

Shiny : : CHEATSHEET



Build an app

A **Shiny** app is a web page (**ui**) connected to a computer running a live R session (**server**).



The **UI** is a collection of input, output, and layout elements

The **server** determines how to render outputs given inputs

An **app** is a combination of UI and

Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

☺ Build with AI assistance:

gallery.shinyapps.io/assistant

💡 Get inspiration & examples:

• shiny.posit.co/r/gallery

• shinylive.io/r/examples

• `runExample()` in R console

```
# app.R
library(shiny)

ui <- bslib::page_fluid(
  sliderInput(
    "n", "Sample size", 0, 100, 25
  ),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}

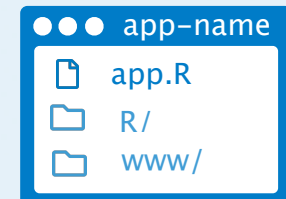
shinyApp(ui, server)
```

Create ***Input()** UI and read with `input$<id>`

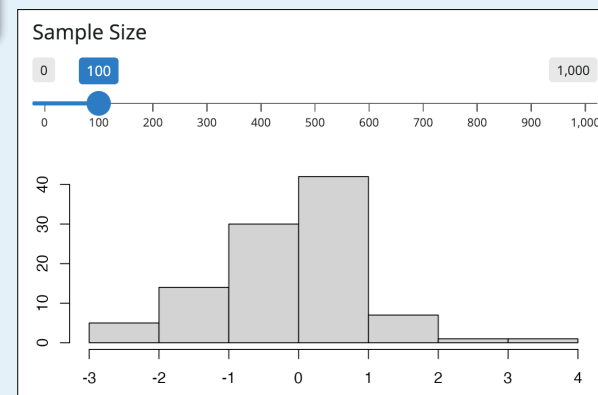
Match ***Output()** with

Save `shinyApp()` to **app.R**

Optionally include supporting code, images, etc. in `R/` and `www/` folders



Launch an **app.R** with `runApp("path/to/app-name")`.



Share

Share your app in four ways:

1. **Host it on Posit Connect Cloud**, a cloud-based service from Posit. To deploy Shiny apps:



Create a free or professional account at connect.posit.cloud



Publish from RStudio, GitHub, or from VS Code or Positron using the Posit Publisher extension

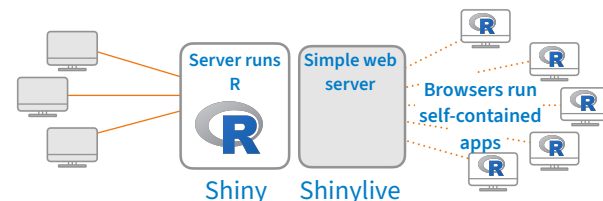
2. **Purchase Posit Connect**, a publishing platform for R and Python. posit.co/connect

3. **Host your own Shiny Server** posit.co/products/open-source/shinyserver

4. **Export to shinylive**, a technology for running apps entirely in the browser. posit-dev.github.io/r-shinylive

Shinylive

Shinylive apps use WebAssembly to run entirely in a browser—no need for a server to run R.



- Edit and/or host apps at shinylive.io/r
- Export an app to Shinylive with `shinylive::export("app-name", "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-r}
#| standalone: true
# [App.py code here...]
```

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

Outputs

Reactively render R outputs



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

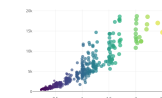
area	peri	shape	perim
1 4990	2791.90	0.0903296	6.3
2 7862	3892.60	0.1486220	6.3
3 7558	3930.66	0.1833120	6.3
4 7352	3869.32	0.1170630	6.3

Current value: 30

Current value: 30



More from htmlwidgets.org ecosystem



`plotOutput(id, width, height,...)`
`renderPlot(expr, ...)`

`tableOutput(id, ...)`
`renderTable(expr, striped, ...)`

`verbatimTextOutput(id, ...)`
`renderPrint(expr, ...)`

`textOutput(id, ...)`
`renderText(expr, ...)`

`uiOutput(id, ...)`
`renderUI(expr, ...)`

`imageOutput(id, ...)`
`renderImage(expr, ...)`

`leafletOutput(id, ...)`
`renderLeaflet(expr, ...)`

`plotlyOutput(id, ...)`
`renderPlotly(expr, ...)`

See output gallery at shiny.posit.co/r/components

Inputs

Collect values from the user.

Access the current value of an input object with `input$<id>`. Input values are **reactive**.

Action `actionButton(id, label, ...)`

Action `actionLink(id, label, ...)`

☒ Choice 1
☒ Choice 2
☐ Choice 3
`checkboxGroupInput(id, label, choices, selected, ...)`

☒ Check me
`checkboxInput(id, label, value, ...)`

`dateInput(id, label, value, ...)`

`dateRangeInput(id, label, start, end, ...)`

Choose File `fileInput(id, label, ...)`

`numericInput(id, label, value, ...)`

☐ Option 1
☒ Option 2
☐ Option 3
`radioButtons(id, label, choices, selected, ...)`

`selectInput(id, label, choices, selected, multiple, ...)`
Also `selectizeInput()`

`sliderInput(id, label, min, max, value, ...)`

`textInput(id, label, value, ...)`
Also `textAreaInput()`

More from the **bslib** package:

☒ `input_dark_mode(id, mode)`

☐ `input_switch(id, label, value, ...)`

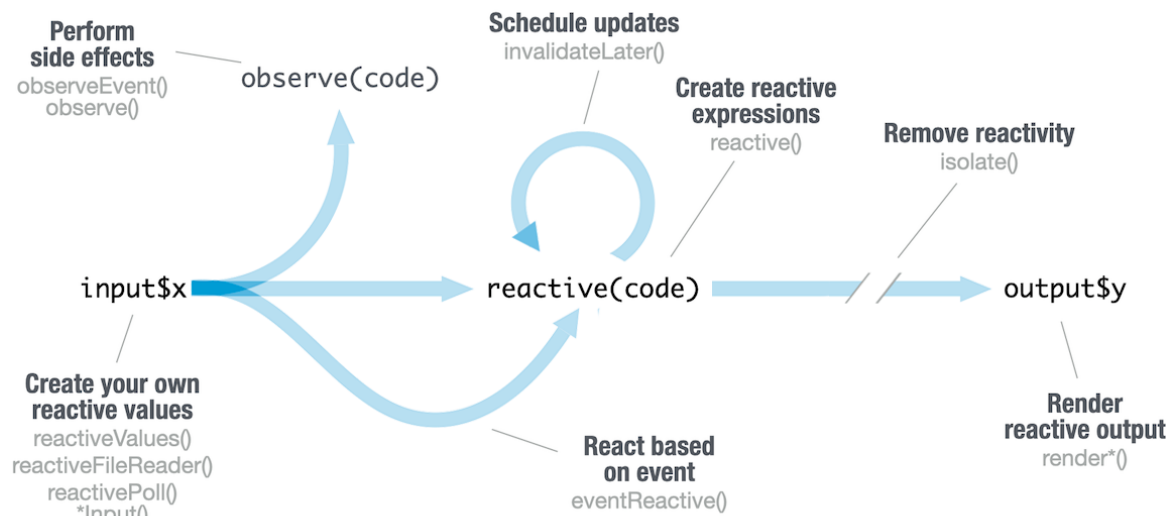
`input_task_button(id, label, value, ...)`

See input gallery at shiny.posit.co/r/components



Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error **Operation not allowed without an active reactive context**.



CREATE REACTIVE VALUES

```
ui <- bslib::page_fluid(
  textInput("a", "", "A")
)

server <- \(input, output){
  print(isolate(input$a))

  rv <- reactiveVal(NULL)
  print(isolate(rv()))
}

shinyApp(ui, server)
```

***Input()** functions
Create a reactive value **input\$<id>** from user input.

reactiveVal(value)
Create a reactive value from a given value. Useful for managing

RENDER REACTIVE OUTPUT

```
ui <- bslib::page_fluid(
  textInput("a", "", "A"),
  textOutput("b")
)

server <- \(input, output){
  output$b <- renderText({
    input$a
  })
}

shinyApp(ui, server)
```

render*() functions
Produces results for a corresponding ***Output()** UI container. Re-render occurs when reactive dependencies change.

Save the results to **output\$<id>**.

CREATE REACTIVE EXPRESSIONS

```
ui <- bslib::page_fluid(
  textInput("a", "", "A"),
  textInput("z", "", "Z"),
  textOutput("b")
)

server <- \(input, output){
  re <- reactive({
    paste(input$a, input$z)
  })
  output$b <- renderText({
    re()
  })
}

shinyApp(ui, server)
```

reactive(x)
Calculate a (reactive) value based on other reactive values.

Useful for encapsulating reactive logic needed across multiple outputs.

PERFORM SIDE EFFECTS

```
ui <- bslib::page_fluid(
  textInput("a", "", "A"),
  actionButton("go", "Go")
)

server <- \(input, output){
  observe(print(input$a))

  observeEvent(input$go, {
    print(input$a)
  })
}

shinyApp(ui, server)
```

observe(x)
Observe changes to reactive values

observeEvent(eventExpr, handlerExpr)
Runs code in 2nd argument when 1st argument changes.

REACT BASED ON EVENT

```
ui <- bslib::page_fluid(
  textInput("a", "", "A"),
  actionButton("go", "Go"),
  textOutput("b")
)

server <- \(input, output){
  re <- eventReactive(
    input$go, {input$a}
  )
  output$b <- renderText({
    re()
  })
}

shinyApp(ui, server)
```

eventReactive(eventExpr, valueExpr)
Creates reactive expression with code in 2nd argument that only invalidates when reactive values in 1st

REMOVE REACTIVE DEPENDENCIES

```
ui <- bslib::page_fluid(
  textInput("a", "", "A"),
  actionButton("go", "Go"),
  textOutput("b")
)

server <- \(input, output){
  output$b <- renderText({
    input$go
    isolate(input$a)
  })
}

shinyApp(ui, server)
```

isolate(expr)
Prevent reactive values from invalidating a reactive expression.

User Interfaces (UI)

Design delightful UI with the **bslib** package. It provides layouts, components, themes, & more.

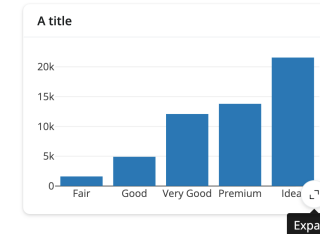
PAGE LAYOUTS

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

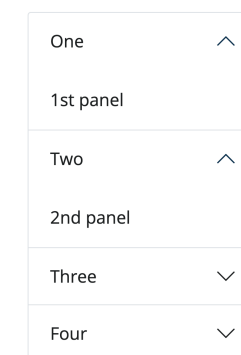
```
navset_card_underline(
  nav_panel("One", "1st panel"),
  nav_panel("Two", "2nd panel"),
  nav_menu("Menu",
    nav_panel("3", "3rd")
  )
)
```

Navigate a set of **nav_panel()**s in various ways with **navset_card_***



ACCORDIONS

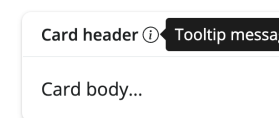
```
accordion(
  open = c("One", "Two"),
  accordion_panel("One", "1st panel"),
  accordion_panel("Two", "2nd panel"),
  accordion_panel("Three", "3rd panel"),
)
```



Tip: place within **sidebar()** to group similar inputs

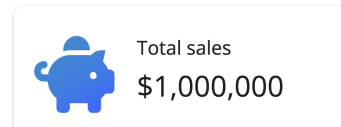
TOOLTIPS

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



VALUE BOXES

```
value_box(
  "Title",
  "Value",
  showcase=icon()
)
```



Custom UI



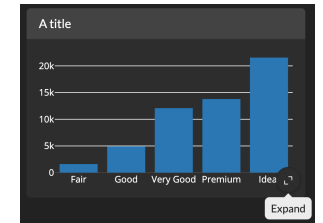
Custom how your app looks and behaves.

THEMES

Bootswatch

Choose from over a dozen pre-packaged themes

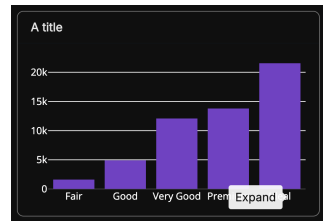
```
library(bslib)
theme <- bs_theme(
  bootswatch="darkly"
)
ui <- page_fluid(
  theme=theme, ...
)
```



Custom themes

Quickly change main colors and fonts. Change in real-time by adding **bs_themer()** to your UI.

```
bs_theme(
  bg = "#222",
  fg = "white",
  primary = "purple",
  base_font =
    font_google("Inter")
)
```



CUSTOM HTML

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

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

If you know these web technologies, you can customize UI to your heart's content. Start small by modifying/authoring HTML and including CSS/JS snippets. Or, go fully custom with **htmlTemplate()**

HTML Add HTML elements with **tags**, a list of functions that parallel common HTML tags, e.g. **tags\$a()**. Unnamed arguments are treated as children and named arguments become HTML attributes.

CSS To include a CSS file, use **includeCSS()**, or
1. Place the file in the **www** subdirectory
2. Link to it with:

```
tags$head(tags$link(href="<file name>",
  rel="stylesheet"))
```

JS To include JS, use **includeScript()** or
1. Place the file in the **www** subdirectory
2. Link to it with:

```
tags$head(tags$script(src="<file name>"))
```

IMAGE To include an image:
1. Place the file in the **www** subdirectory
2. Link to it with **img(src="<file name>")**