `input.x1()` in the server function.

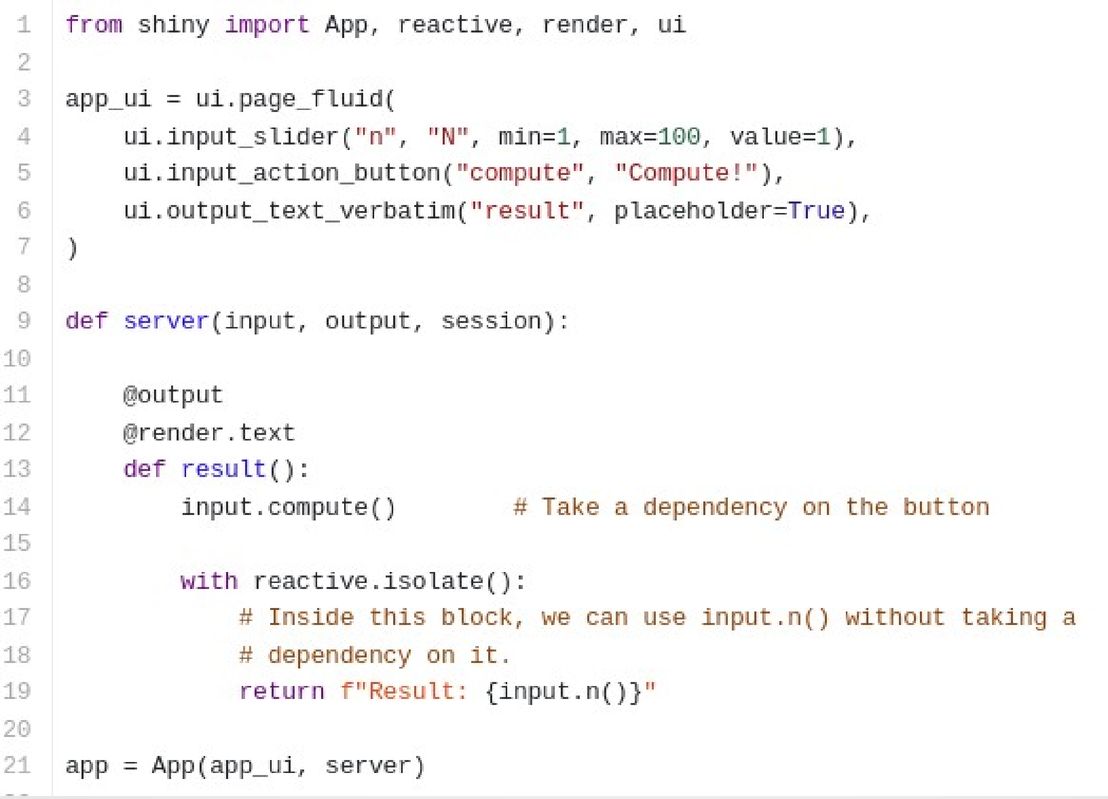
  `id` values must be unique across all input and output objects on a page, and should follow Python variable/function naming rules (lowercase with underscores, alphanumeric characters allowed, cannot start with a number).
- `label`: a description for the input that will appear next to it. Can usually be None if no label is desired.

Note that many inputs take additional arguments. For example, an `input_checkbox` lets you indicate if it should start checked or not.

See full list of inputs **HERE**.

Handling Events – Reactivity `.isolate`

The slider created with the code below doesn’t trigger an output (when changed by the user) until the ‘Compute!’ button is pressed. **However, WILL execute when the app is first loaded**.
See example `` `05_reactive_slider_button_isolate/app.py` ``

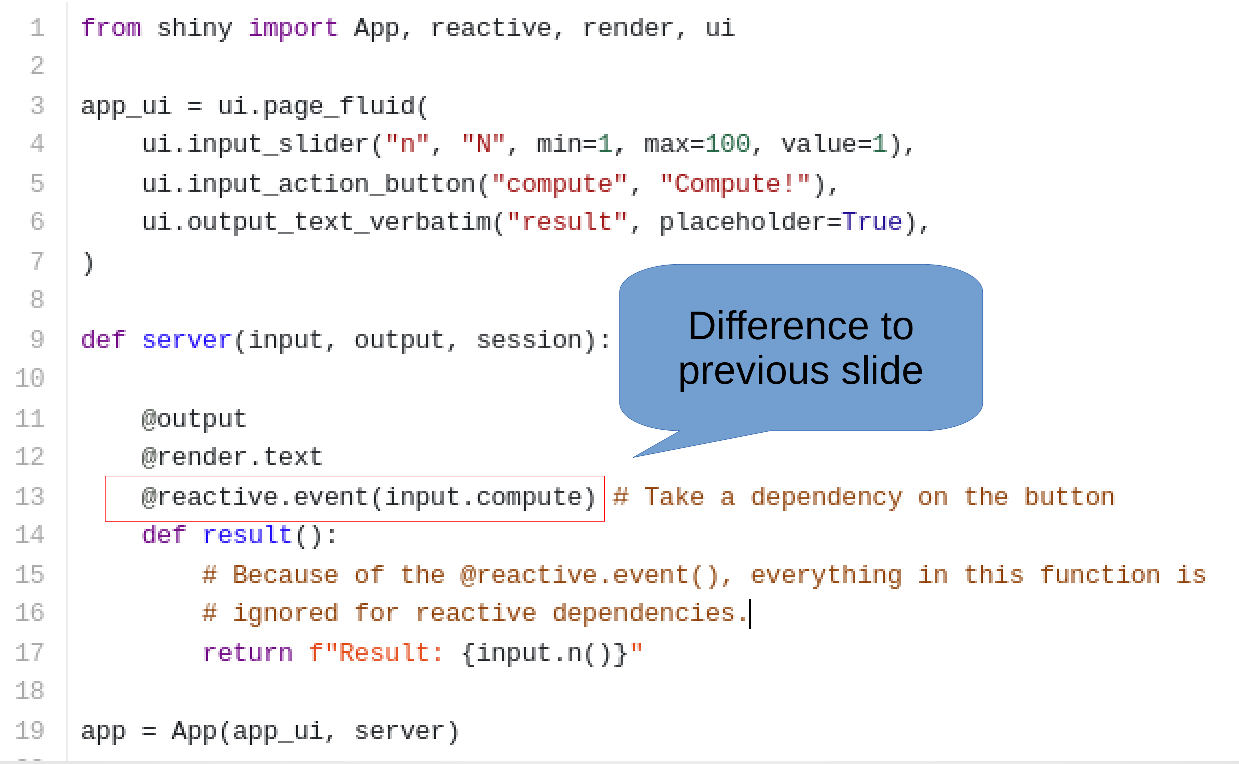


```python
from shiny import App, reactive, render, ui

app_ui = ui.page_fluid(
    ui.input_slider("n", "N", min=1, max=100, value=1),
    ui.input_action_button("compute", "Compute!"),
    ui.output_text_verbatim("result", placeholder=True),
)

def server(input, output, session):

    @output
    @render.text
    def result():
        input.compute()        # Take a dependency on the button

        with reactive.isolate():
            # Inside this block, we can use input.n() without taking a
            # dependency on it.
            return f"Result: {input.n()}"

app = App(app_ui, server)
```

Handling Events – Reactivity `.event`

The slider created with the code below doesn’t trigger an output (when changed by the user) until the ‘Compute!’ button is pressed. **However, will NOT execute when the app is first loaded.**

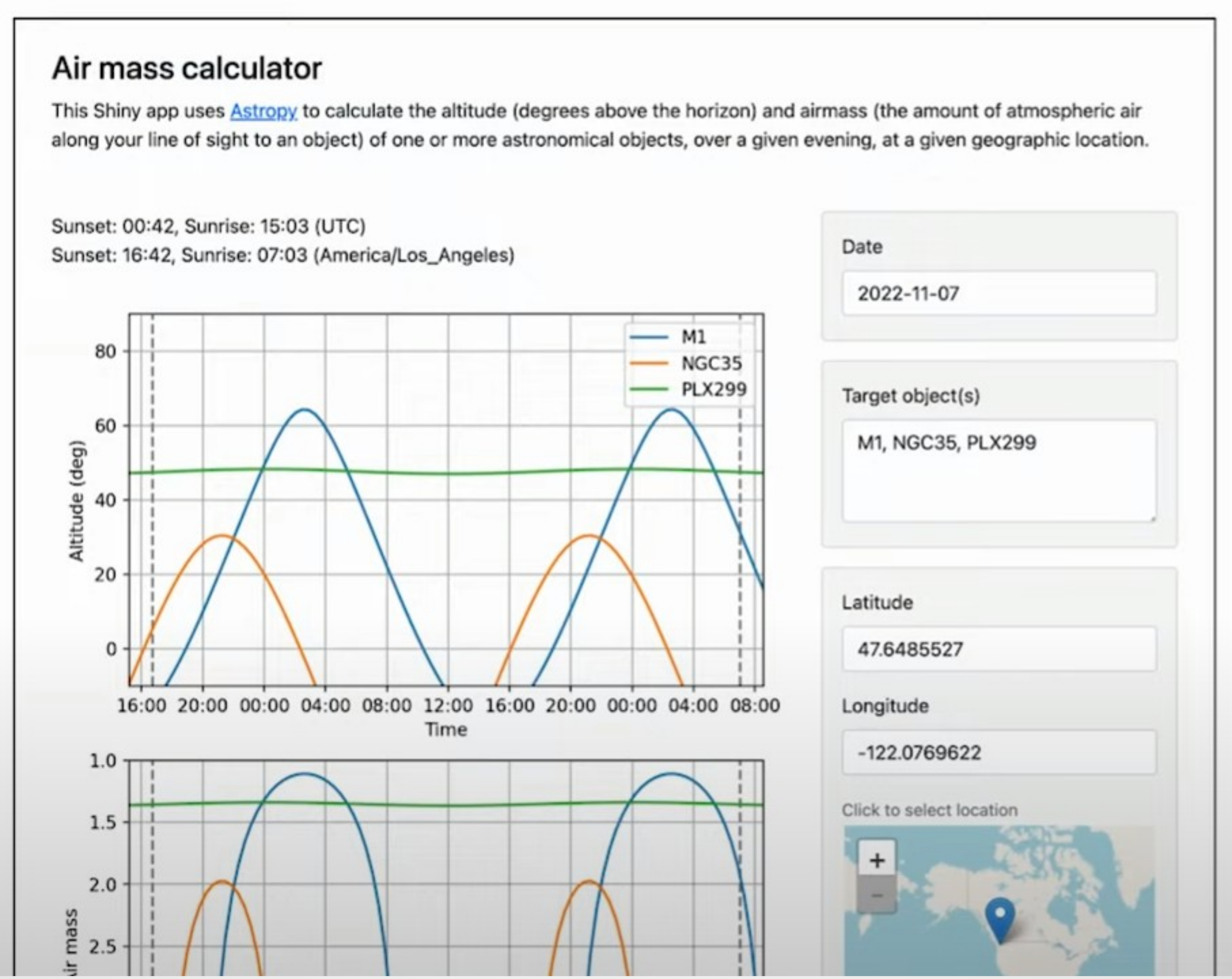
See example `06_reactive_slider_button_event/app.py`



```python
from shiny import App, reactive, render, ui

app_ui = ui.page_fluid(
    ui.input_slider("n", "N", min=1, max=100, value=1),
    ui.input_action_button("compute", "Compute!"),
    ui.output_text_verbatim("result", placeholder=True),
)

def server(input, output, session):

    @output
    @render.text
    @reactive.event(input.compute) # Take a dependency on the button
    def result():
        # Because of the @reactive.event(), everything in this function is
        # ignored for reactive dependencies.
        return f"Result: {input.n()}"

app = App(app_ui, server)
```

Difference to previous slide

Something a little more complex...



Air mass calculator

This Shiny app uses Astropy to calculate the altitude (degrees above the horizon) and airmass (the amount of atmospheric air along your line of sight to an object) of one or more astronomical objects, over a given evening, at a given geographic location.

Sunset: 00:42, Sunrise: 15:03 (UTC)
Sunset: 16:42, Sunrise: 07:03 (America/Los_Angeles)

Date: 2022-11-07

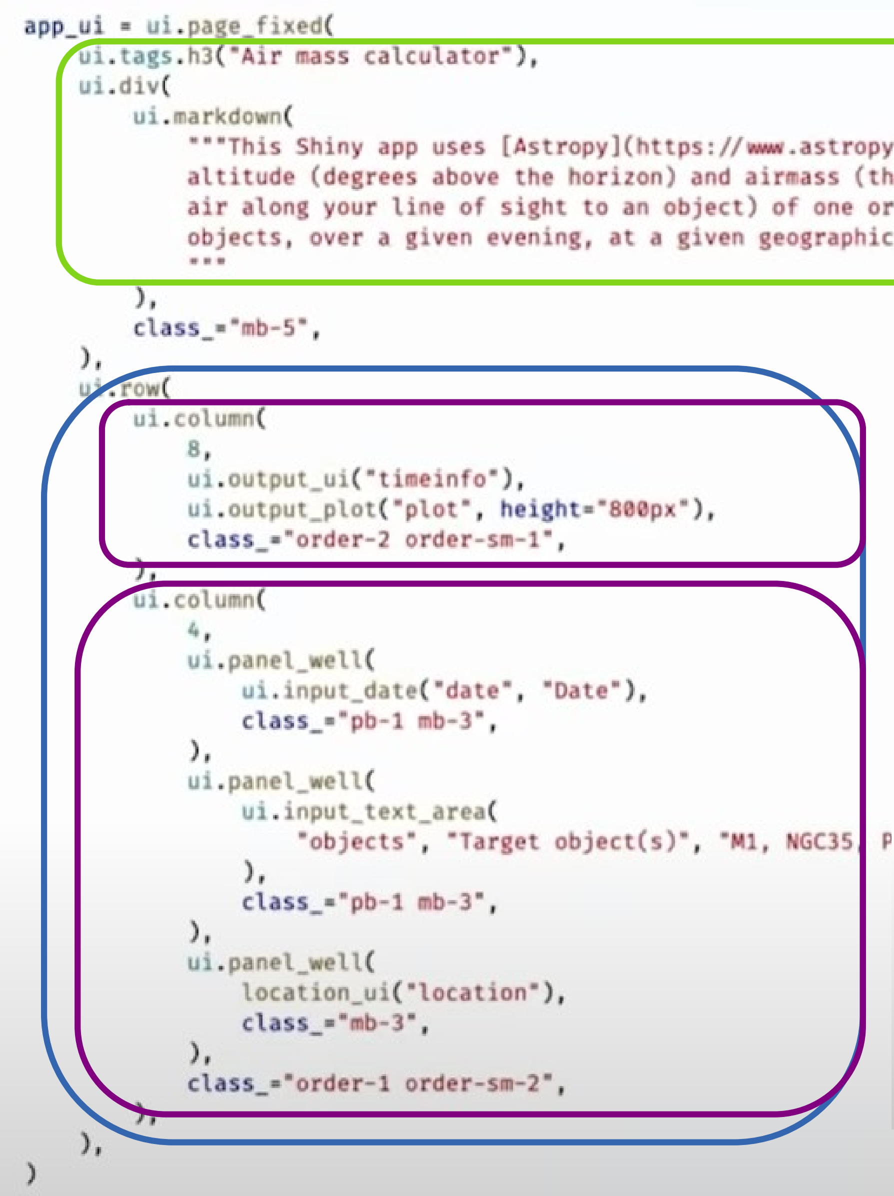
Target object(s): M1, NGC35, PLX299

Latitude: 47.6485527

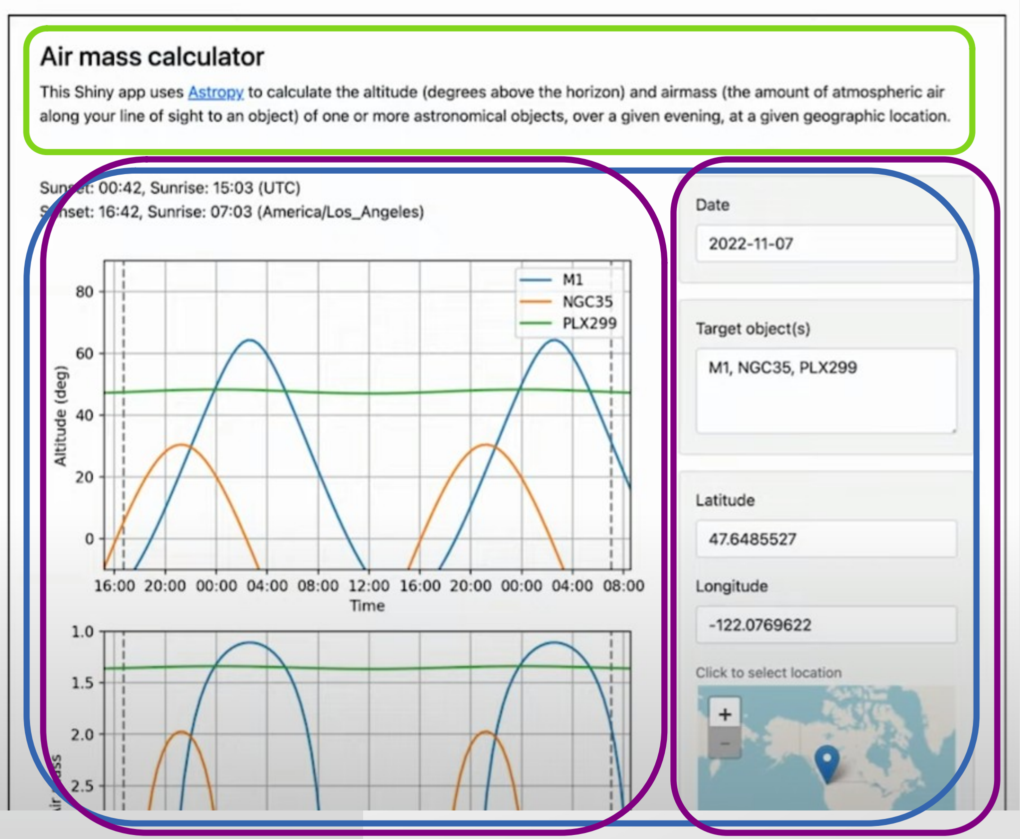
Longitude: -122.0769622

Click to select location

Header, Body, Columns



```
app_ui = ui.page_fixed(
    ui.tags.h3("Air mass calculator"),
    ui.div(
        ui.markdown(
            """This Shiny app uses [Astropy](https://www.astropy.org/) to calculate the
            altitude (degrees above the horizon) and airmass (the amount of atmospheric
            air along your line of sight to an object) of one or more astronomical
            objects, over a given evening, at a given geographic location.
            """
        ),
        class_="mb-5",
    ),
    ui.row(
        ui.column(
            8,
            ui.output_ui("timeinfo"),
            ui.output_plot("plot", height="800px"),
            class_="order-2 order-sm-1",
        ),
        ui.column(
            4,
            ui.panel_well(
                ui.input_date("date", "Date"),
                class_="pb-1 mb-3",
            ),
            ui.panel_well(
                ui.input_text_area(
                    "objects", "Target object(s)", "M1, NGC35, P
                ),
                class_="pb-1 mb-3",
            ),
            ui.panel_well(
                location_ui("location"),
                class_="mb-3",
            ),
            class_="order-1 order-sm-2",
        ),
    ),
)
```



Air mass calculator

This Shiny app uses Astropy to calculate the altitude (degrees above the horizon) and airmass (the amount of atmospheric air along your line of sight to an object) of one or more astronomical objects, over a given evening, at a given geographic location.

Sunset: 00:42, Sunrise: 15:03 (UTC)

Sunset: 16:42, Sunrise: 07:03 (America/Los_Angeles)

Date: 2022-11-07

Target object(s): M1, NGC35, PLX299

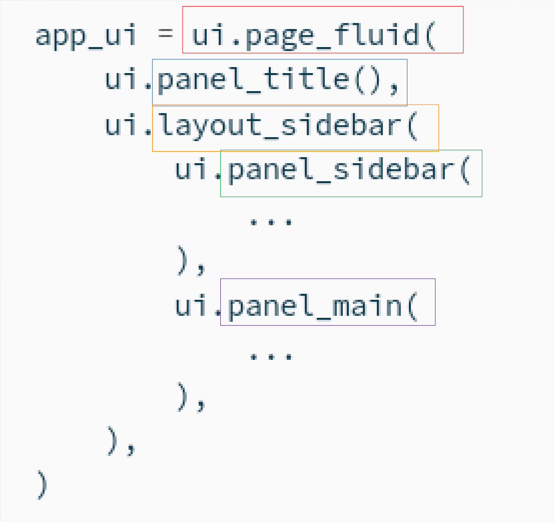
Latitude: 47.6485527

Longitude: -122.0769622

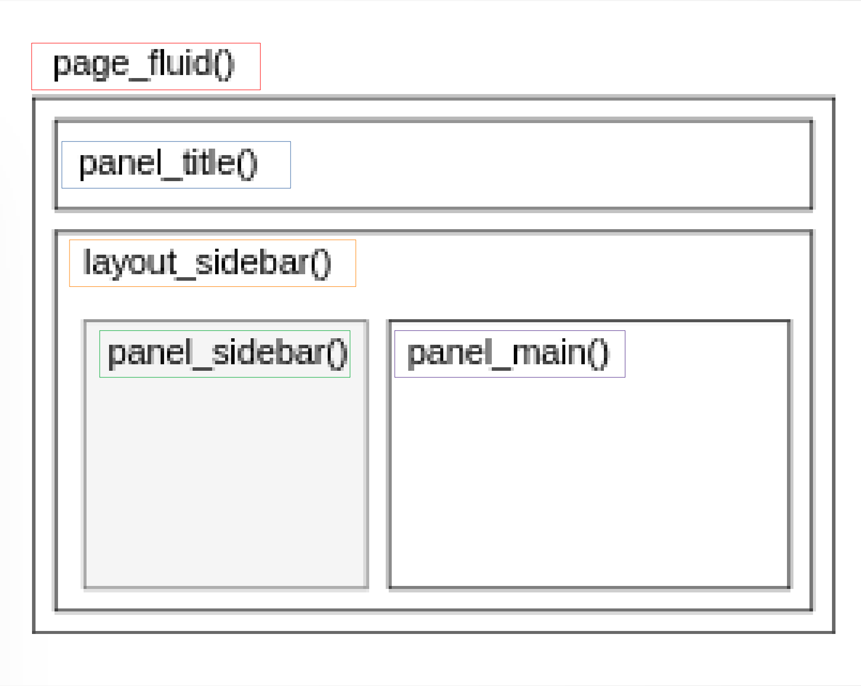
Click to select location

HTML Page Layouts

See am example of an HTML page layout at `07_html_page_sidebar/app.py`



```
app_ui = ui.page_fluid(
    ui.panel_title(),
    ui.layout_sidebar(
        ui.panel_sidebar(
            ...
        ),
        ui.panel_main(
            ...
        ),
    ),
)
```

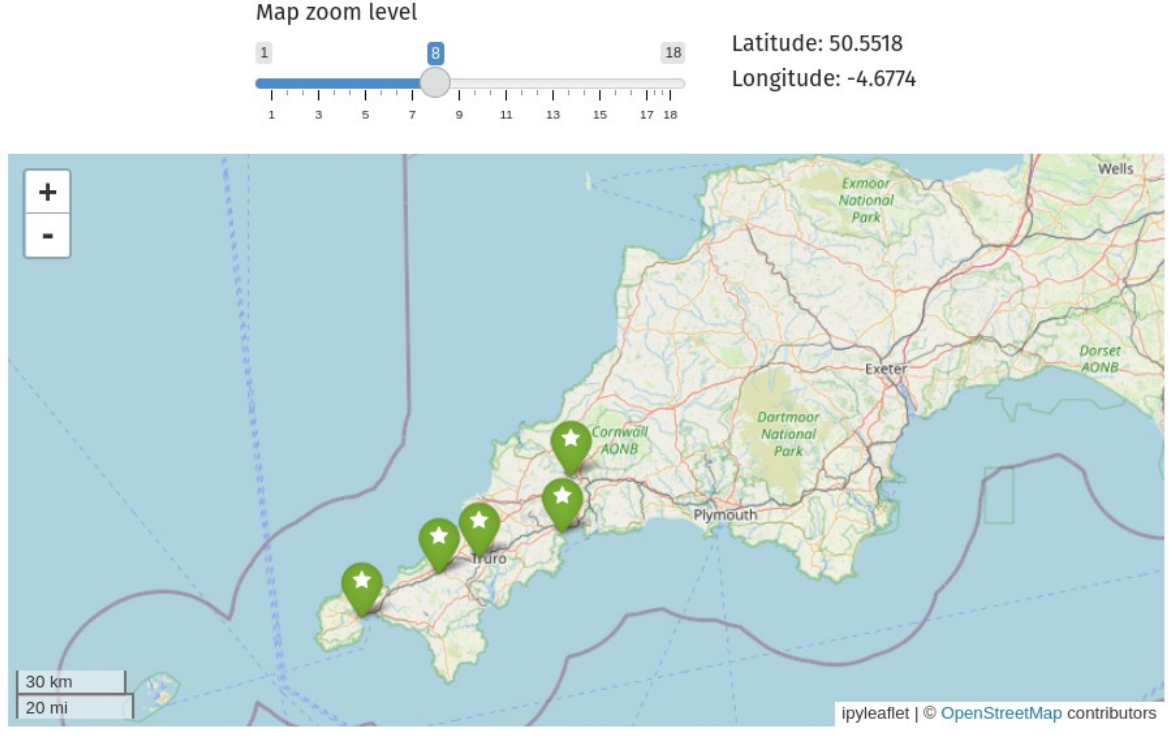


Using Imported Custom Functions

- Sometimes, to improve the readability of your code you can put functions into a separate file and then import them for use within you Streamlit app.
- This is standard Python functionality – not specific to Shiny
- **This will only work with local (i.e., VS Code) – not online...**
- See example `08_pyshiny_app_import_custom_functions/app.py`

Interactive Maps with Shiny for Python

- Possible to render maps – very similar to Folium
- See example `09_map_example/app.py`



Map zoom level

8

Latitude: 50.5518

Longitude: -4.6774

Exercise

- See `exercise` folder
- There are 5 incidences of missing values within the code (replaced with **XXXX**)
- Works best with web environment (only last step that won’t work on VSCode)
- Solutions to be provided on Wednesday

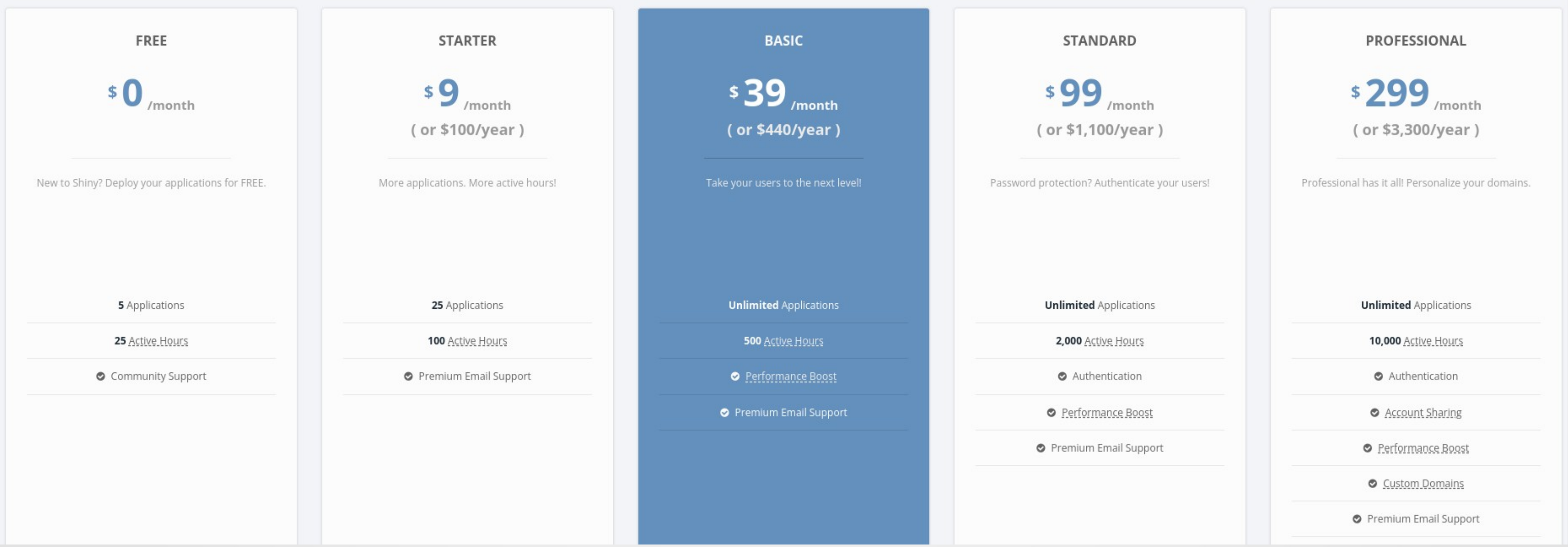
Differences with Streamlit?

- Streamlit does not require you to organise your code as per previous slide (i.e., follow the patter of `app_ui`, `server`, `app = App()`
- Streamlit will re-run the entire app every time there is a user interaction (i.e., click, change of parameter, move a slider, etc)

The above make it very easy to create an app – however with increasing complexity comes increasing problems.

Deploying Shiny Apps

- Deploy to **shinyapp.io** (cloud hosting)
- Deploy to Shiny Server (open source)
- Deploy to Posit Connect (commercial)
- Read more about these options **HERE**



FREE	STARTER	BASIC	STANDARD	PROFESSIONAL
$ 0 /month	$ 9 /month	$ 39 /month	$ 99 /month	$ 299 /month
	(or $100/year)	(or $440/year)	(or $1,100/year)	(or $3,300/year)
New to Shiny? Deploy your applications for FREE.	More applications. More active hours!	Take your users to the next level!	Password protection? Authenticate your users!	Professional has it all! Personalize your domains.
5 Applications	**25** Applications	**Unlimited** Applications	**Unlimited** Applications	**Unlimited** Applications
25 Active Hours	**100** Active Hours	**500** Active Hours	**2,000** Active Hours	**10,000** Active Hours
✓ Community Support	✓ Premium Email Support	✓ Performance Boost ✓ Premium Email Support	✓ Authentication ✓ Performance Boost ✓ Premium Email Support	✓ Authentication ✓ Account Sharing ✓ Performance Boost ✓ Custom Domains ✓ Premium Email Support