Shiny : : CHEAT SHEET

Build an App

Shiny makes it easy to create truly reactive data & AI apps in pure Python

Shiny apps easily scale in complexity and sophistication thanks to its reactivity model and other opinionated design choices.

Build with AI assistance:
gallery.shinyapps.io/assistant

Get inspiration & templates:

- Run `shiny create` in terminal
- shiny.posit.co/py/templates
- shiny.posit.co/py/gallery

Supports output from many popular packages

Collect user input with `input_*()`

Reactively render a plot output (re-execute when relevant `input` changes)

```python
# app.py
import matplotlib.pyplot as plt
import numpy as np
from shiny.express import (
  input, render, ui
)

ui.input_slider(
  "n", "Sample Size", 0, 1000, 50
)

@render.plot
def dist():
  x = np.random.randn(input.n())
  plt.hist(x, range=[-3, 3])
```

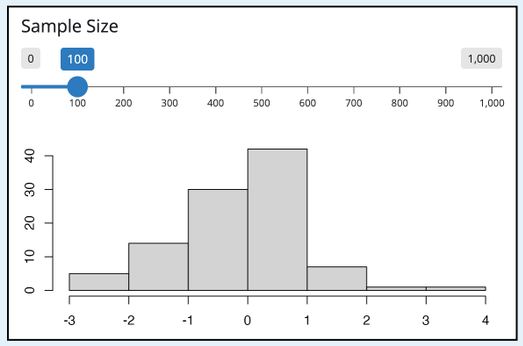
Reactively reads the slider's value

Save as **app.py**

Optionally include supporting code, images, etc. in same directory.

app-name
- app.py
- www/
- <other>/

Launch apps via the VSCode extension or with the `shiny run` CLI



Share

Share your app in three ways:

1. **Host it on shinyapps.io**, a cloud based service from Posit. To deploy Shiny apps:

   Create a free or professional account at **shinyapps.io**

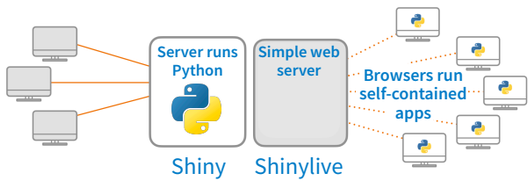
   Use the reconnect-python package to publish with `rsconnect deploy shiny <path to directory>`

2. **Purchase Posit Connect**, a publishing platform for R and Python. **posit.co/connect**

3. **Use open source deployment options** **shiny.posit.co/py/docs/deploy.html**

Shinylive

Shinylive apps use WebAssembly to run entirely in a browser–no need for a special server to run Python.



- Edit and/or host Shinylive apps at **shinylive.io**
- Create a Shinylive version of an app to deploy with
  `shinylive export myapp site`
  Then deploy to a hosting site like Github or Netlify
- Embed Shinylive apps in Quarto sites, blogs, etc.

```
---
filters:
 - shinylive
---
An embedded Shinylive app:

```{shinylive-python}
#| standalone: true

# [App.py code here…]
```
```

To embed a Shinylive app in a Quarto doc, include the bold syntax.

Outputs

Decorate a function with **@render.*** to reactively render Python outputs

@render.data_frame

Species	Island	Bill Length (mm)	Body Mass (g)
Chinstrap	Dream	45.70	3650
Chinstrap	Dream	55.80	4000
Chinstrap	Dream	43.50	3400
Chinstrap	Dream	49.60	3775

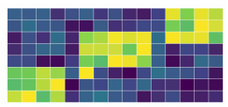
@render.code

```
   area    peri     shape perm
1 4990 2791.90 0.0903296  6.3
2 7002 3892.60 0.1486220  6.3
3 7558 3930.66 0.1833120  6.3
4 7352 3869.32 0.1170630  6.3
```

@render.download

Download

@render.plot



@render.text

Current value: 30

@render.ui

Current *value*: **30**

@render.image

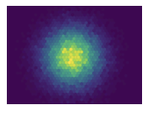
posit

And many more via the **shinywidgets** project

@render_altair

@render_plotly

@render_bokeh



@render_widget

Inputs

Collect values from the user.

Reactively read input values with **`input.<id>()`**

Action — **`ui.input_action_button(`** `id, label, …)`

Action — **`ui.input_action_link(`** `id, label, …)`

Processing... — **`ui.input_task_button(`** `id, label, …)`

Check me — **`ui.input_checkbox(`** `id, label, value, …)`

Choice 1, Choice 2, Choice 3 — **`ui.input_checkbox_group(`** `id, label, choices, selected, …)`

`ui.input_dark_mode(``id, mode)`

2025-07-28 — **`ui.input_date(`** `id, label, value, …)`

`ui.input_date_range(` `id, label, start, end, …)`

Choose File — **`ui.input_file(`**`id, label, …)`

1 — **`ui.input_numeric(`** `id, label, value, …)`

Option 1, Option 2, Option 3 — **`ui.input_radio_buttons(`** `id, label, choices, selected, …)`

Choice 1A / Choice 1B — **`ui.input_select(`** `id, label, choices, selected, …)`
Also **`ui.input_selectize()`**

0 50 100 — **`ui.input_slider(`** `id, label, min, max, value, …)`

`ui.input_switch(` `id, label, value, …)`

Enter text... — **`ui.input_text(`** `id, label, value, …)`
Also **`ui.input_text_area()`**

Reactivity

Reactive values work together with reactive functions. A reactive value must be read from within a reactive functions to avoid the error `No current reactive context`

Module located at `shiny.reactive`

Create a reactive value from other (reactive) values. Helps avoid redundant logic and computation.

Create a reactive UI output.

Perform side effects like logging, updating a database, etc.

```
from shiny import reactive
from shiny.express import (
  input, render, ui
)

ui.input_text("text", "Enter text")

@reactive.calc
def length():
    return len(input.text())

@render.text
def length_output():
    return f"{length()} characters"

@reactive.effect
def length_log():
    print(f"{length()} characters")
```

Reactive functions re-execute when any of their reactive dependencies (i.e., values) change. However, sometimes you want to ignore all but one (i.e., event):

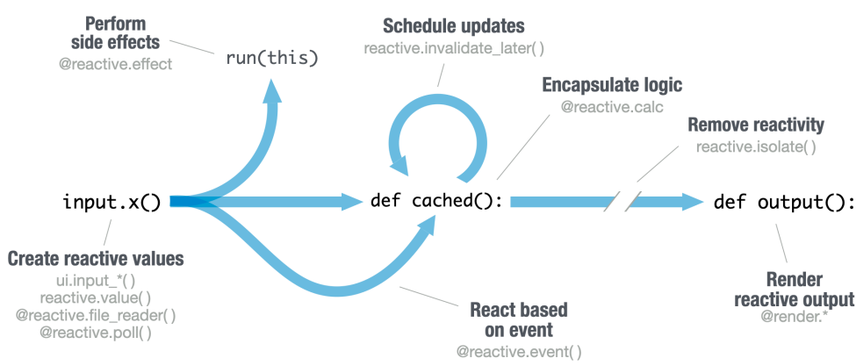
Don't execute until `input.submit` is truthy (i.e. button is clicked)

```
ui.input_text("name", "Enter name")
ui.input_action_button("submit", "Submit")

@render.text
@reactive.event(input.submit)
def greeting():
  return f"Hello {input.name()}!"

@reactive.effect
@reactive.event(input.submit)
def log_name():
  print(f"Name submitted {input.name()}")
```

A `reactive.value()` can be useful for programmatically setting/reading a reactive value. This is often useful when the value can't be derived from `input` values alone.



User Interfaces (UI)

Design delightful UI with a collection of layouts, components, themes, & more.

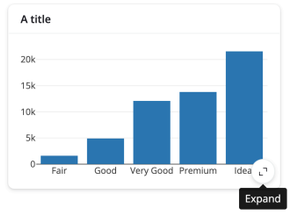
PAGE LAYOUTS

`page_sidebar()`	Screen-filling sidebar layout
`page_fillable()`	Screen-filling page layout
`page_fixed()`	Constrained width page
`page_fluid()`	Basic full-width page
`page_navbar()`	Multi-page app with a top nav bar

CARDS

Visually group UI elements together with the **`card()`** component.

```
card(full_screen = T,
  card_header("A title"),
  plotOutput("my_output"),
  card_footer("A footer")
)
```



UI LAYOUTS

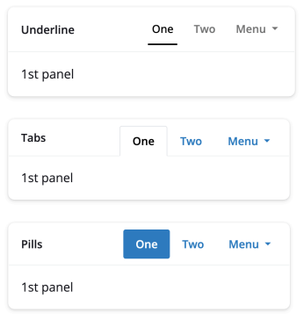
Multiple columns

`layout_columns()`	Bootstrap's 12-column grid
`layout_column_wrap()`	Equal-width columns
`layout_sidebar()`	Resizable 2-column layout

Multiple panels

Navigate a set of **`nav_panel()`**s in various ways with **`navset_card_[underline/tab/pill]()`**

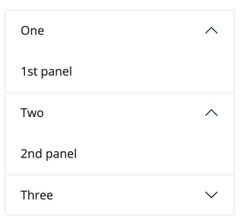
```
navset_card_underline(
  title = "Underline",
  nav_panel(
    "One", "1st panel"
  ),
  nav_panel(
    "Two", "2nd panel"
  ),
  nav_menu(
    "Menu",
    nav_panel(
      "Three",
      "3rd panel"
    )
  )
)
```



ACCORDIONS

Combine with **sidebar()** to group similar inputs

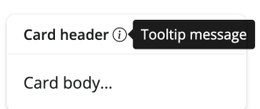
```
accordion(
 open = c("One", "Two"),
 accordion_panel(
   "One", "1st panel"
 ),
 accordion_panel(
   "Two", "2nd panel"
 ),
 accordion_panel(
   "Three", "3rd panel"
 ),
)
```



TOOLTIPS

Provide UX hints and additional context on demand

```
tooltip(
  icon("info-circle"),
  "Tooltip message"
)
```



Custom UI

Make the app behave and look exactly how you want it with web tooling and theming

UI as HTML

Shiny UI is powered by HTML, CSS, and JS:

```
ui.page_fluid(class = "pt-3")
#> <div class="container-fluid pt-3"></div>
```

- Create bespoke experiences with custom HTML (`ui.tags`) and CSS/JS snippets (`ui.include_css()` / `ui.include_js()`).
- Can also interface with popular frameworks (React, Vue, Svelte, etc).

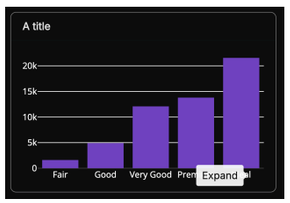
LOCAL FILES

Statically serve any file (image, CSS, JS, etc) by placing them in a `www/` next to `app.py`

THEMES

Choose from set of pre-packaged themes via **shinyswatch** or change main colors / fonts via **brand-yml**

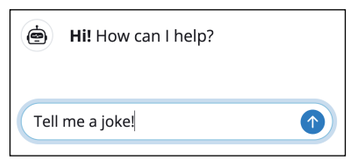
```
color:
  foreground: '#222'
  background: white
  primary: purple
typography:
  fonts:
    - family: Inter
      source: google
```



Generative AI

Build streaming Gen AI interfaces like chatbots and more with the `Chat` and `MarkdownStream` components.

- Use `Chat` to implement a streaming chat interface. Just provide a suitable callback to handle `user_input`.
- Use AI framework of choice (e.g., chatlas, LangChain, etc) for response generation.




```
from chatlas import ChatOpenAI
from shiny.express import ui

chat_client = ChatOpenAI()
chat = ui.Chat("chat")
chat.ui(
  messages=["**Hi!** How can I help?"]
)

@chat.on_user_submit
def _(user_input: str):
  stream = await chat.stream_async(
    user_input
  )
  chat.append_message_stream(stream)
```

Express / Core

- An **app.py** that imports from `shiny.express` uses 'Express mode' to make development faster.
- Express extends "Core" Shiny to make UI and server logic one in the same.
- Core may be more suitable for sophisticated apps where a decoupling of UI and server is beneficial.

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

app_ui = ui.page_fixed(
  ui.input_slider(
    "n", "Sample Size", 0, 100, 50
  )
)

def server(input):
  @render.plot
  def dist():
    x = np.random.randn(input.n())
    plt.hist(x, range=[-3, 3])

app = App(app_ui, server)
```