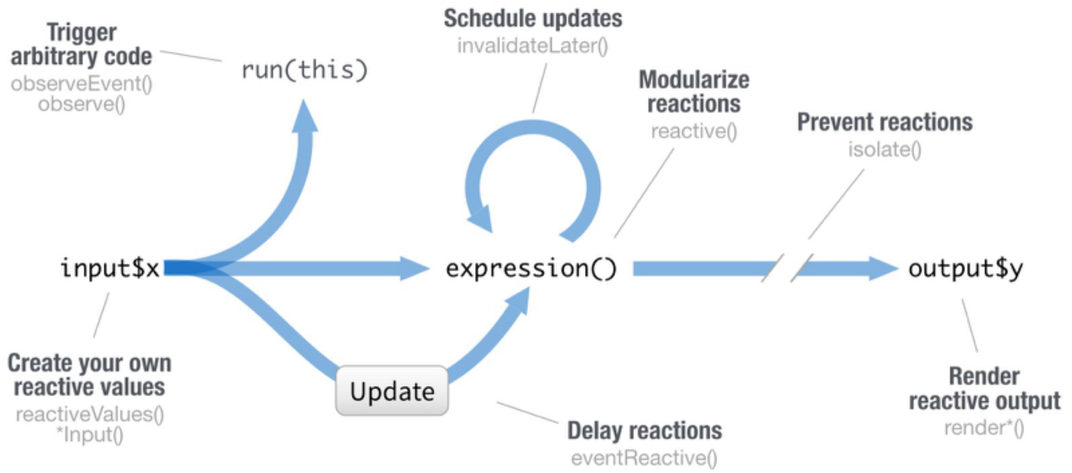
250 Northern Ave, Boston, MA 02210
Phone: 844-448-1212
Email: info@rstudio.com
Web: http://www.rstudio.com

Reactive Programming

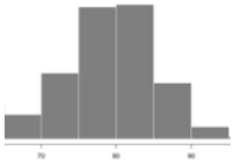
And User Interface

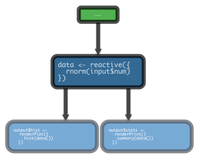


Garrett Grolemund

Data Scientist and Professional Educator
April 2016
Email: garrett@rstudio.com

Use…

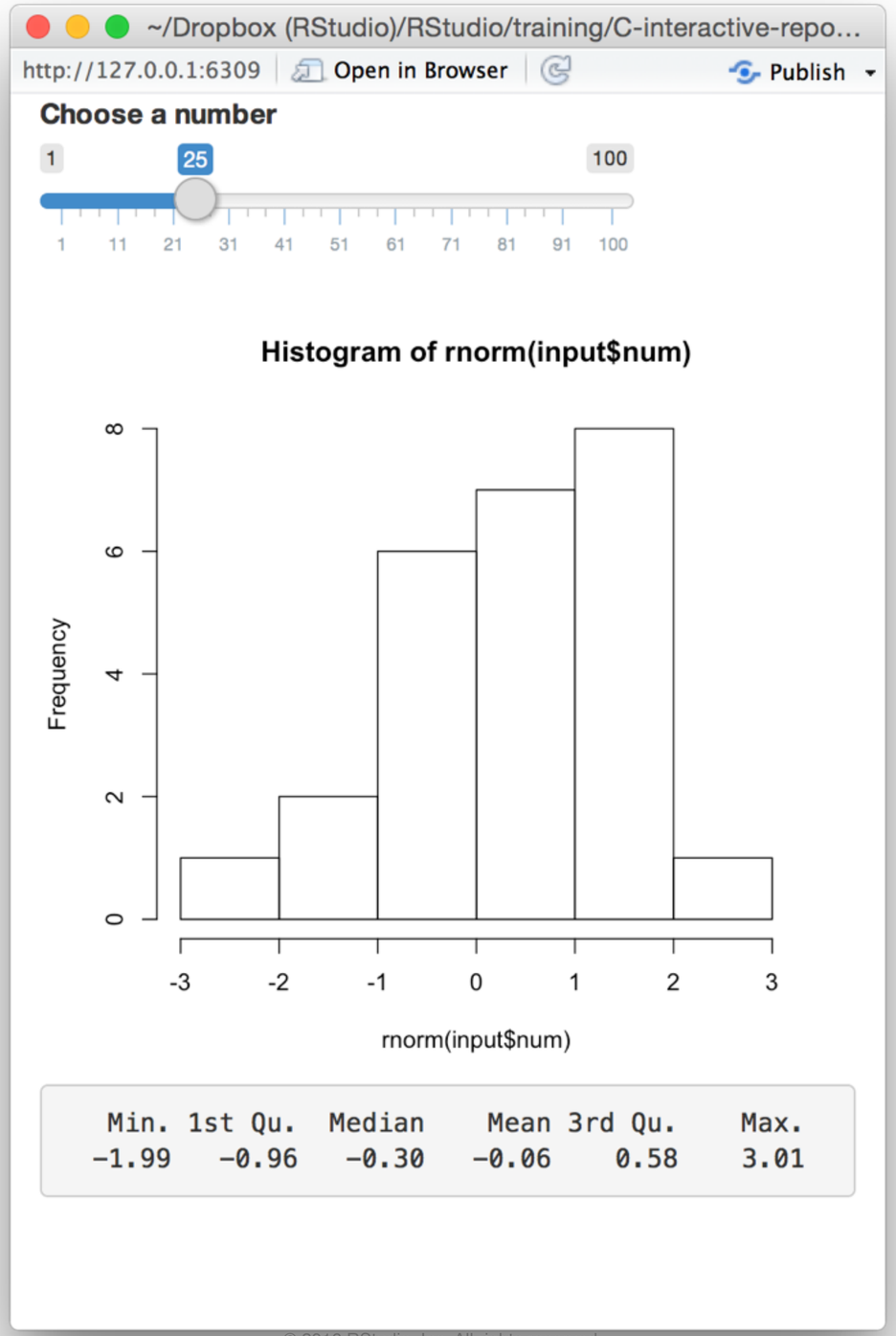




render() to make an **object to display** in the UI.

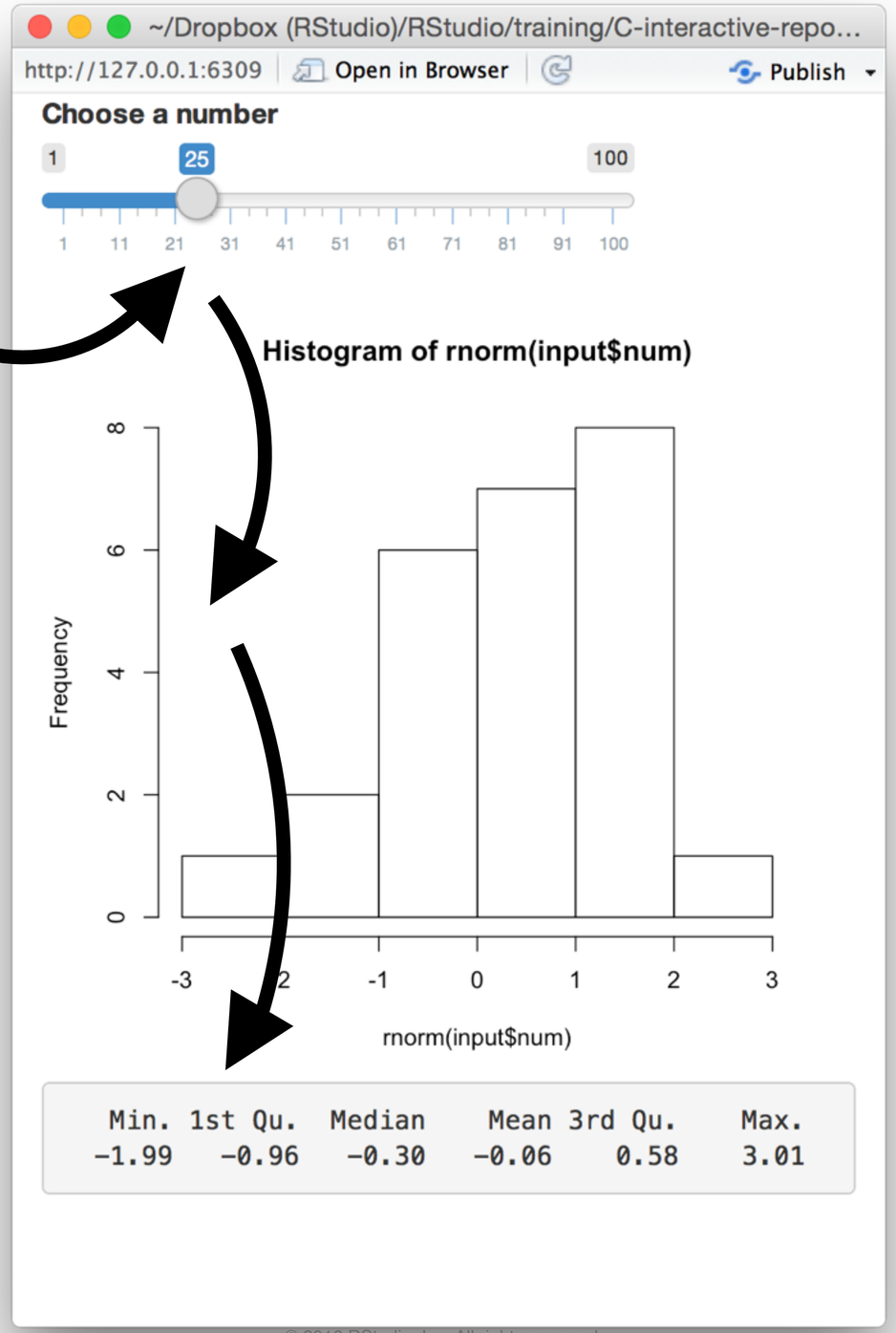
reactive() to make an **object to use** in downstream code.

```
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   data <- reactive({ rnorm(input$num) })
   output$hist <- renderPlot({
     hist(data())
   })
   output$sum <- renderPrint({
     summary(data())
   })
}
shinyApp(ui = ui, server = server)
```



```
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
  data <- reactive({ rnorm(input$num) })
  output$hist <- renderPlot({
    hist(data())
  })
  output$sum <- renderPrint({
    summary(data())
  })
}
shinyApp(ui = ui, server = server)
```

Can we delay the reactions?



Prevent reactions with isolate()

isolate()

Makes a reactive object non-reactive.

```
renderPlot({ hist(rnorm(isolate(input$num))) })
```

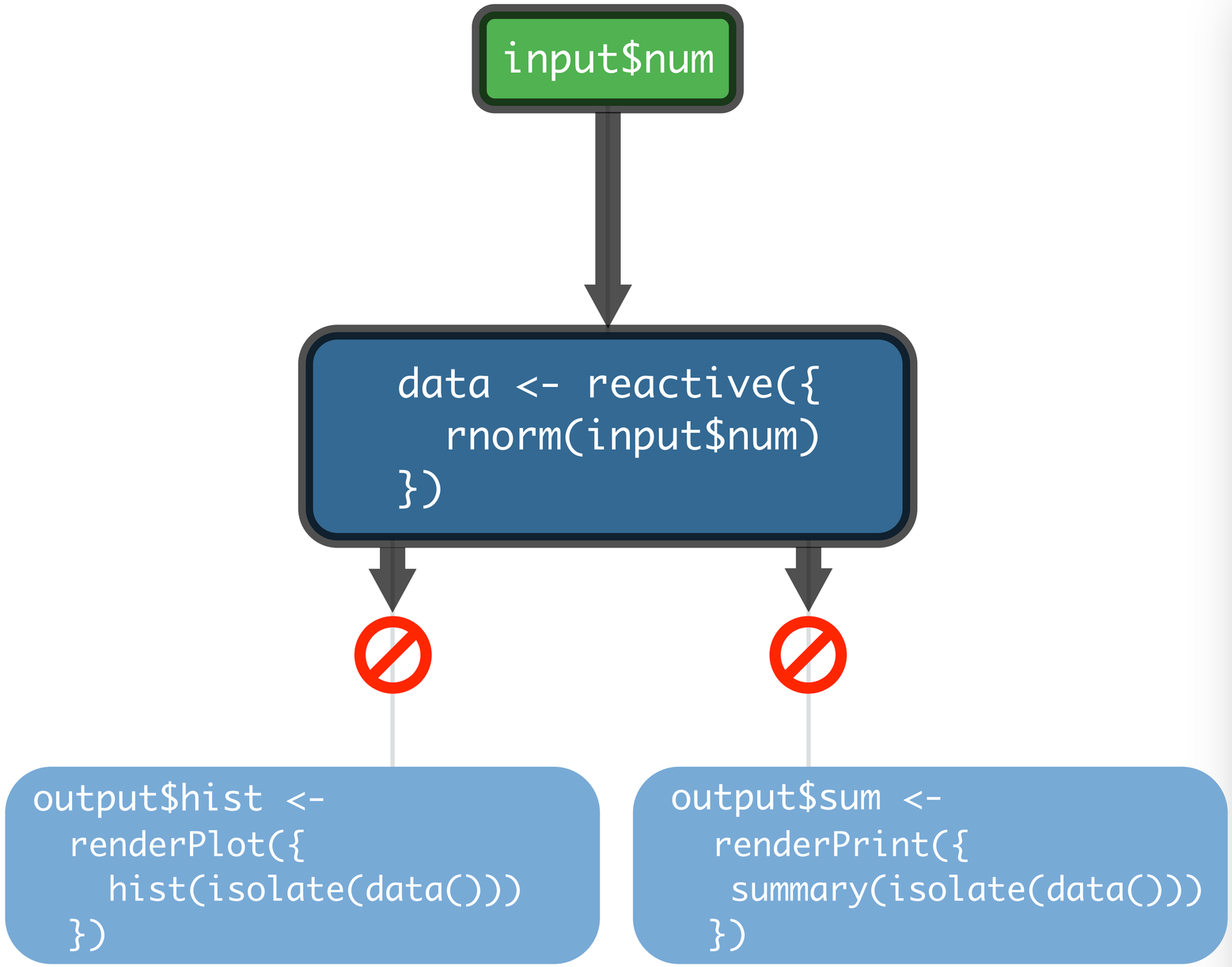
- Calls reactive values *without taking a dependency*
- Never notified, never reacts.

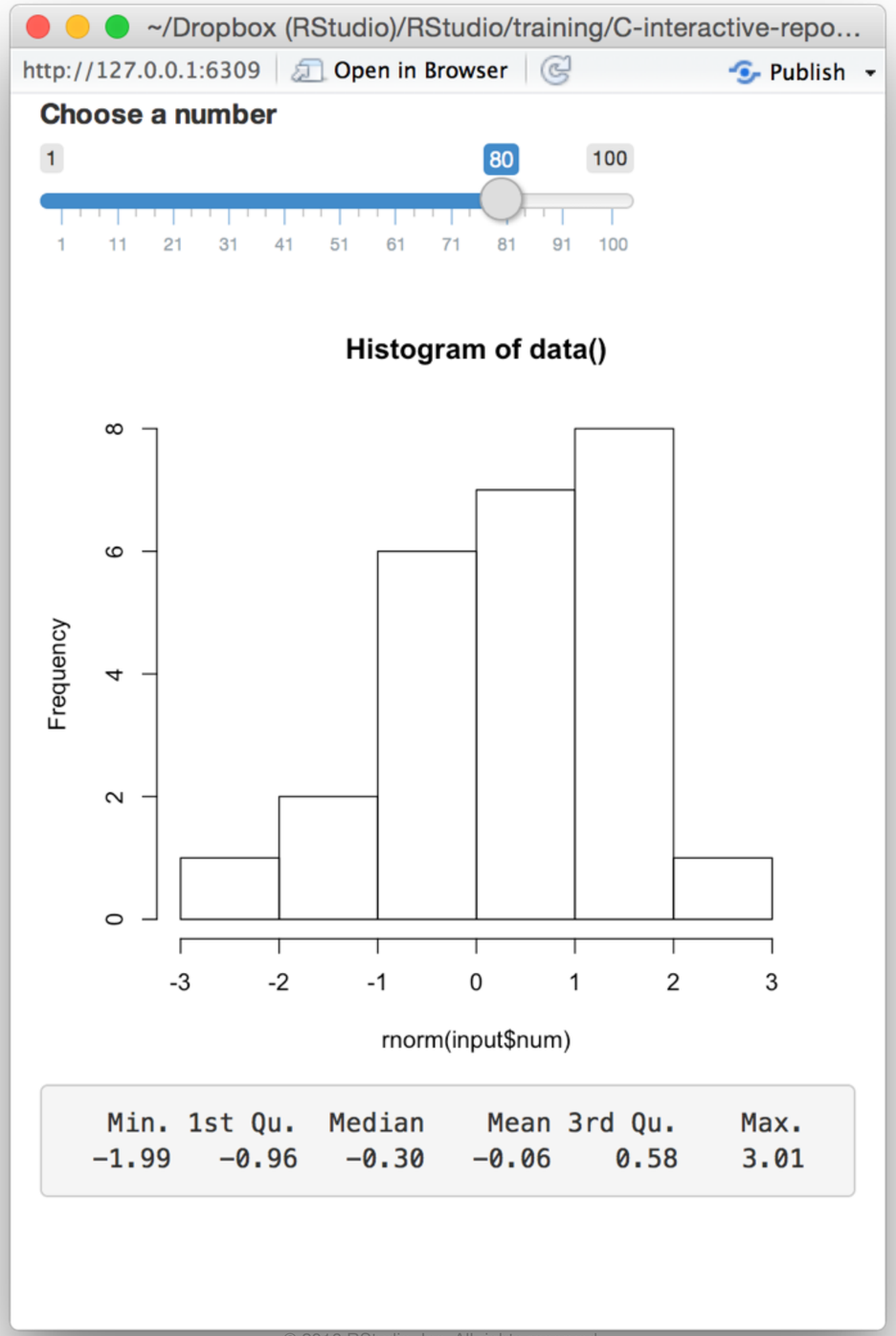
isolate()

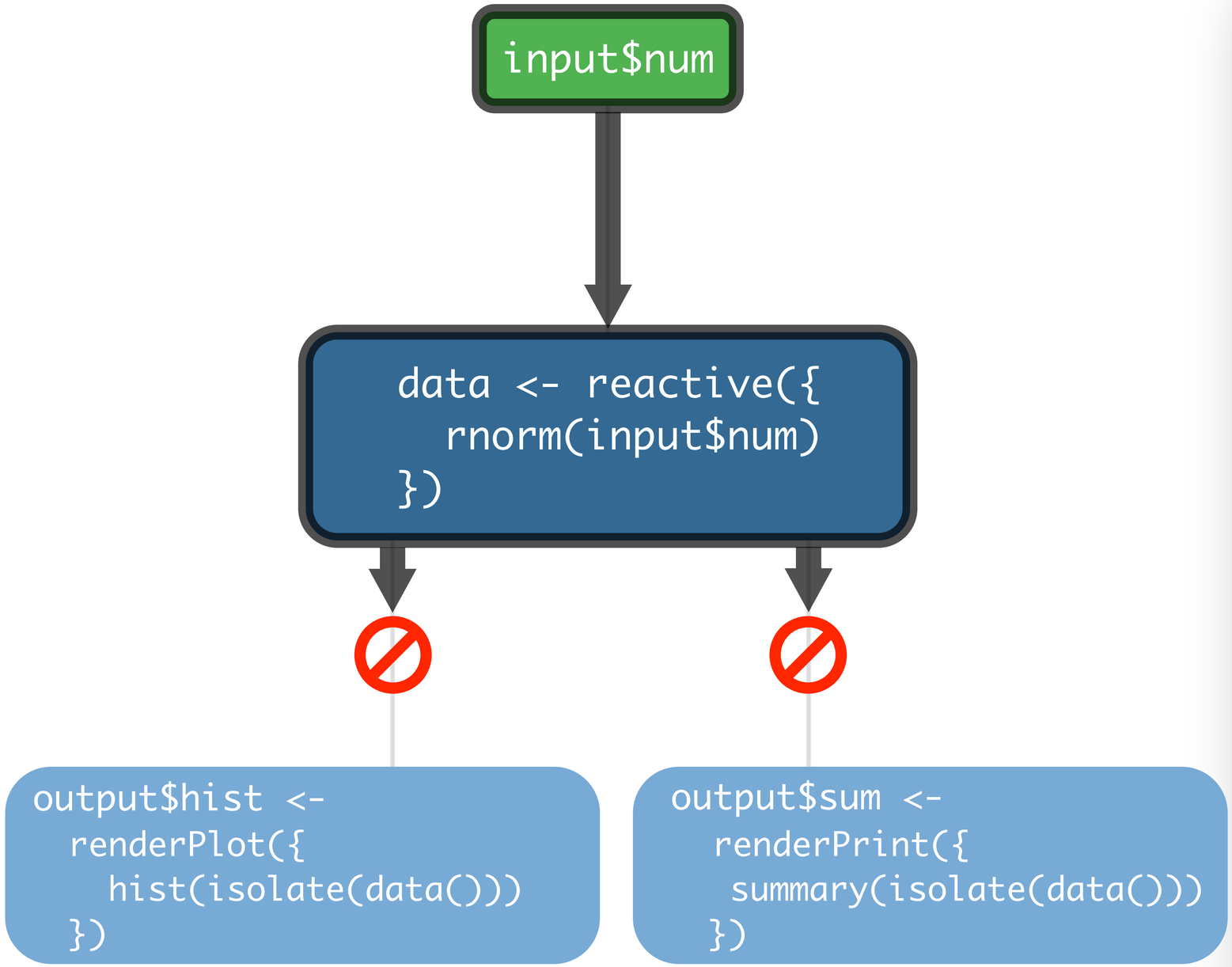
Makes a reactive object non-reactive.

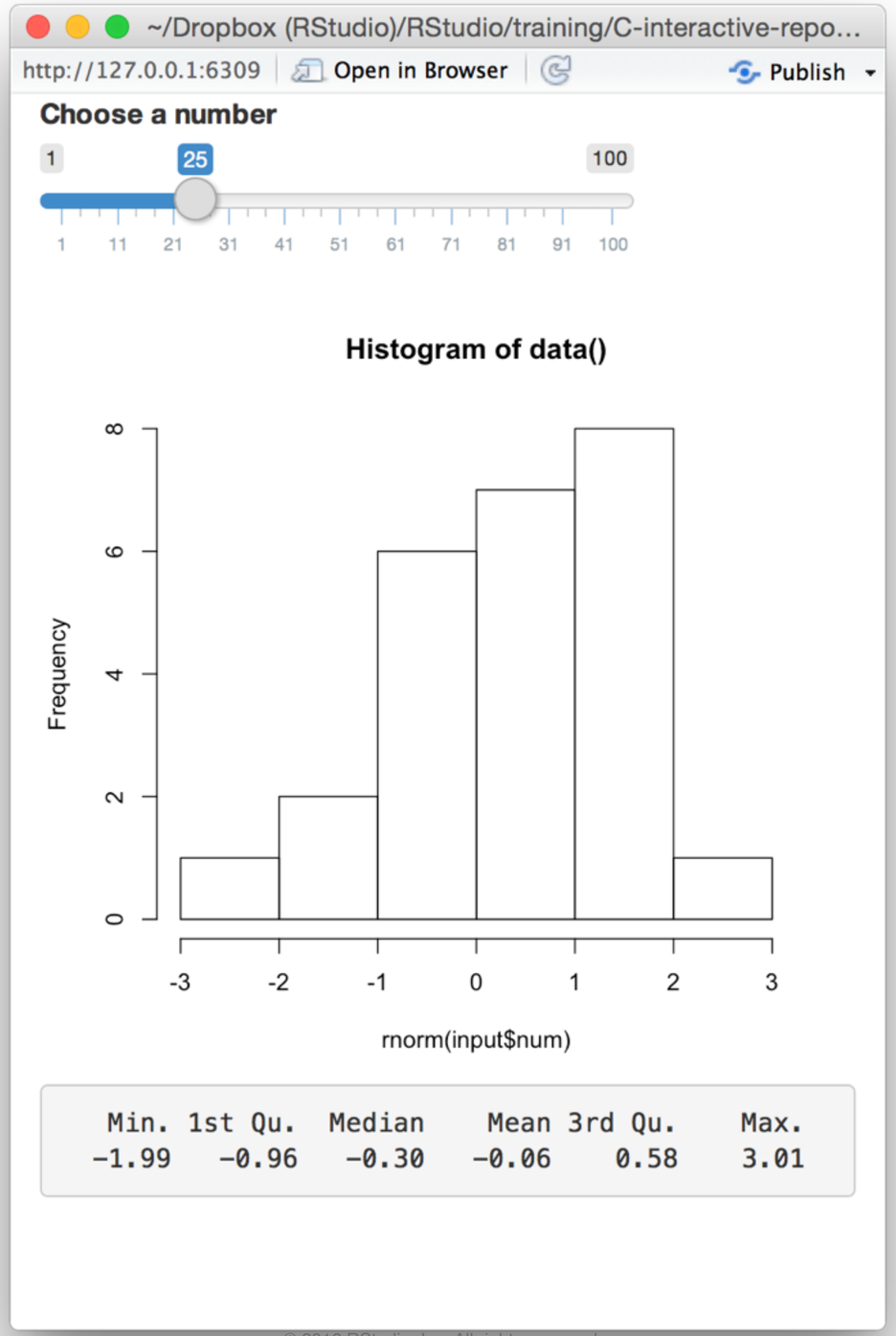
```
renderPlot(isolate({ hist(rnorm(input$num)) }))
```

- Calls reactive values *without taking a dependency*
- Never notified, never reacts.

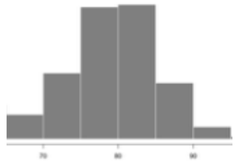




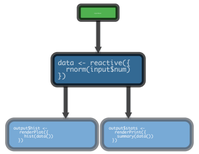




Use…



render() to make an **object to display** in the UI.

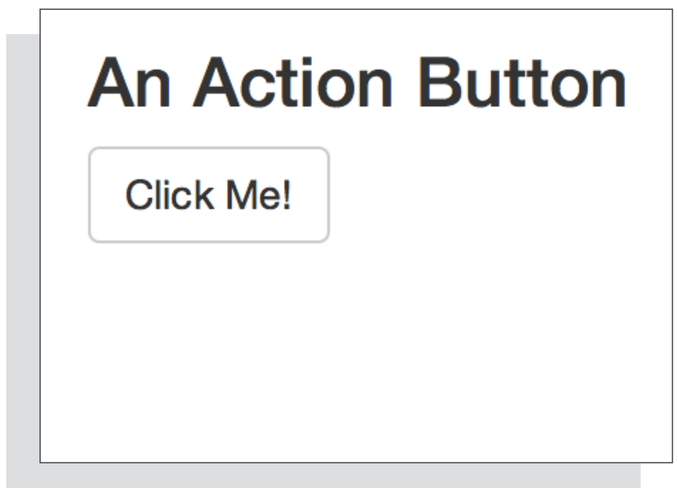


reactive() to make an **object to use** in downstream code.

isolate() to return a **non-reactive object**.

Delay reactions with eventReactive()

Action buttons



input function

Notice: Id not ID

input name (for internal use)

label to display

```
actionButton(inputId = "go", label = "Click Me!")
```

The value of an action button increases by one each time it is pressed.

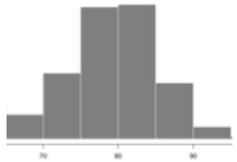
eventReactive()

Let's you control when an expression is invalidated

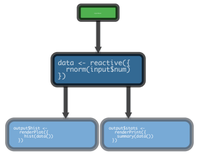
```
data <- eventReactive(input$go, { rnorm(input$num) })
```

- Takes a dependency on only reactive values in its first argument (eventExpr)
- Only notifies downstream objects when invalidated

Use…



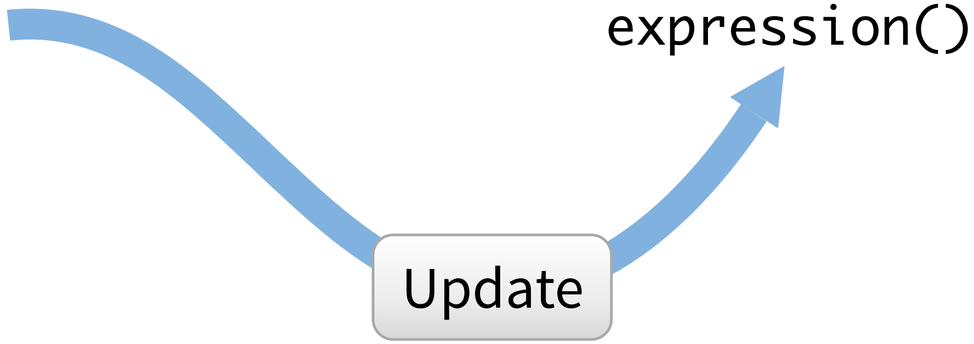
render() to make an **object to display** in the UI.

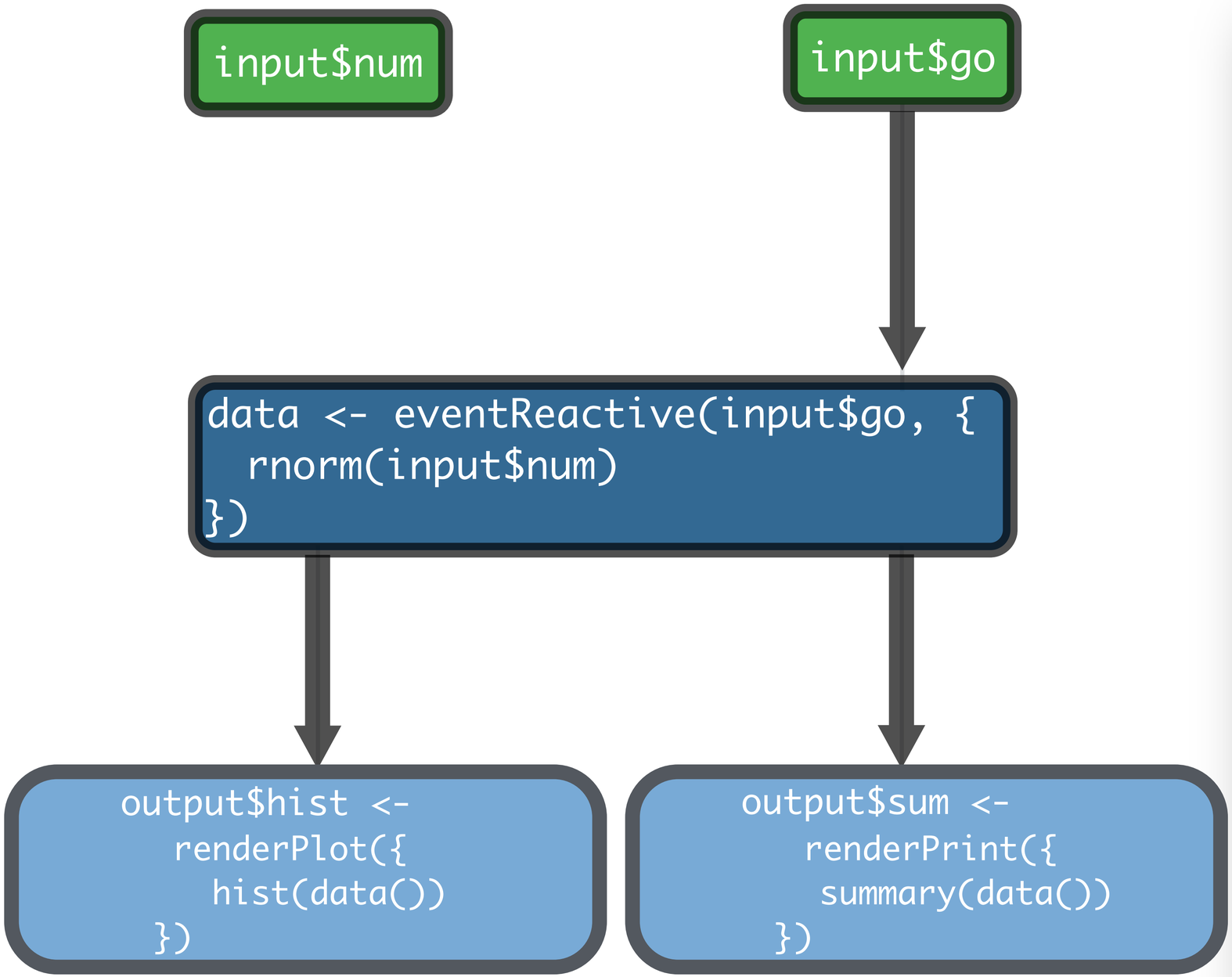


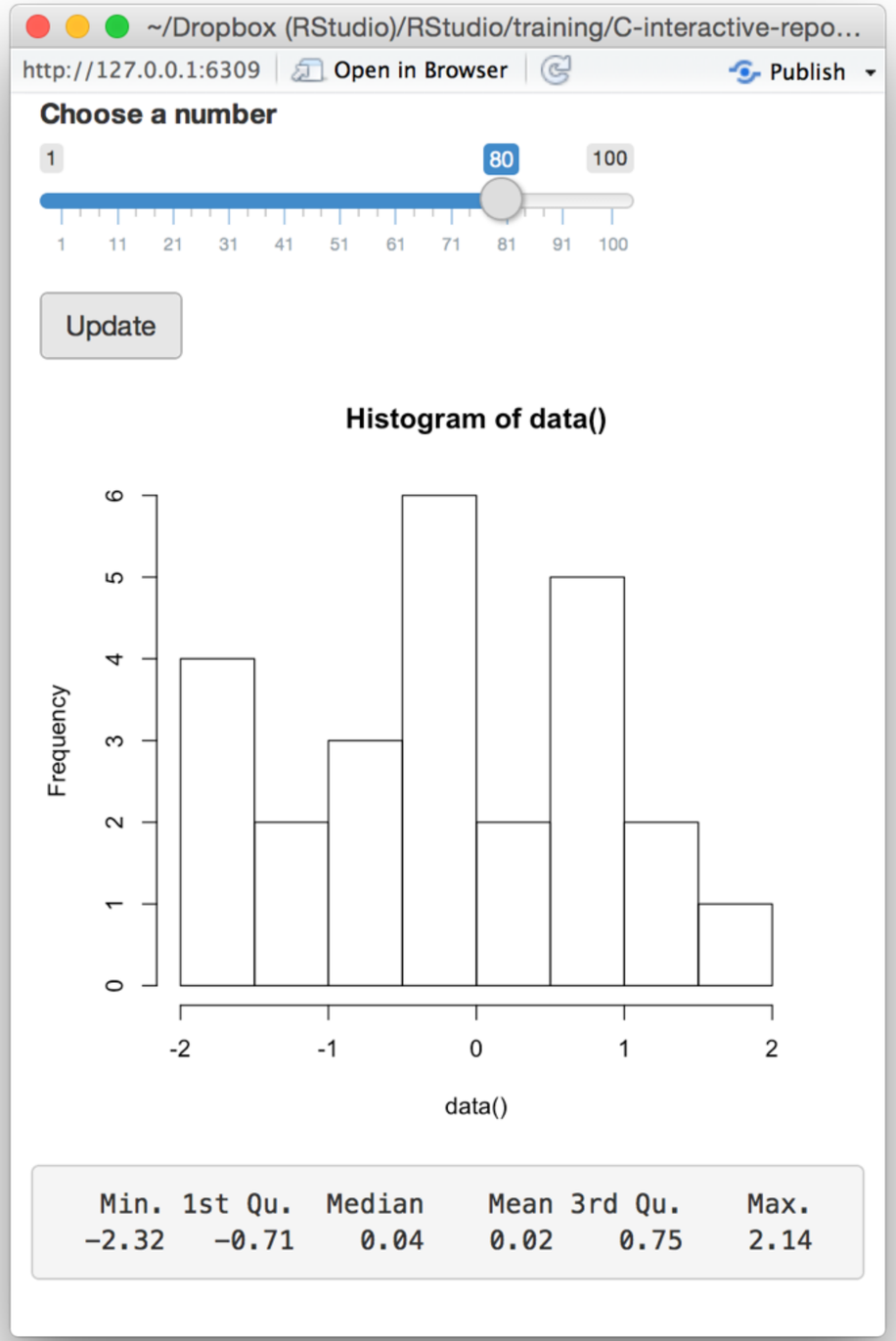
reactive() to make an **object to use** in downstream code.

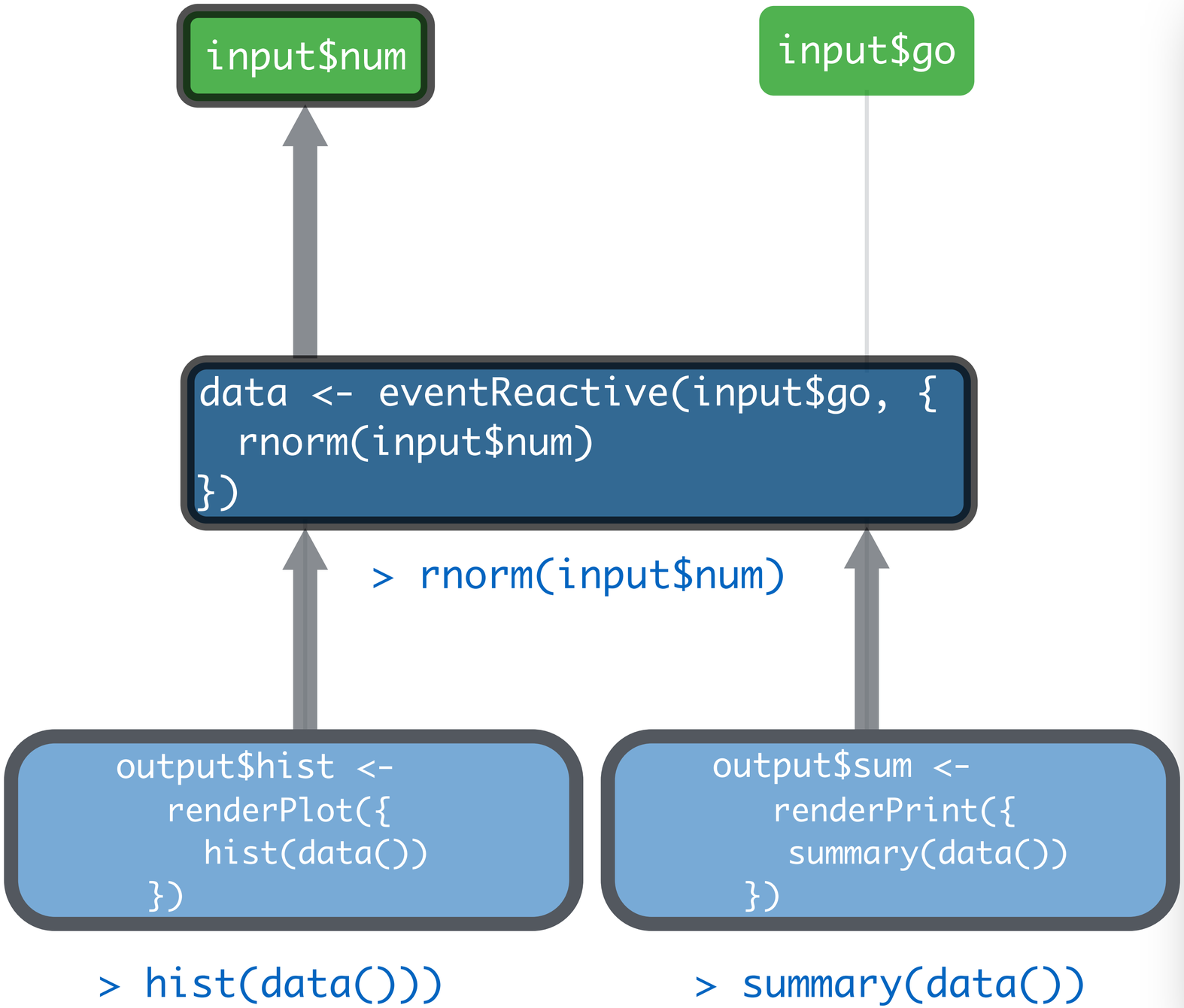
isolate() to return a **non-reactive object**.

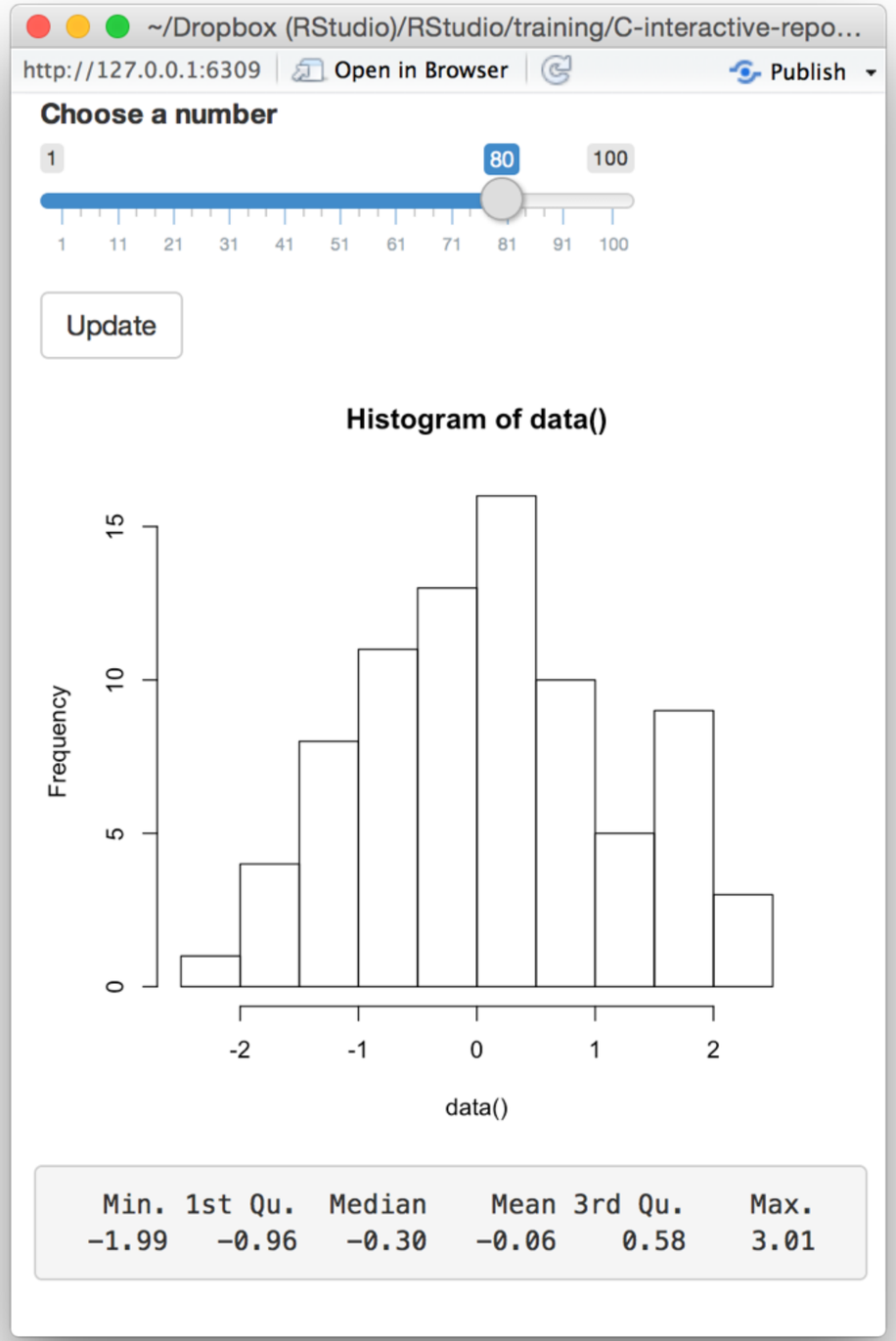
eventReactive() to **delay a reaction**.



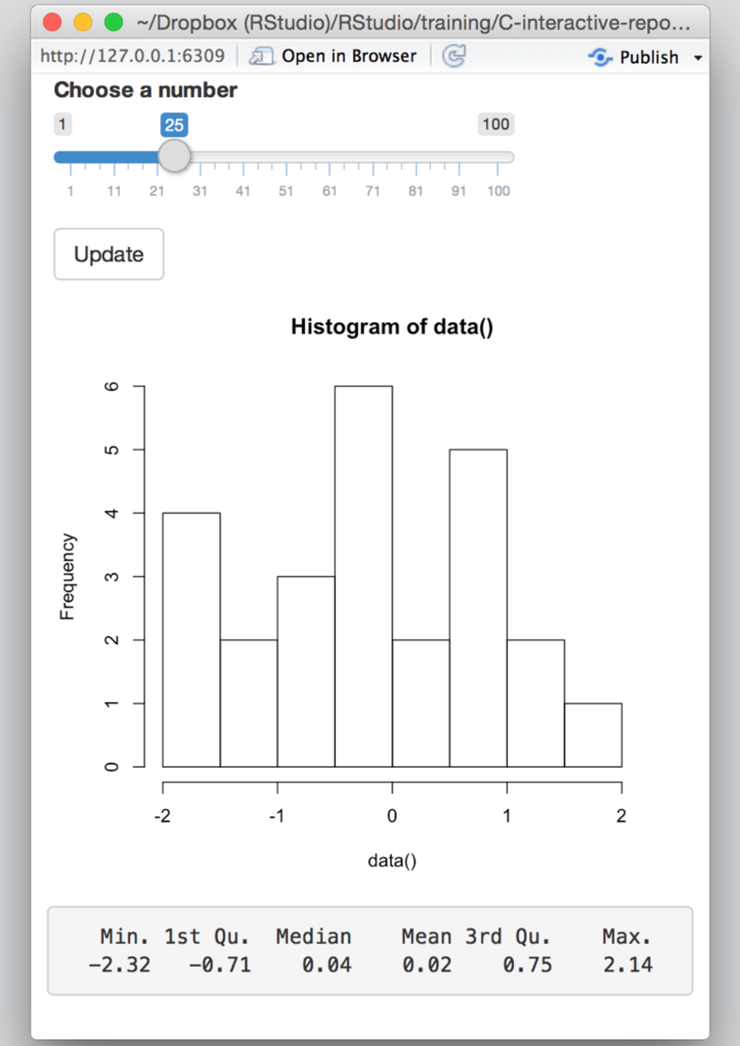








Your Turn

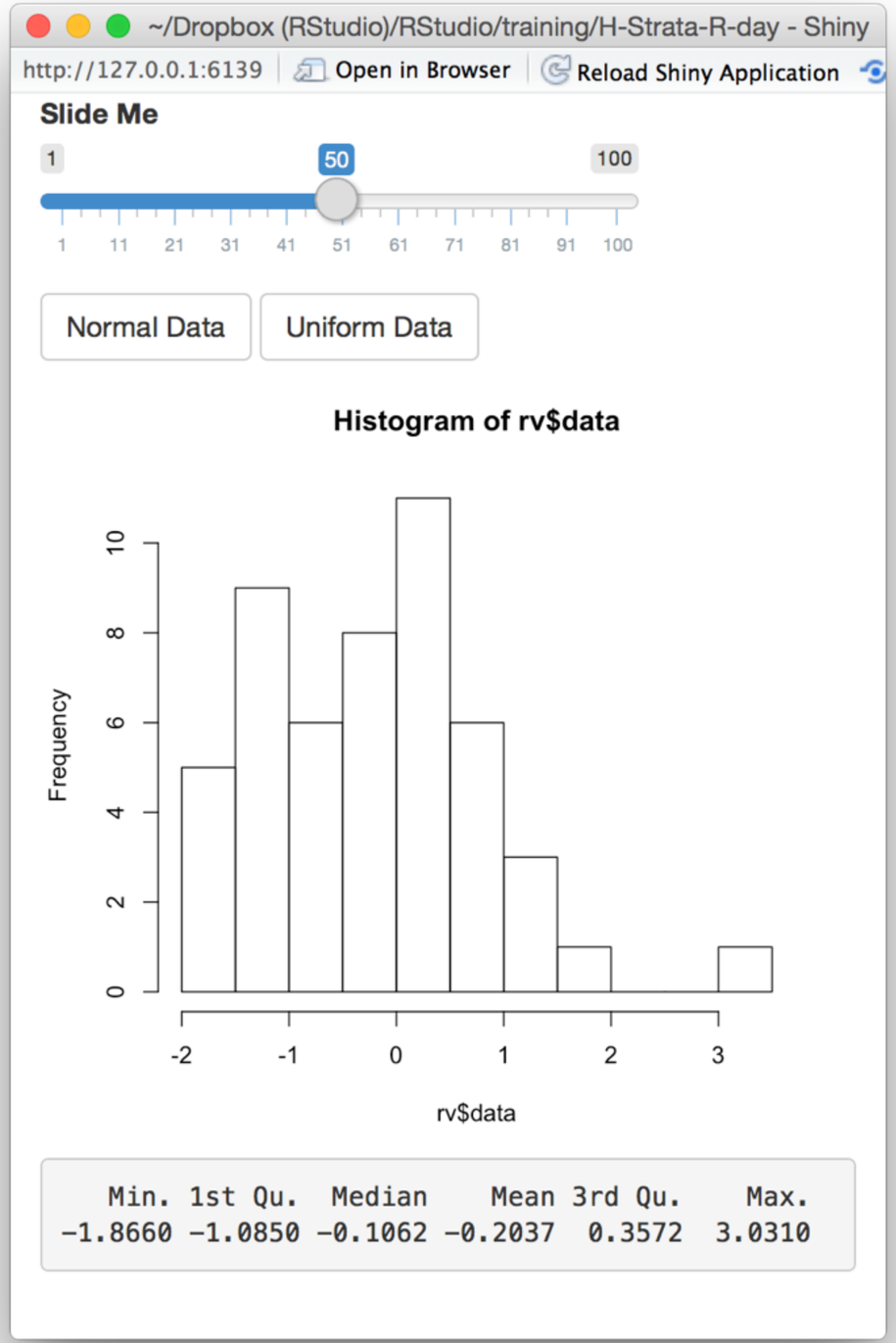


Add an **actionButton()** to the app. Then replace **reactive()** with **eventReactive()** so that the app only responds when the button is clicked.

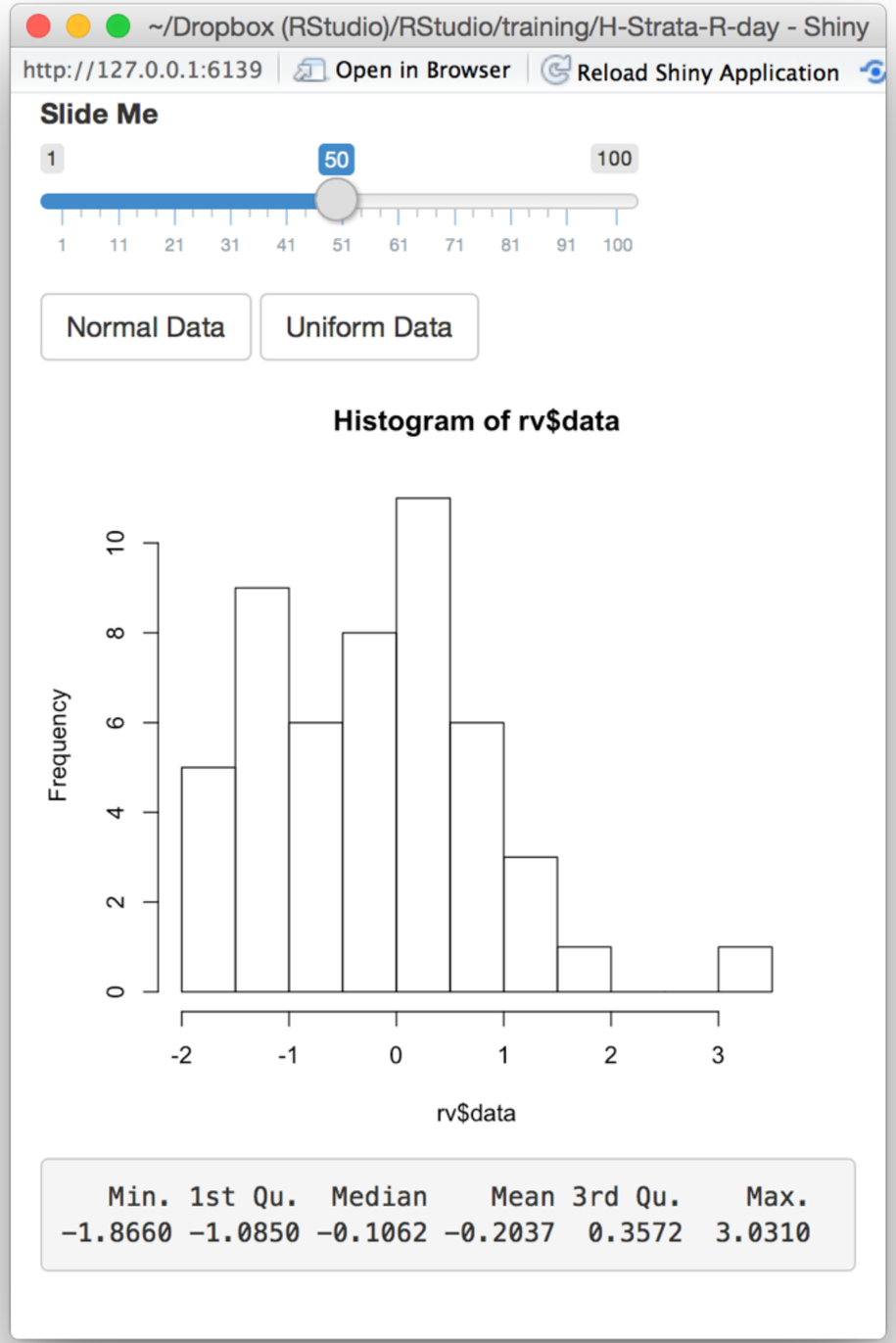
Ensure that you can predict how the app will work.

Trigger side effects with observeEvent()

```r
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  actionButton("norm", "Normal Data"),
  actionButton("unif", "Uniform Data"),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   rv <- reactiveValues(data = rnorm(50))

   observeEvent(input$norm, {rv$data <- rnorm(input$num)})
   observeEvent(input$unif, {rv$data <- runif(input$num)})

   output$hist <- renderPlot({hist(rv$data)})
   output$sum <- renderPrint({summary(rv$data)})
}
shinyApp(ui = ui, server = server)
```



```
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  actionButton("norm", "Normal Data"),
  actionButton("unif", "Uniform Data"),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   rv <- reactiveValues(data = rnorm(50))

   observeEvent(input$norm, {rv$data <- rnorm(input$num)})
   observeEvent(input$unif, {rv$data <- runif(input$num)})

   output$hist <- renderPlot({hist(rv$data)})
   output$sum <- renderPrint({summary(rv$data)})
}
shinyApp(ui = ui, server = server)
```



observeEvent()

Triggers code to run.

```
observeEvent(input$norm, {rv$data <- rnorm(input$num)})
```

- Takes a dependency on only reactive values in its first argument (eventExpr)
- Runs chunk of code when invalidated

input$norm

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

```
> rv$data <-
+ rnorm(input$num)
```

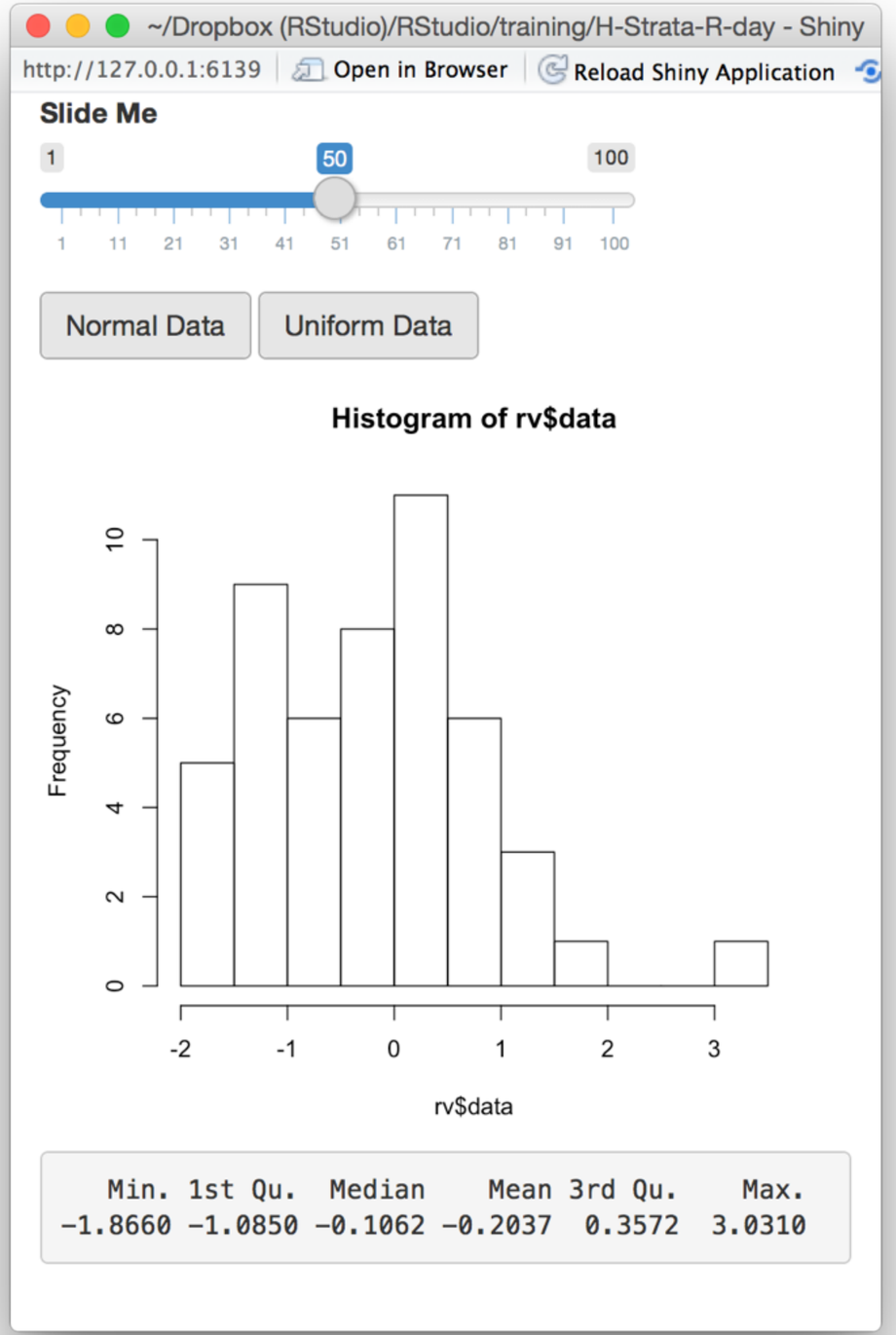
```
output$hist <-
   renderPlot({
     hist(data())
  })
```

input$unif

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

```
> rv$data <-
+ runif(input$num)
```

```
output$sum <-
   renderPrint({
    summary(data())
  })
```

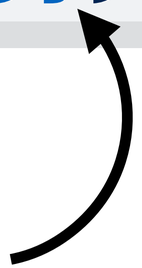


observe()

Also triggers code to run on server.

Uses same syntax as render*(), reactive(), and isolate()

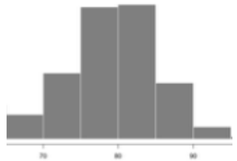
```
observe({rv$data <- rnorm(input$num)})
```



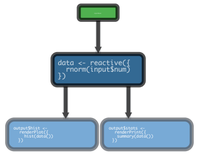
- Takes a dependency on every reactive value in code chunk*
- Reruns entire code chunk when invalidated

**Unless you use isolate()*

Use…



render() to make an **object to display** in the UI.



reactive() to make an **object to use** in downstream code.

isolate() to return a **non-reactive object**.

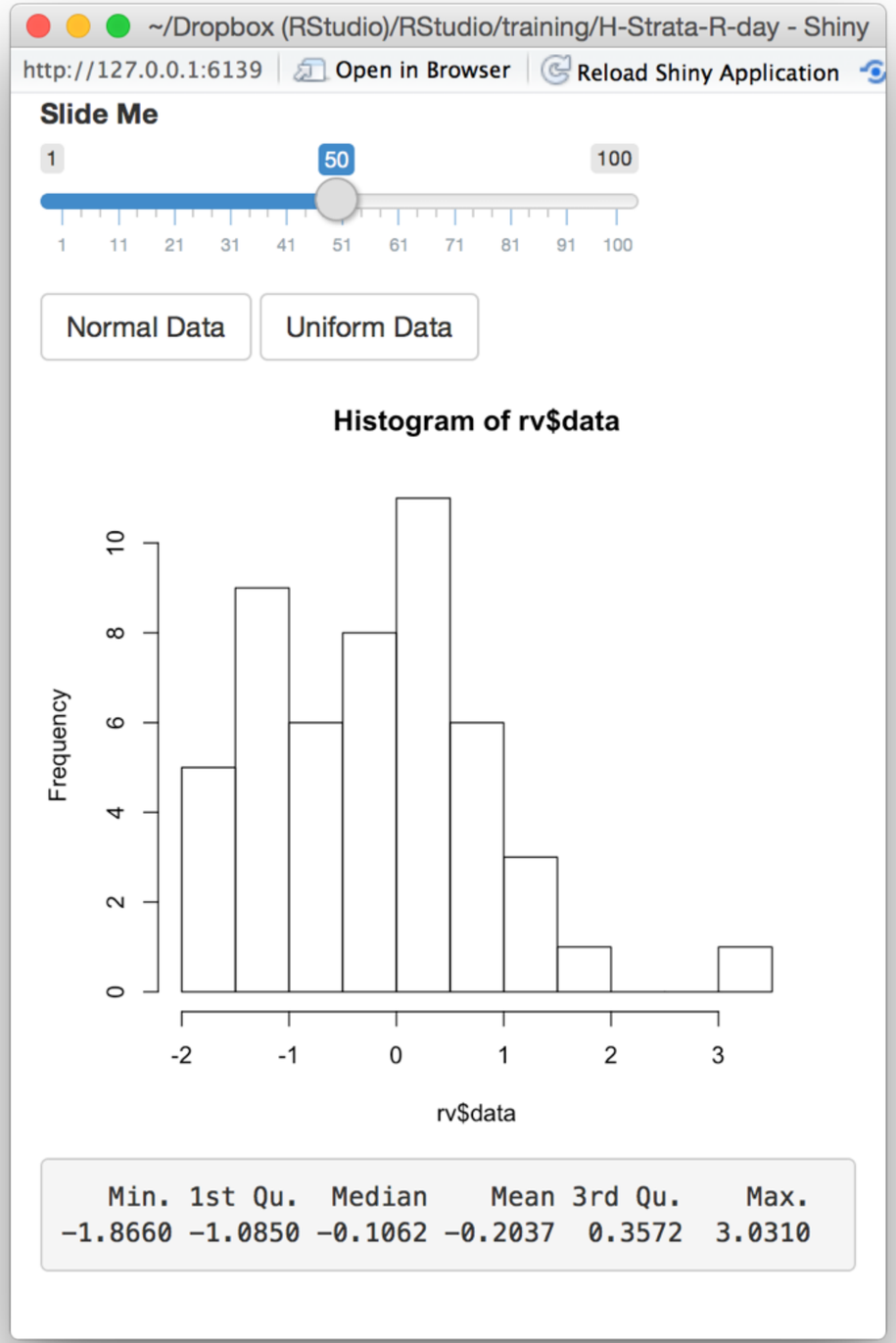
eventReactive() to **delay a reaction**.

observeEvent() or **observe()** to **trigger code**.

Maintain state with

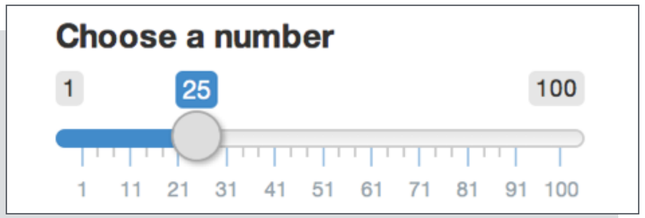
reactiveValues()

```
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  actionButton("norm", "Normal Data"),
  actionButton("unif", "Uniform Data"),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   rv <- reactiveValues(data = rnorm(50))

   observeEvent(input$norm, {rv$data <- rnorm(input$num)})
   observeEvent(input$unif, {rv$data <- runif(input$num)})

   output$hist <- renderPlot({hist(rv$data)})
   output$sum <- renderPrint({summary(rv$data)})
}
shinyApp(ui = ui, server = server)
```

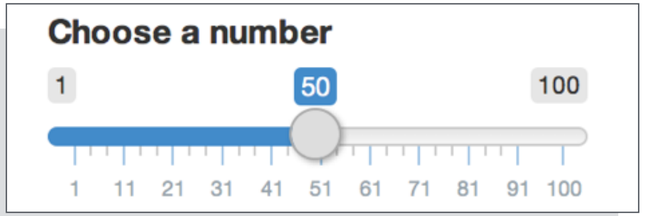


Reactive values

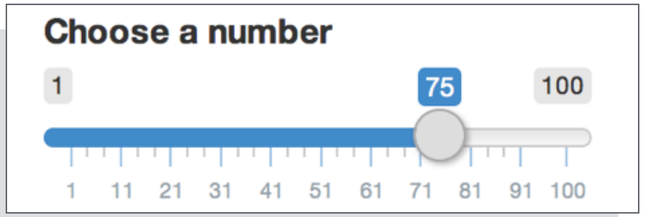
The input list contains values that change whenever a user changes an input.




```
input$num = 25
```




```
input$num = 50
```




```
input$num = 75
```

You **cannot** set these values in your code

reactiveValues()

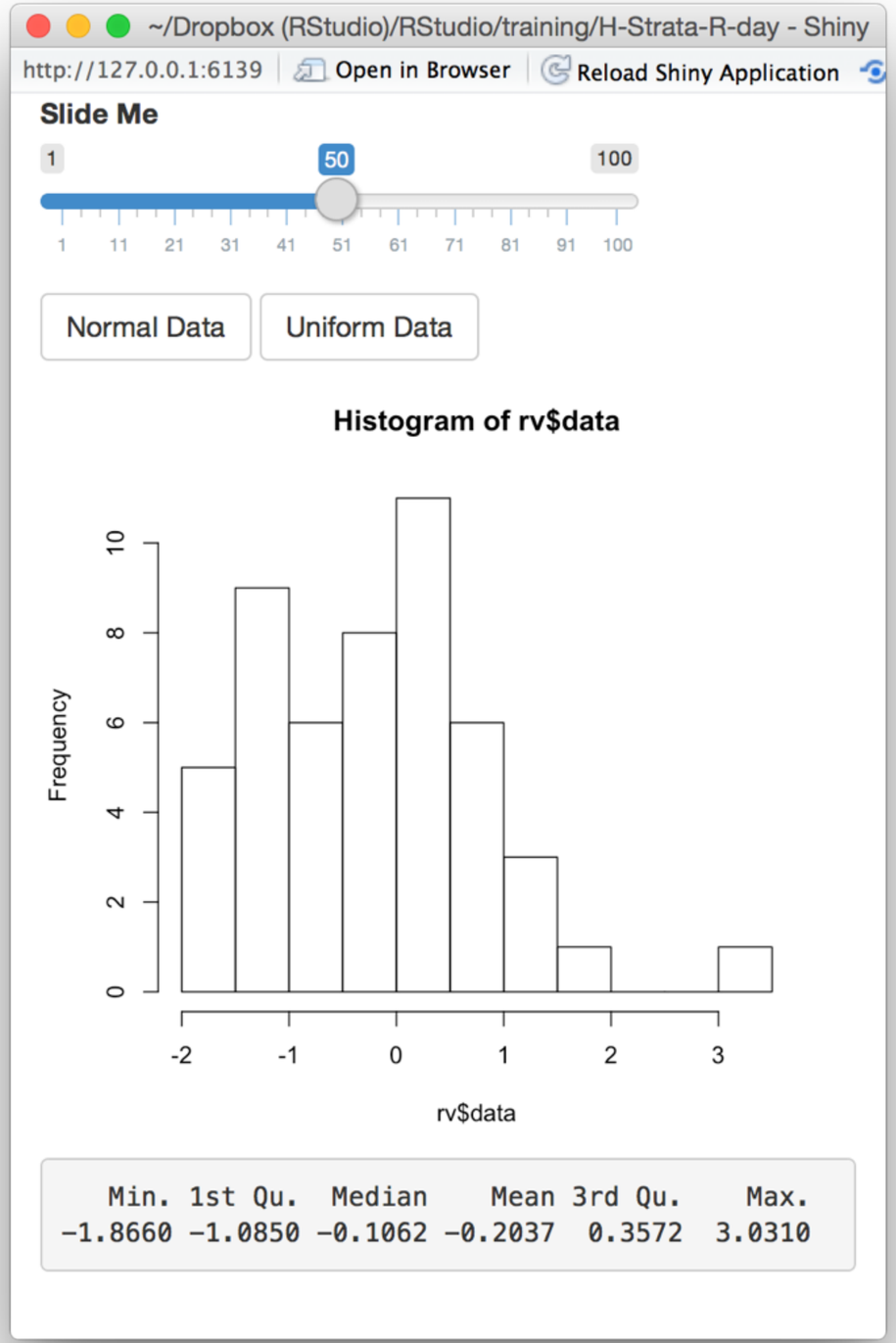
Creates a list of reactive values that you can manipulate

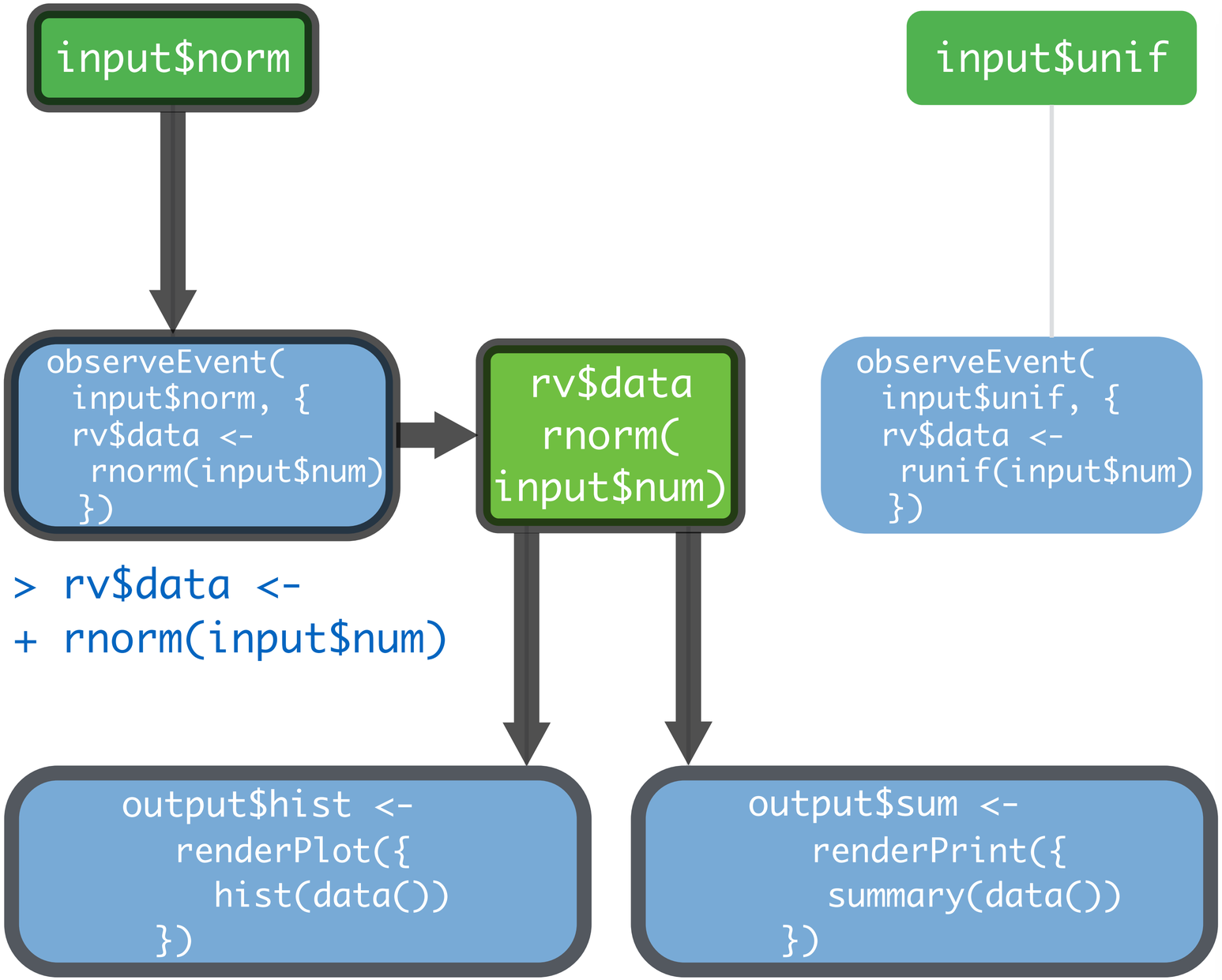
```
rv <- reactiveValues(data = rnorm(100))
```

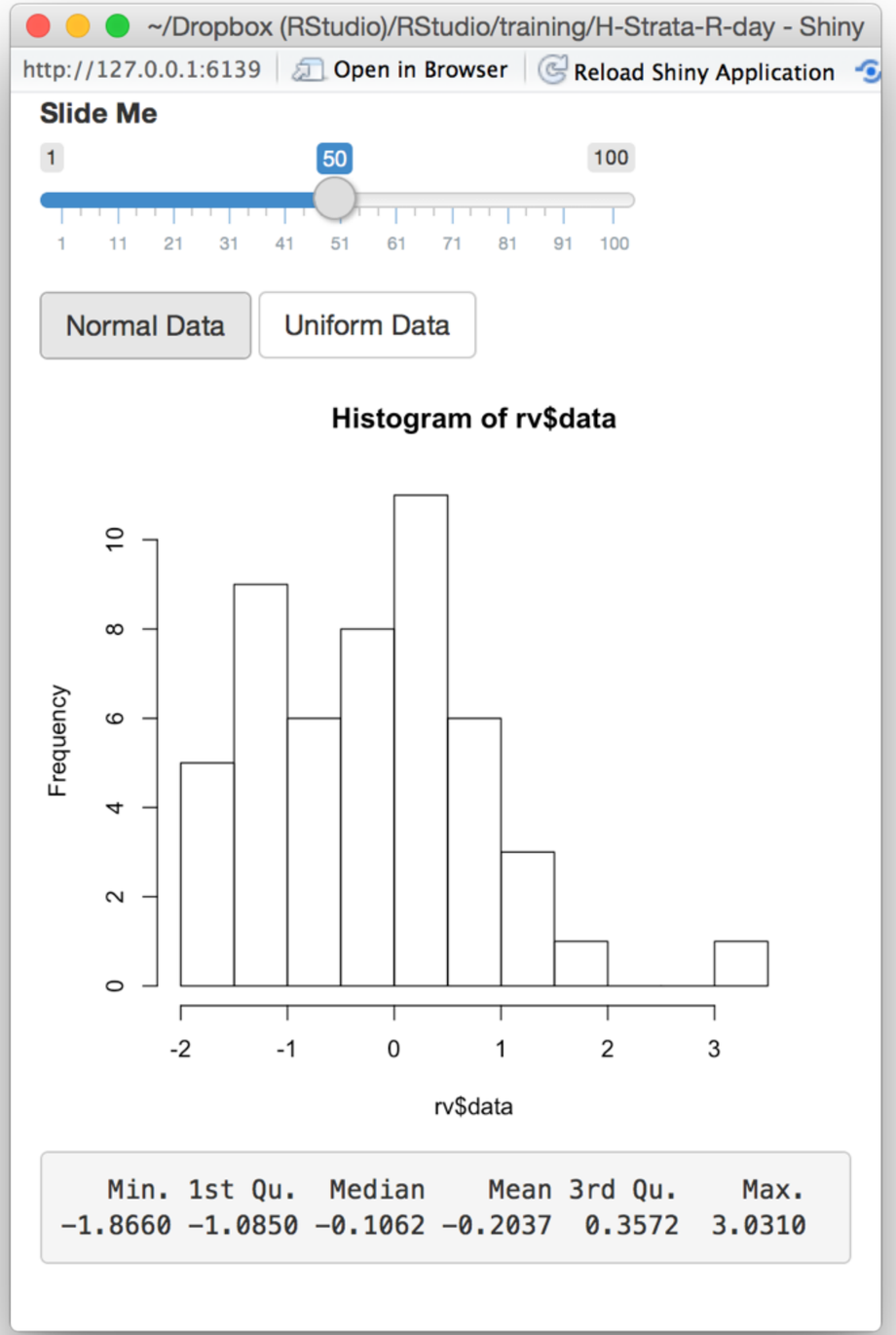
(optional) elements to add to the list

- Builds a list of reactive values
- Reactive objects will take dependencies on these values in the usual ways

```
ui <- fluidPage(
  sliderInput("num", "Slide Me", 1, 100, 50),
  actionButton("norm", "Normal Data"),
  actionButton("unif", "Uniform Data"),
  plotOutput("hist"),
  verbatimTextOutput("sum")
)
server <- function(input, output) {
   rv <- reactiveValues(data = rnorm(50))

   observeEvent(input$norm, {rv$data <- rnorm(input$num)})
   observeEvent(input$unif, {rv$data <- runif(input$num)})

   output$hist <- renderPlot({hist(rv$data)})
   output$sum <- renderPrint({summary(rv$data)})
}
shinyApp(ui = ui, server = server)
```







input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

```
rv$data
rnorm(
input$num)
```

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

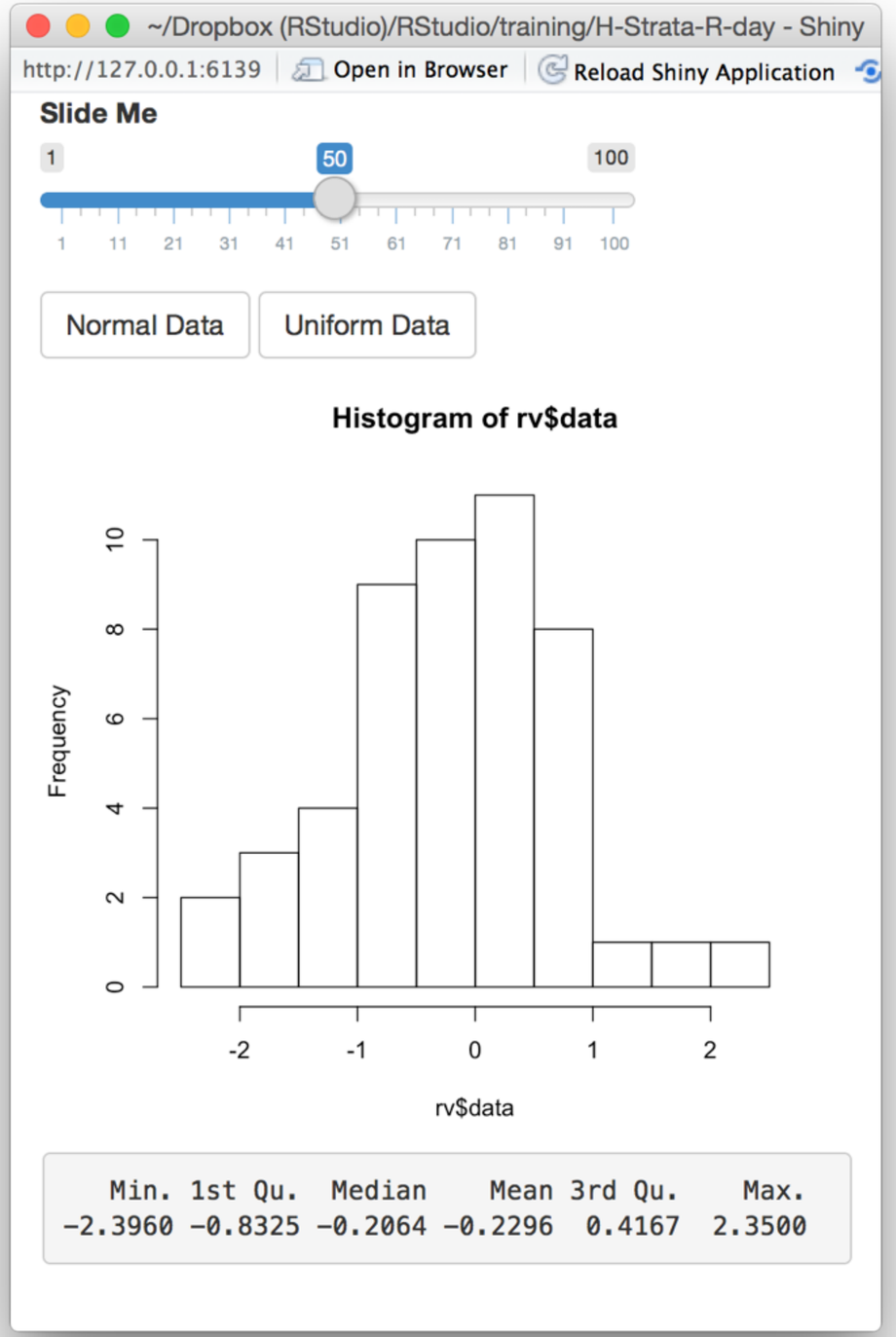
```
> rv$data <-
+ rnorm(input$num)
```

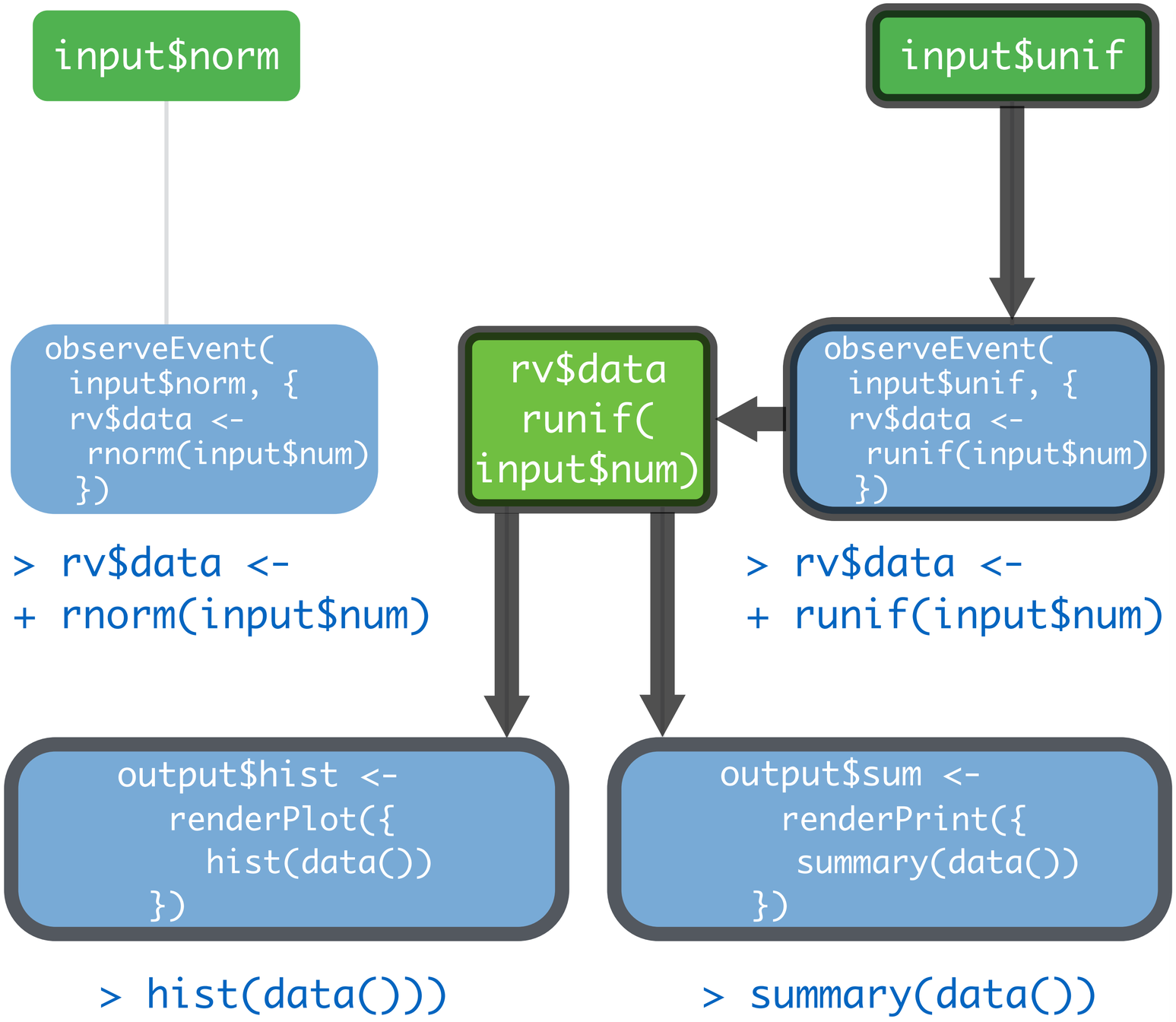
```
output$hist <-
   renderPlot({
     hist(data())
  })
```

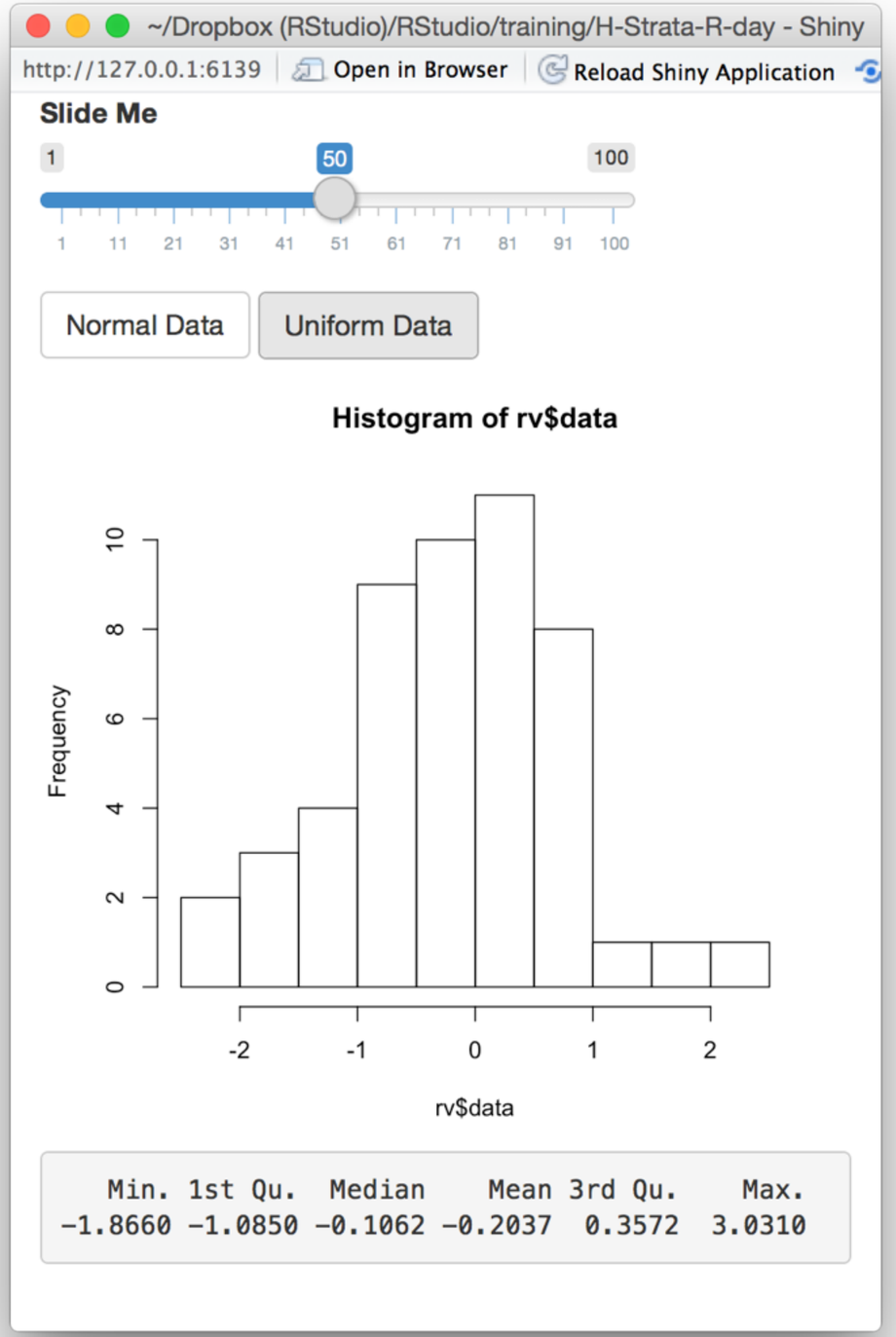
```
output$sum <-
   renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```

```
> summary(data())
```







input$norm

input$unif

```
observeEvent(
  input$norm, {
  rv$data <-
   rnorm(input$num)
  })
```

```
rv$data
runif(
input$num)
```

```
observeEvent(
  input$unif, {
  rv$data <-
   runif(input$num)
  })
```

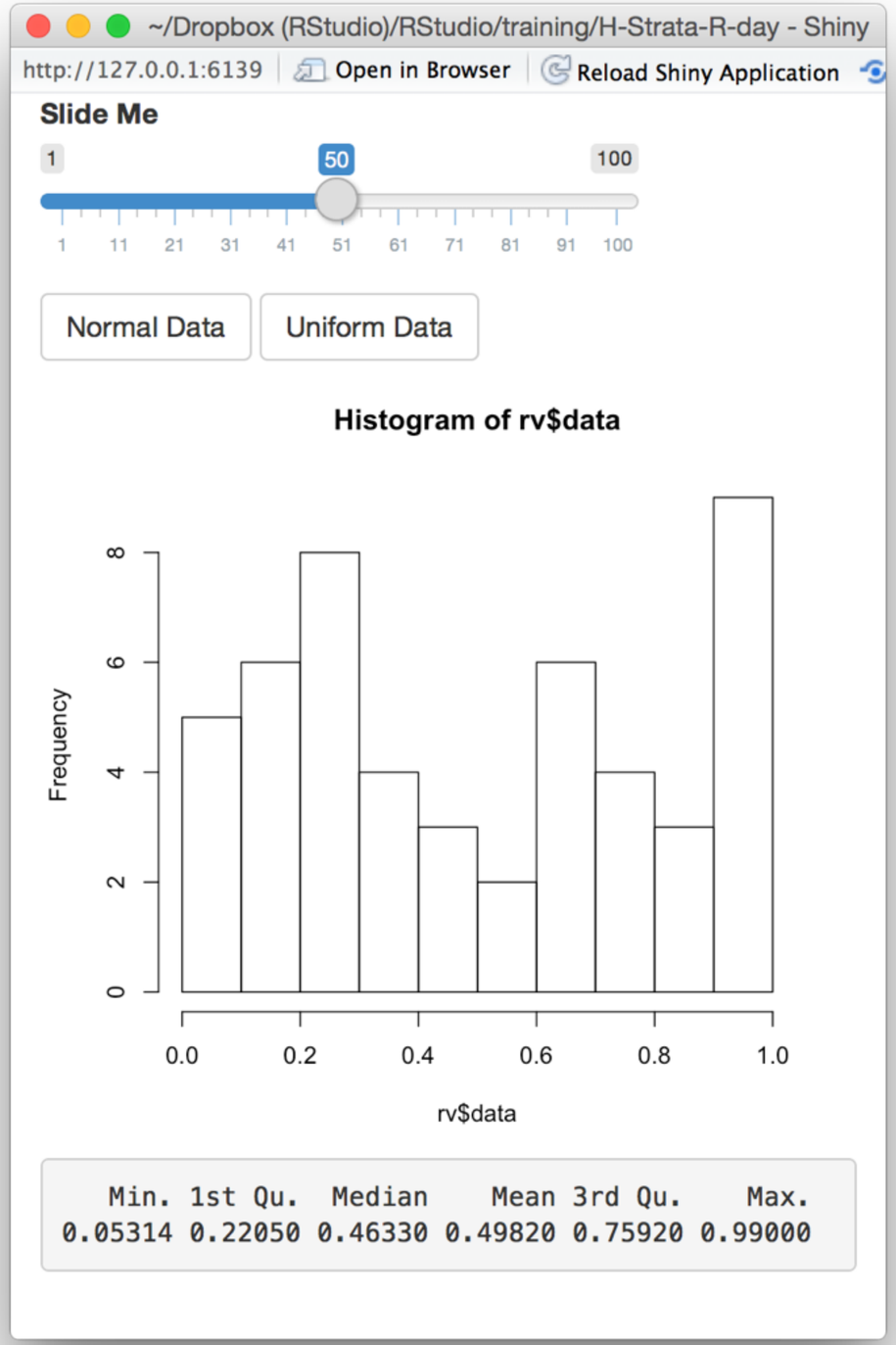
```
> rv$data <-
+ runif(input$num)
```

```
output$hist <-
   renderPlot({
     hist(data())
  })
```

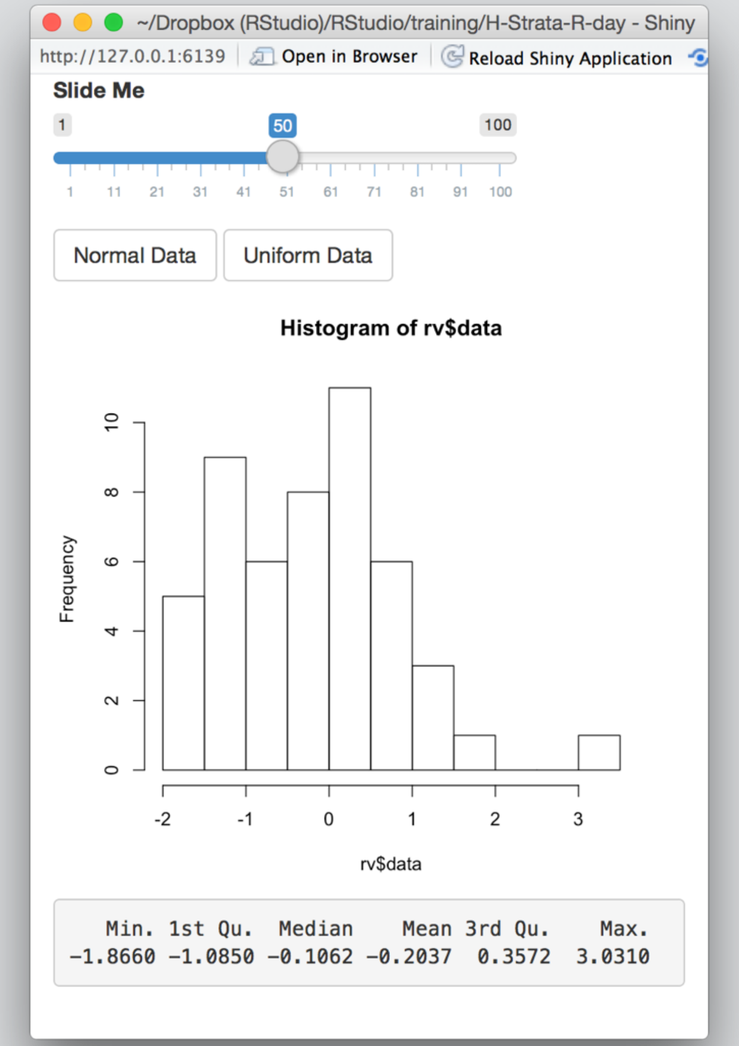
```
output$sum <-
   renderPrint({
    summary(data())
  })
```

```
> hist(data()))
```

```
> summary(data())
```



Your Turn

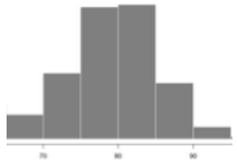


Edit your app to create the app we've examined. Then ensure that it works.

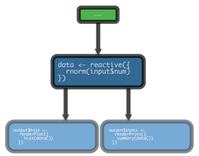
You will need to create:

- two action buttons
- a list of reactive values with one element
- two observers

Use…



render() to make an **object to display** in the UI.



reactive() to make an **object to use** in downstream code.

isolate() to return a **non-reactive object**.

eventReactive() to **delay a reaction**.

observeEvent() or **observe()** to **trigger code**.

```
rv$data <-
```

reactiveValues() to **make your own** reactive values.

observers vs.
reactive expressions

	Reactive Expression	Observer
Implicit dependencies	`reactive()`	`observe()`
Explicit dependencies	`eventReactive()`	`observeEvent()`

Side Effects

Functions can be executed for one or both of the following reasons:

1. You want its return value.
2. You want it to have *some other effect*.

These are (a bit misleadingly) called *side effects*. Any effect that is not the return value is a side effect.

Quiz

Does this code have a side effect?

```
foo <- function(a, b) {
  (b - a) / a
}

foo()
```

Quiz

Does this code have a side effect?

```
foo <- function(a, b) {
  (b - a) / a
}

foo()
```

No

Quiz

Does this code have a side effect?

```
foo <- function() {
  options(digits.secs = 6)
  as.character(Sys.time())
}

foo()
```

Quiz

Does this code have a side effect?

```
foo <- function() {
  options(digits.secs = 6)
  as.character(Sys.time())
}

foo()
```

Yes

Quiz

Does this code have a side effect?

```
foo <- function(df) {
  df$foo <- factor(df$foo)
  df
}
foo()
```

Quiz

Does this code have a side effect?

```
foo <- function(df) {
  df$foo <- factor(df$foo)
  df
}
foo()
```

No

Quiz

Does this code have a side effect?

```
foo <- function(values) {
  assign("values", values, envir = .globalEnv)

}
foo()
```

Quiz

Does this code have a side effect?

```
foo <- function(values) {
  assign("values", values, envir = .globalEnv)

}
foo()
```

Yes

Quiz

Does this code have a side effect?

```
foo <- function(values) {
  runif(10)
}
foo()
```

Quiz

Does this code have a side effect?

```
foo <- function(values) {
  runif(10)
}
foo()
```

Not Really…

Reactive expressions vs. observers

Reactive Expression	Observer
Callable	Not Callable
Returns a value	No return value
Lazy (passive)	Eager (active)
Cached	N/A
For Calculations	**For Side Effects**

Reactive expressions vs. observers

Reactive Expression

Callable

Returns a value

Lazy (passive)

Cached

For Calculations

Did the side effect occur?

- Buggy
- Hard to reason about

Reactive expressions vs. observers

How do you retrieve the result of the calculation?

- Buggy
- Hard to reason about

Did the calculation need to happen?

- Slow

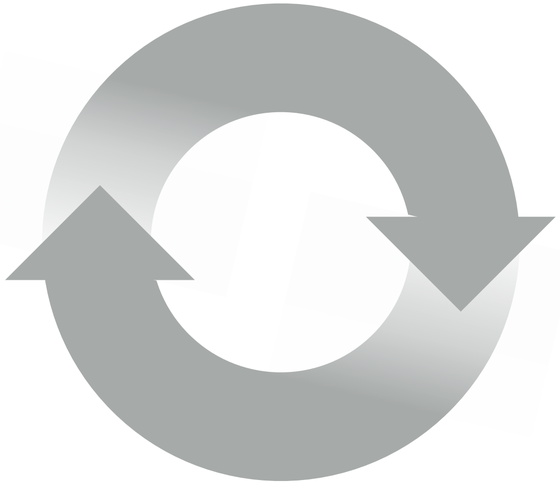
Observer
Not Callable
No return value
Eager (active)
N/A
For Side Effects

Calculations = reactive()

Side Effects = observer()

Shiny

+



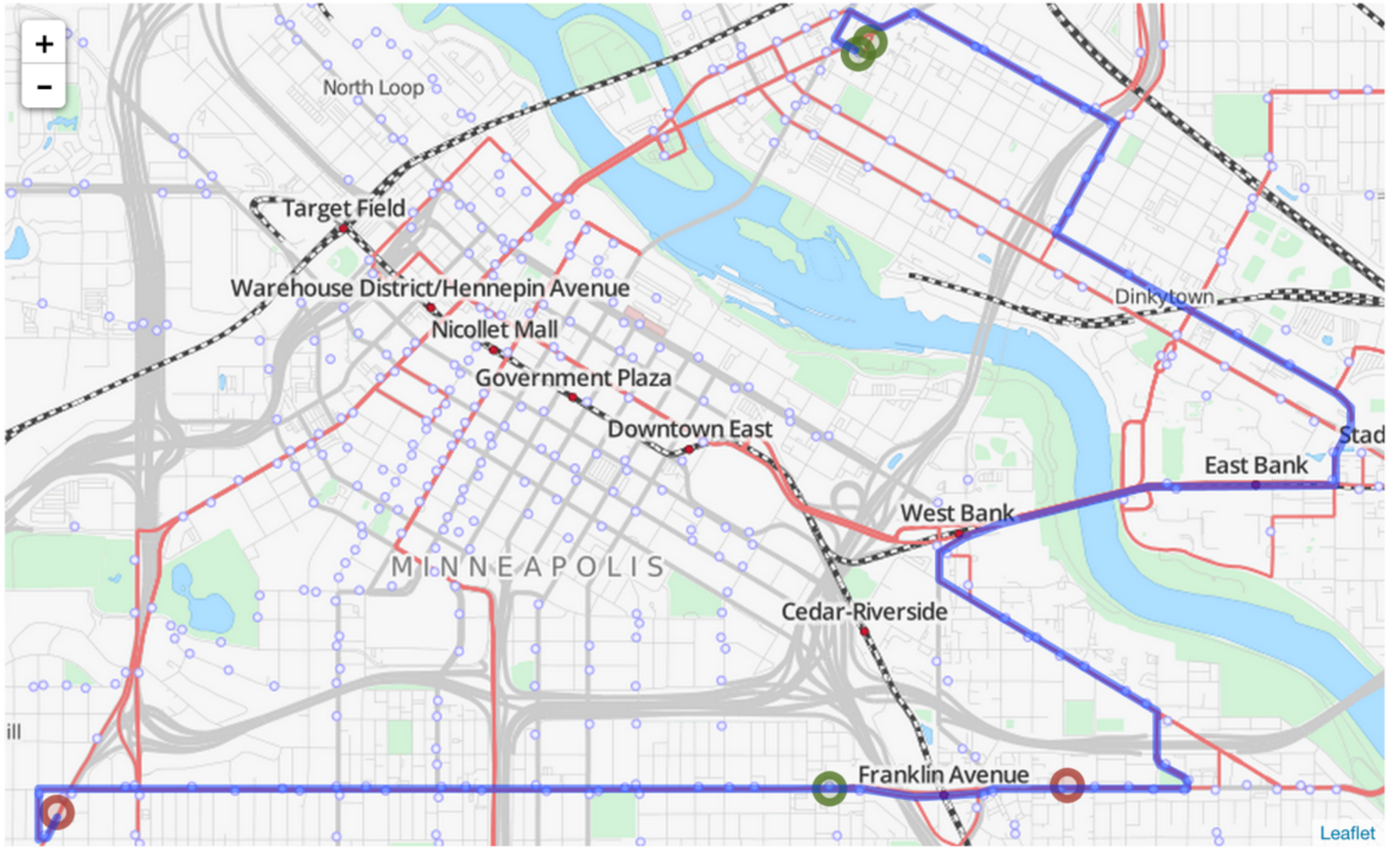
Reactive
Programming

+

Web based
User Interface

Shiny UI

Twin Cities Buses



Color	Direction	Number of vehicles
	Northbound	0
	Southbound	0

Route

2

Show

- [x] Northbound
- [x] Southbound
- [x] Eastbound
- [x] Westbound

Note: a route number can have several different trips, each with a different path. Only the most commonly-used path will be displayed on the map.

Zoom to fit buses

Refresh interval

1 minute

Data refreshed 25 seconds ago.

Refresh now

Source data updates every 30 seconds.

Shiny Widgets Gallery

For each widget below, the Current Value(s) window displays the value that the widget provides to shinyServer. Notice that the values change as you interact with the widgets.

Action button

Action

Current Value:

```
[1] 0
attr(,"class")
[1] "numeric"                    "shinyAct
ionButtonValue"
```

See Code

Single checkbox

☑ Choice A

Current Value:

```
[1] TRUE
```

See Code

Checkbox group

☑ Choice 1
☐ Choice 2
☐ Choice 3

Current Values:

```
[1] "1"
```

See Code

Date input

2014-01-01

Current Value:

```
[1] "2014-01-01"
```

Date range

2015-02-09 to 2015-02-09

Current Values:

```
[1] "2015-02-09" "2015-02-09"
```

File input

Choose File No file chosen

Current Value:

```
NULL
```

intRo

DATA

Sources

Transform

SUMMARIES

Graphical

Numerical

STATISTICS

Contingency

Regression

T test

Plot Type

Histogram

X Variable (x)

displ

X Min

X Max

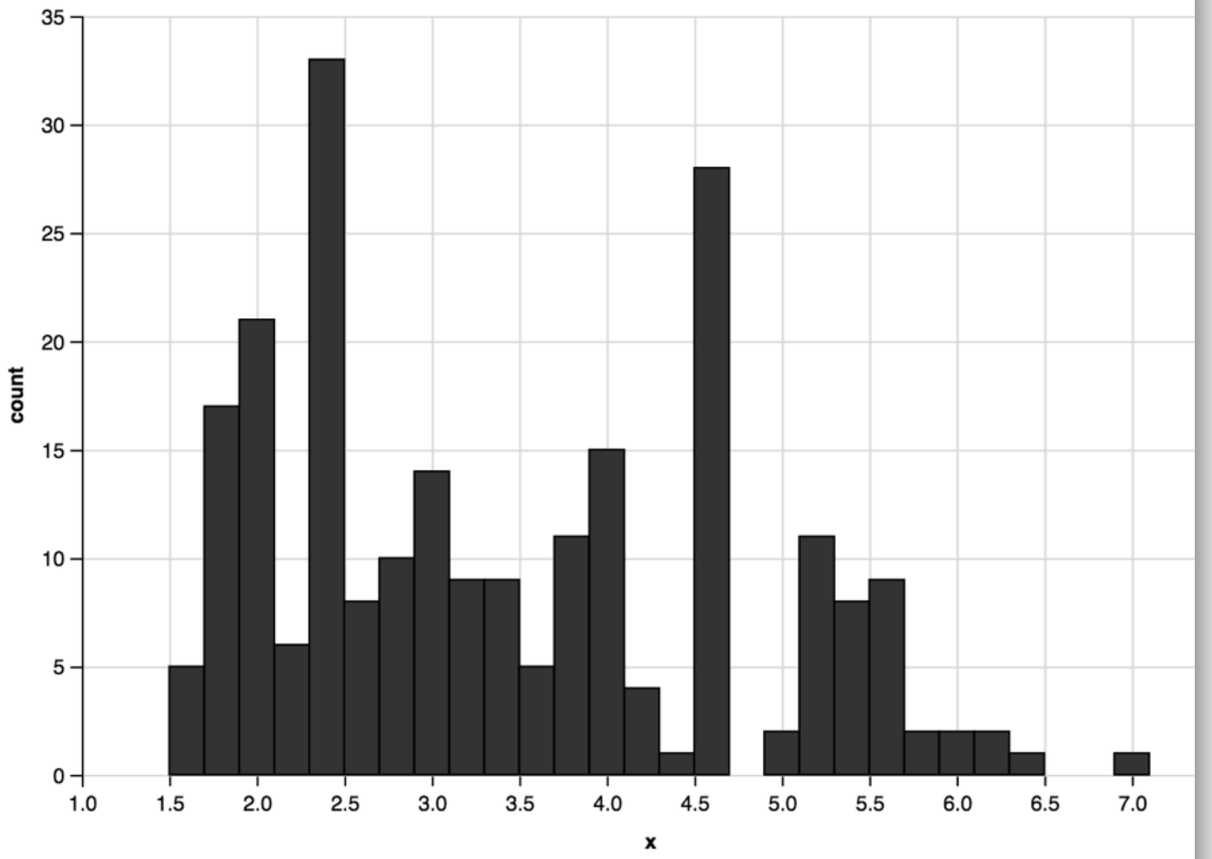
Y Min

Y Max

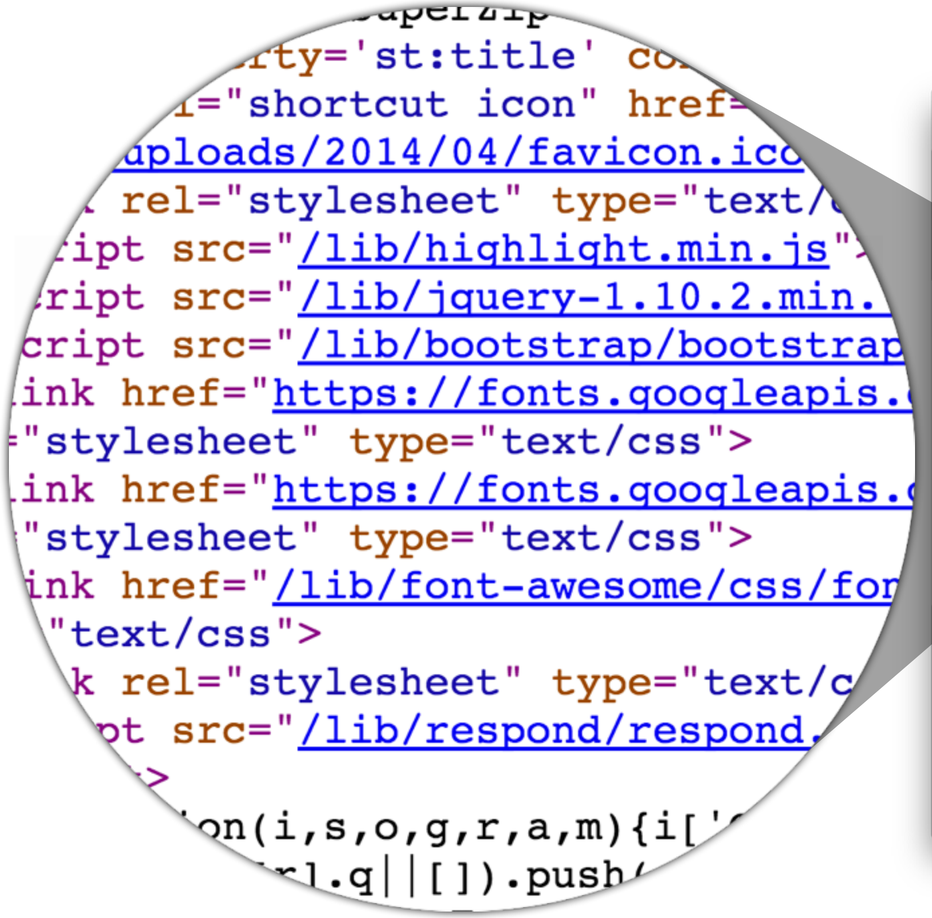
Bin Width

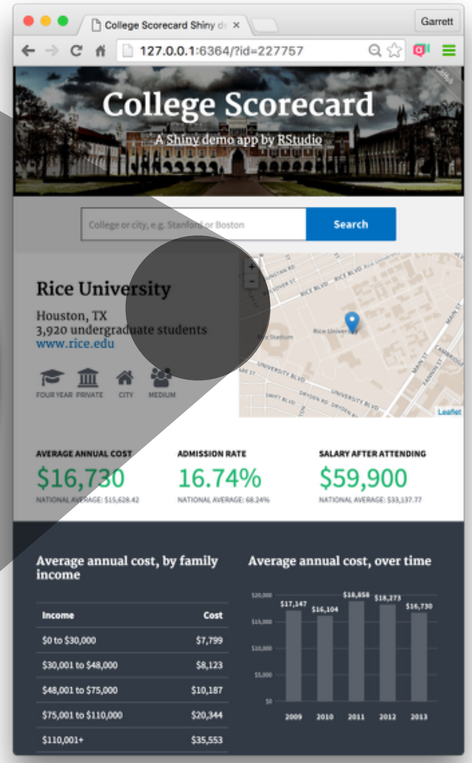
Store Graphical Result

X Variable: displ



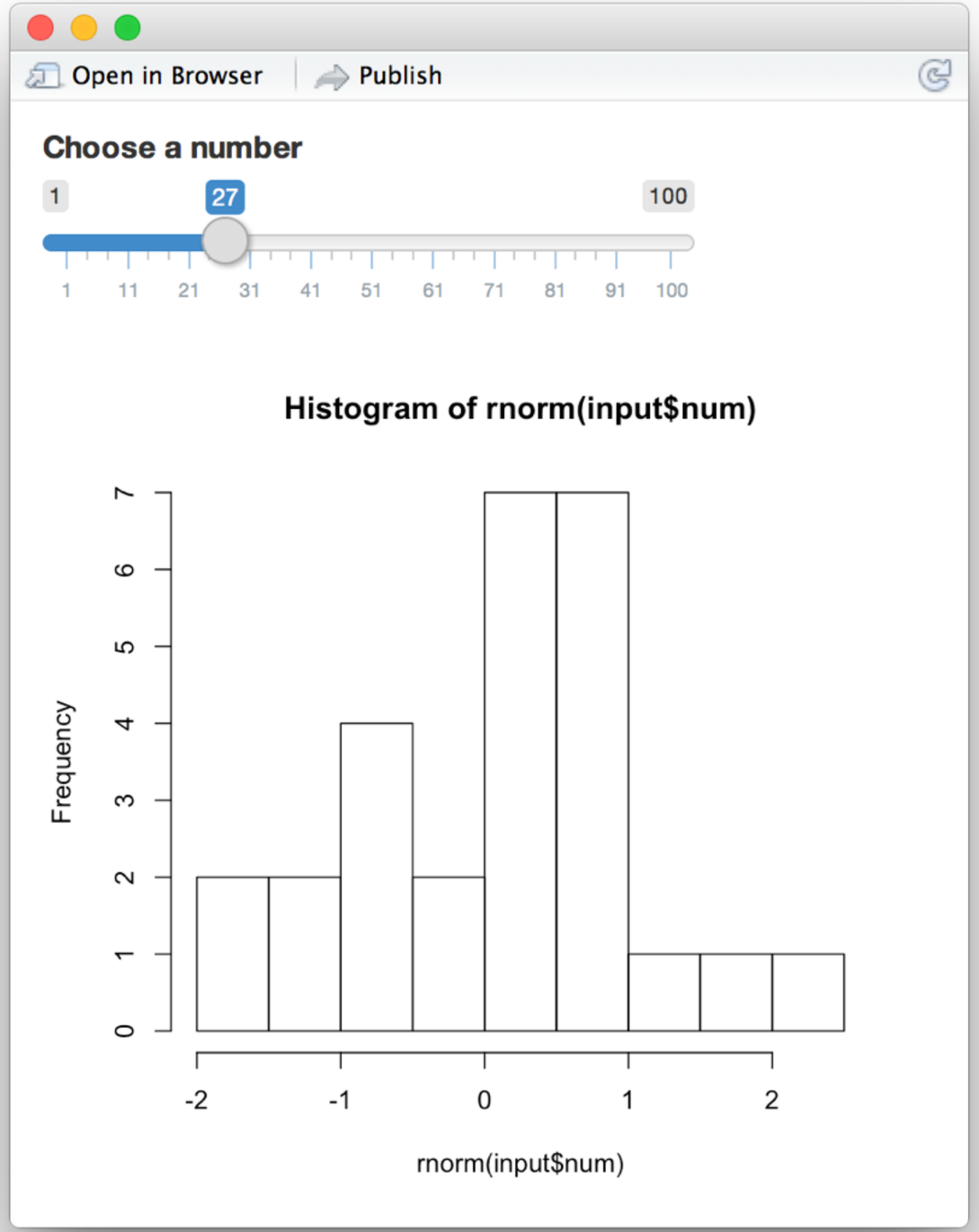
Server computer
running R and Shiny



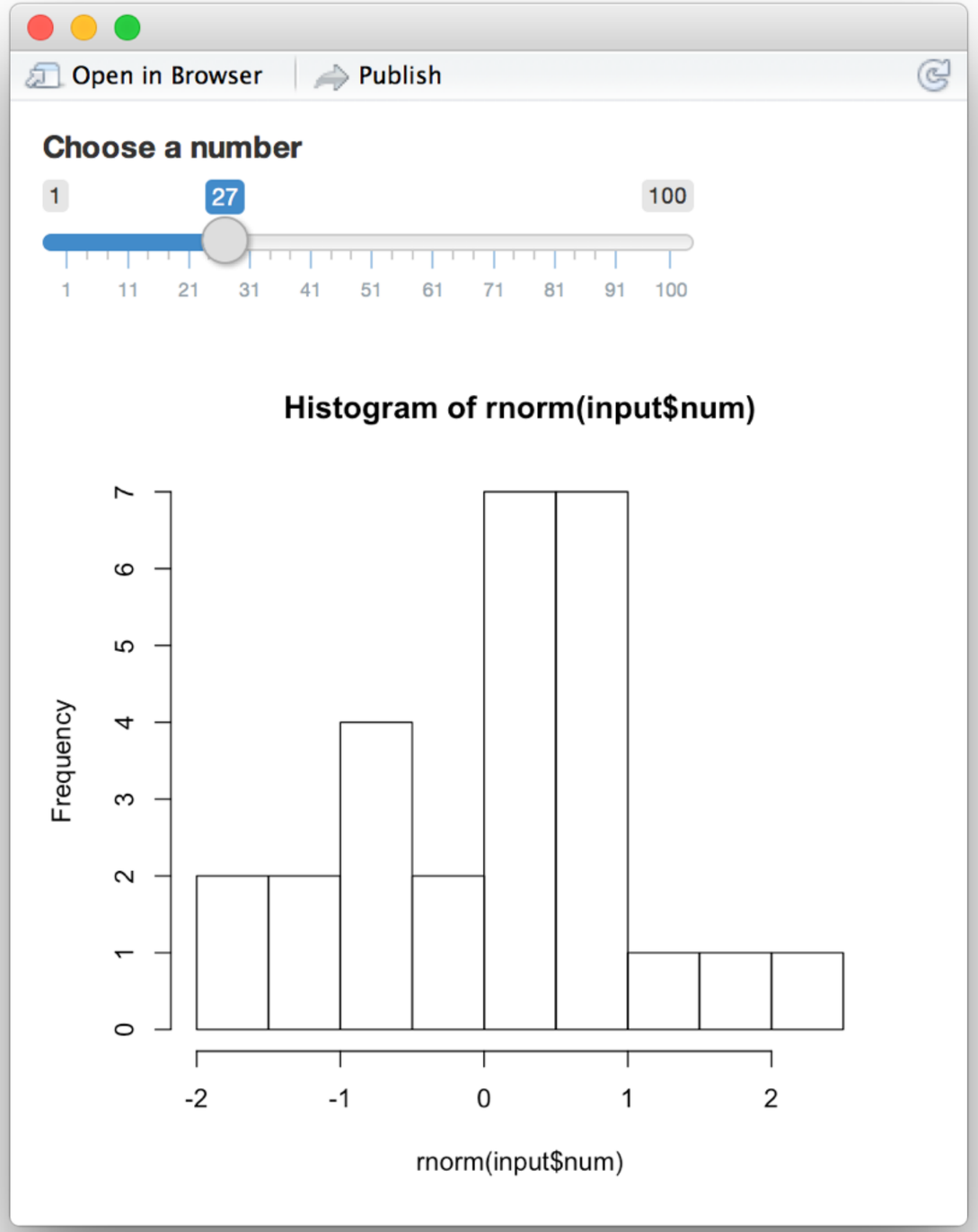


A Web browser

```
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
```



```
ui <-
<div class="container-fluid">
  <div class="form-group shiny-input-
container">
    <label class="control-label"
for="num">Choose a number</label>
    <input class="js-range-slider"
id="num" data-min="1" data-max="100"
data-from="25" data-step="1" data-
grid="true" data-grid-num="9.9" data-
grid-snap="false" data-prettify-
separator="," data-keyboard="true" data-
keyboard-step="1.01010101010101" data-
drag-interval="true" data-data-
type="number"/>
  </div>
  <div id="hist" class="shiny-plot-
output" style="width: 100% ; height:
400px"></div>
</div>
```



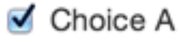
R Functions
to write HTML

Input functions

Buttons

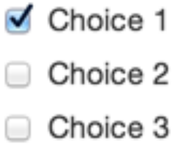
`actionButton()`
`submitButton()`

Single checkbox



`checkboxInput()`

Checkbox group



`checkboxGroupInput()`

Date input

`dateInput()`

Date range

`dateRangeInput()`

File input

`fileInput()`

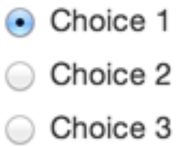
Numeric input

`numericInput()`

Password Input

`passwordInput()`

Radio buttons

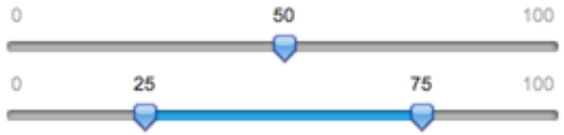


`radioButtons()`

Select box

`selectInput()`

Sliders



`sliderInput()`

Text input

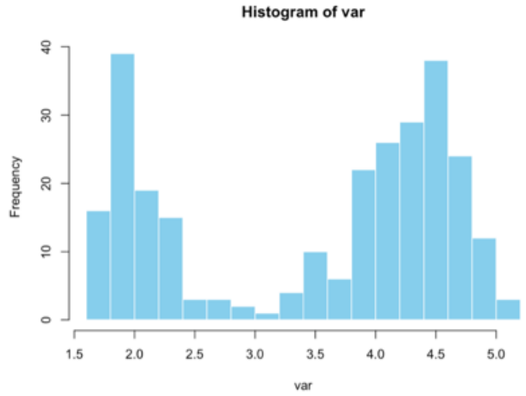
`textInput()`

Output functions

dataTableOutput()

htmlOutput()

imageOutput()



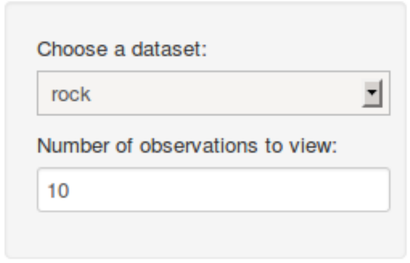
plotOutput()

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa

tableOutput()

foo

textOutput()



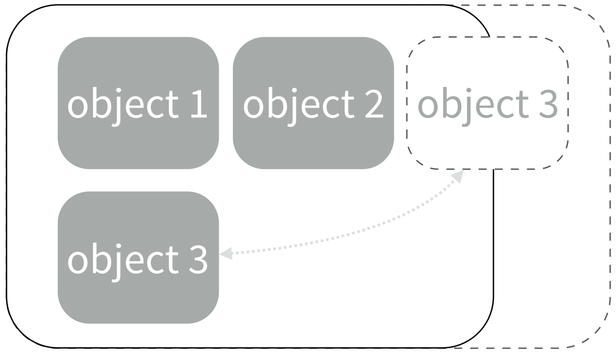
uiOutput()

```
      area             peri            shape              perm
 Min.   : 1016   Min.   : 308.6   Min.   :0.09033   Min.   :   6.30
 1st Qu.: 5305   1st Qu.:1414.9   1st Qu.:0.16226   1st Qu.:  76.45
 Median : 7487   Median :2536.2   Median :0.19886   Median : 130.50
 Mean   : 7188   Mean   :2682.2   Mean   :0.21811   Mean   : 415.45
 3rd Qu.: 8870   3rd Qu.:3989.5   3rd Qu.:0.26267   3rd Qu.: 777.50
 Max.   :12212   Max.   :4864.2   Max.   :0.46413   Max.   :1300.00
```

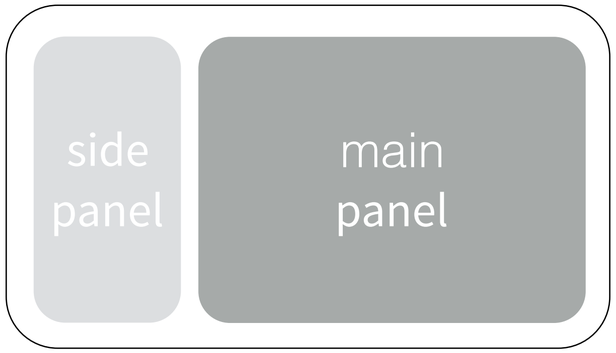
verbatimTextOutput()

Layout functions

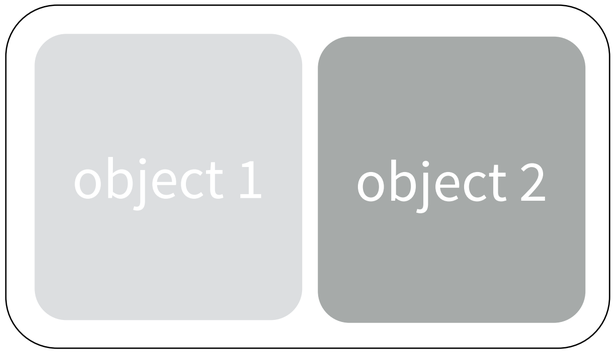
flowLayout()



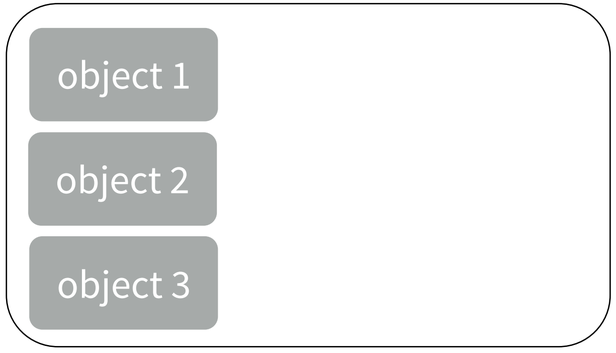
sidebarLayout()



splitLayout()

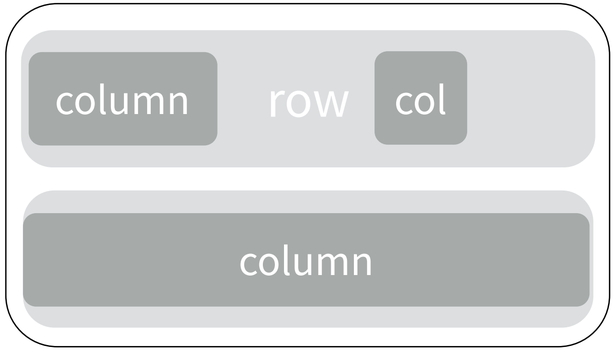


verticalLayout()



fluidRow()

column()



Panel functions

absolutePanel()

Panel position set rigidly (absolutely), not fluidly

conditionalPanel()

A JavaScript expression determines whether panel is visible or not.

fixedPanel()

Panel is fixed to browser window and does not scroll with the page

headerPanel()

Panel for the app's title, used with pageWithSidebar()

inputPanel()

Panel with grey background, suitable for grouping inputs

mainPanel()

Panel for displaying output, used with pageWithSidebar()

navlistPanel()

Panel for displaying multiple stacked tabPanels(). Uses sidebar navigation

sidebarPanel()

Panel for displaying a sidebar of inputs, used with pageWithSidebar()

tabPanel()

Stackable panel. Used with navlistPanel() and tabsetPanel()

tabsetPanel()

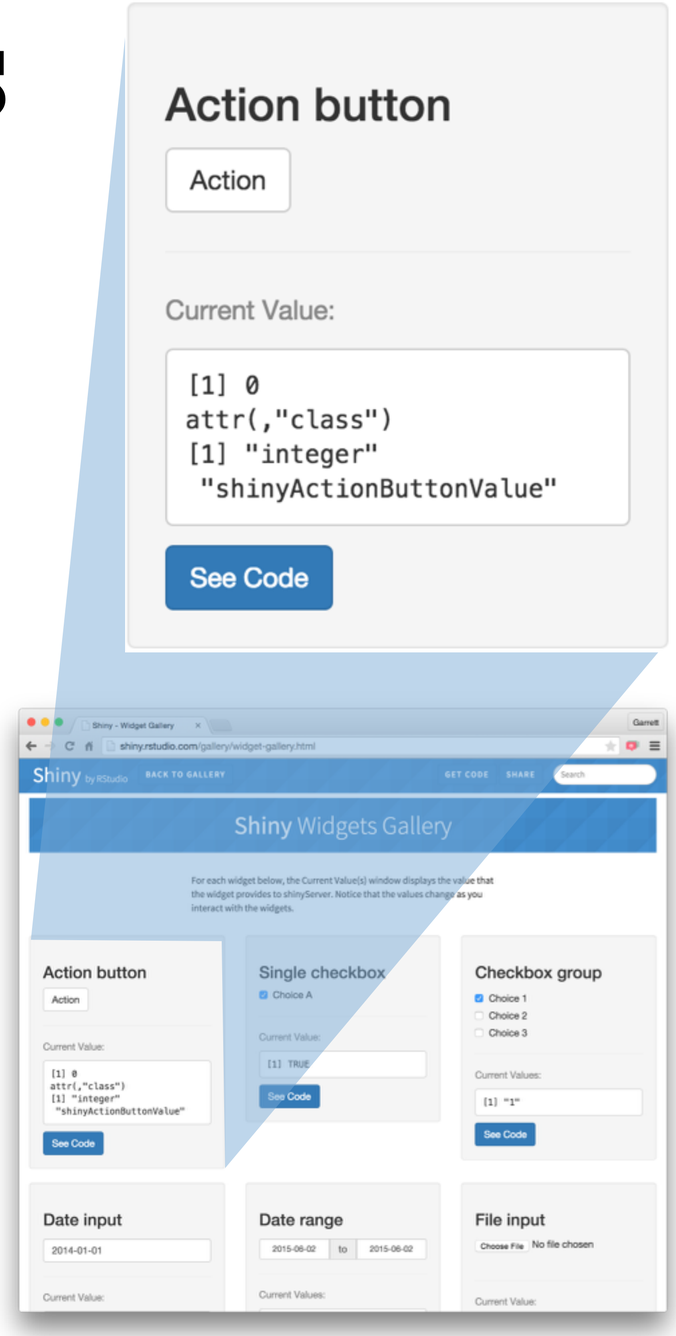
Panel for displaying multiple stacked tabPanels(). Uses tab navigation

titlePanel()

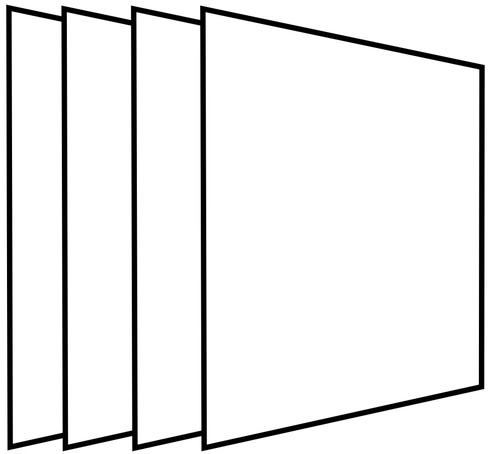
Panel for the app's title, used with pageWithSidebar()

wellPanel()

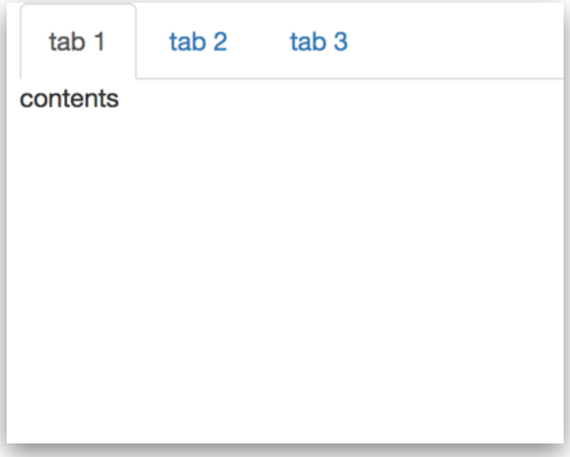
Panel with grey background.



"Layering" functions

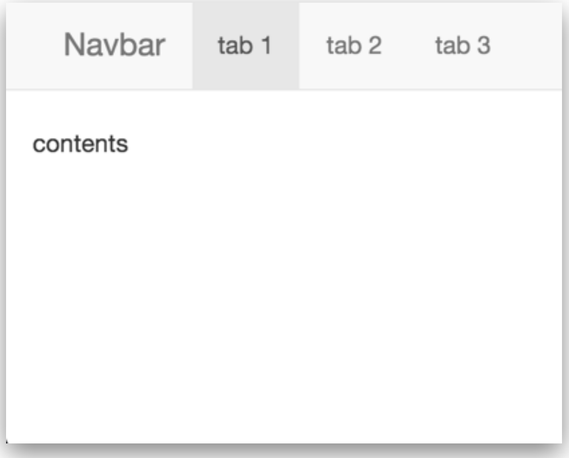


tabPanel()s



tabsetPanel()

navlistPanel()



navbarPage()

Formatting functions

```
tags$a()
tags$audio()
tags$body()
tags$code()
tags$dd()
tags$dt()
tags$figure()
tags$h4()
tags$hr()
tags$ins()
tags$link()
tags$nav()
tags$output()
tags$ruby()
tags$section()
tags$style()
tags$td()
tags$title()
tags$video()
```

```
tags$abbr()
tags$b()
tags$br()
tags$col()
tags$del()
tags$em()
tags$footer()
tags$h5()
tags$html()
tags$kbd()
tags$mark()
tags$noscript()
tags$p()
tags$rp()
tags$select()
tags$sub()
tags$textarea()
tags$tr()
tags$wbr()
```

```
tags$address()
tags$base()
tags$button()
tags$colgroup()
tags$details()
tags$embed()
tags$form()
tags$h6()
tags$i()
tags$keygen()
tags$map()
tags$object()
tags$param()
tags$rt()
tags$small()
tags$summary()
tags$tfoot()
tags$track()
html()
```

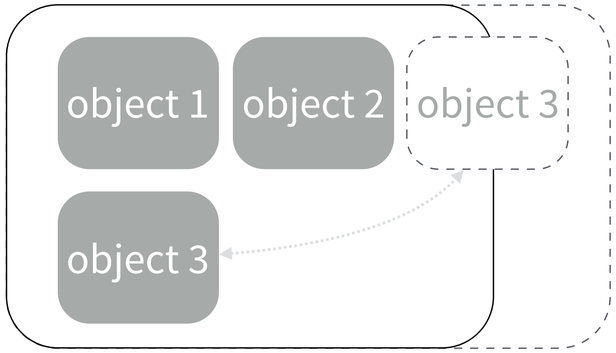
```
tags$area()
tags$bdi()
tags$canvas()
tags$command()
tags$dfn()
tags$eventsource()
tags$h1()
tags$head()
tags$iframe()
tags$label()
tags$menu()
tags$ol()
tags$pre()
tags$s()
tags$source()
tags$sup()
tags$th()
tags$u()
```

```
tags$article()
tags$bdo()
tags$caption()
tags$data()
tags$div()
tags$fieldset()
tags$h2()
tags$header()
tags$img()
tags$legend()
tags$meta()
tags$optgroup()
tags$progress()
tags$samp()
tags$span()
tags$table()
tags$thead()
tags$ul()
```

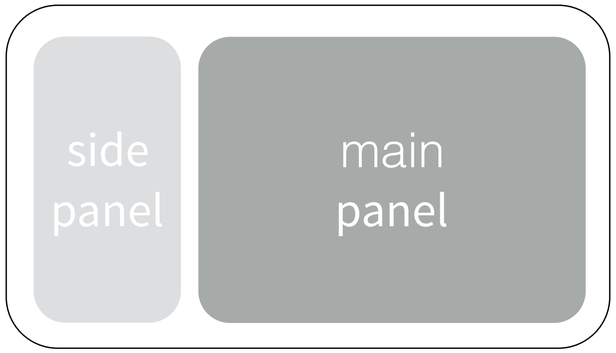
```
tags$aside()
tags$blockquote()
tags$cite()
tags$datalist()
tags$dl()
tags$figcaption()
tags$h3()
tags$hgroup()
tags$input()
tags$li()
tags$meter()
tags$option()
tags$q()
tags$script()
tags$strong()
tags$tbody()
tags$time()
tags$var()
```

Layout functions

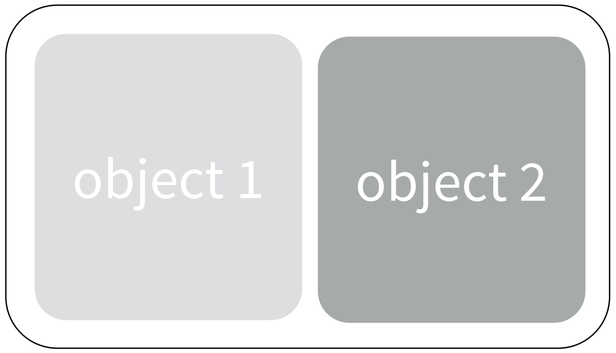
flowLayout()



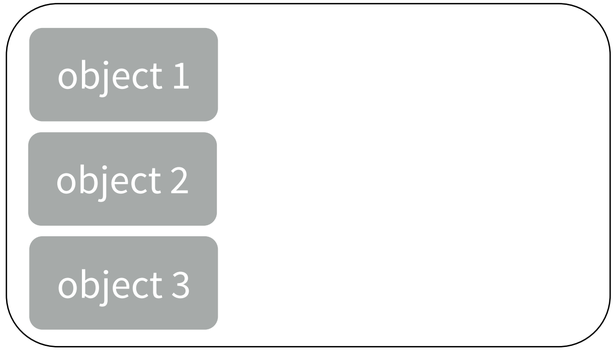
sidebarLayout()



splitLayout()

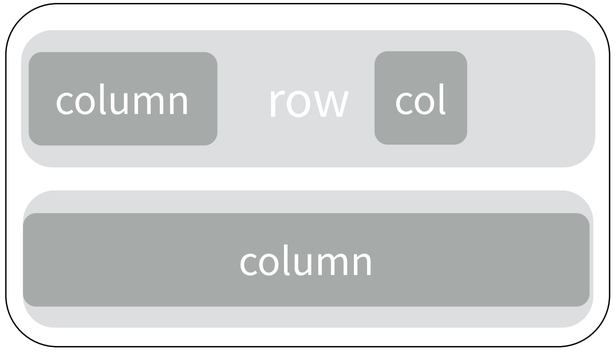


verticalLayout()



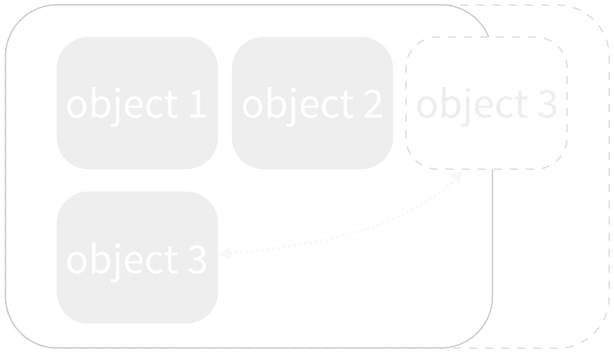
fluidRow()

column()



Layout functions

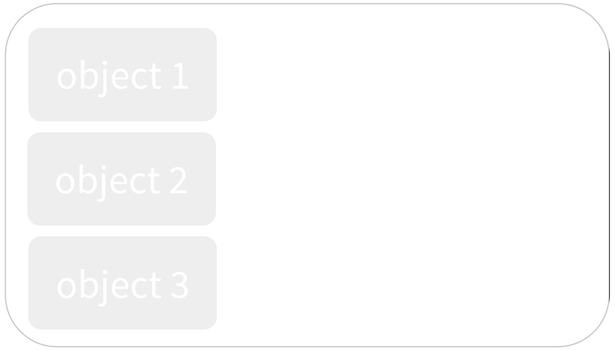
flowLayout()



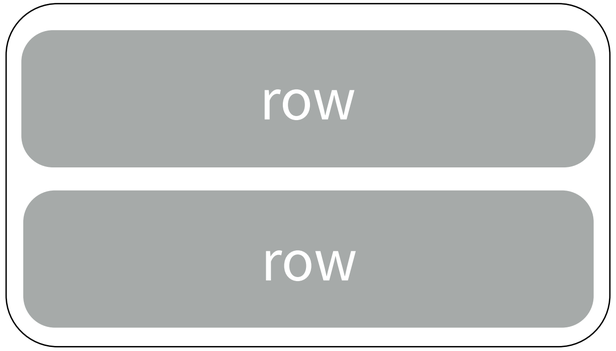
sidebarLayout()

splitLayout()

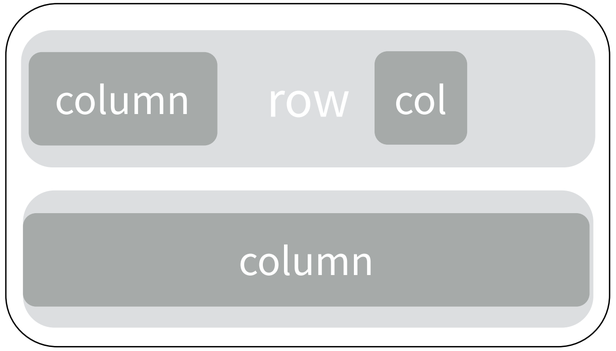
verticalLayout()



fluidRow()



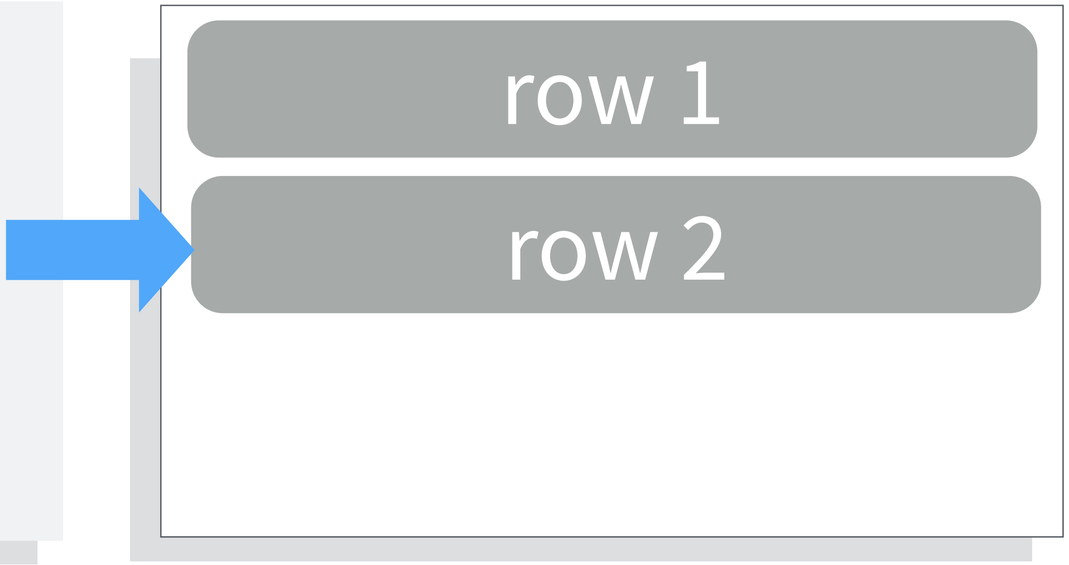
column()



fluidRow()

`fluidRow()` adds rows to the grid. Each new row goes below the previous rows.

```
fluidPage(
  fluidRow(),
  fluidRow()
)
```

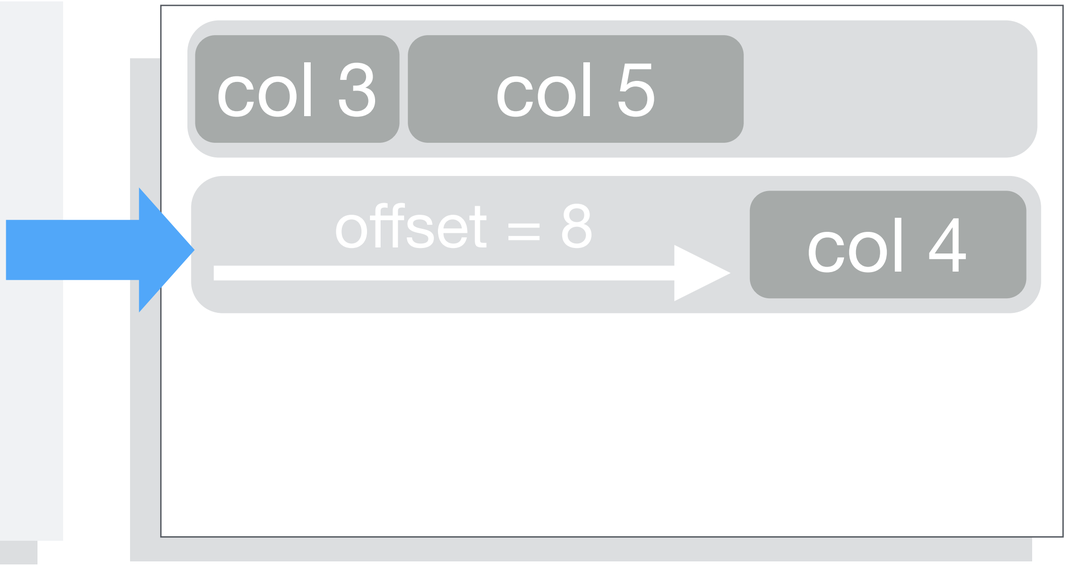


column()

`column()` adds columns within a row. Each new column goes to the left of the previous column.

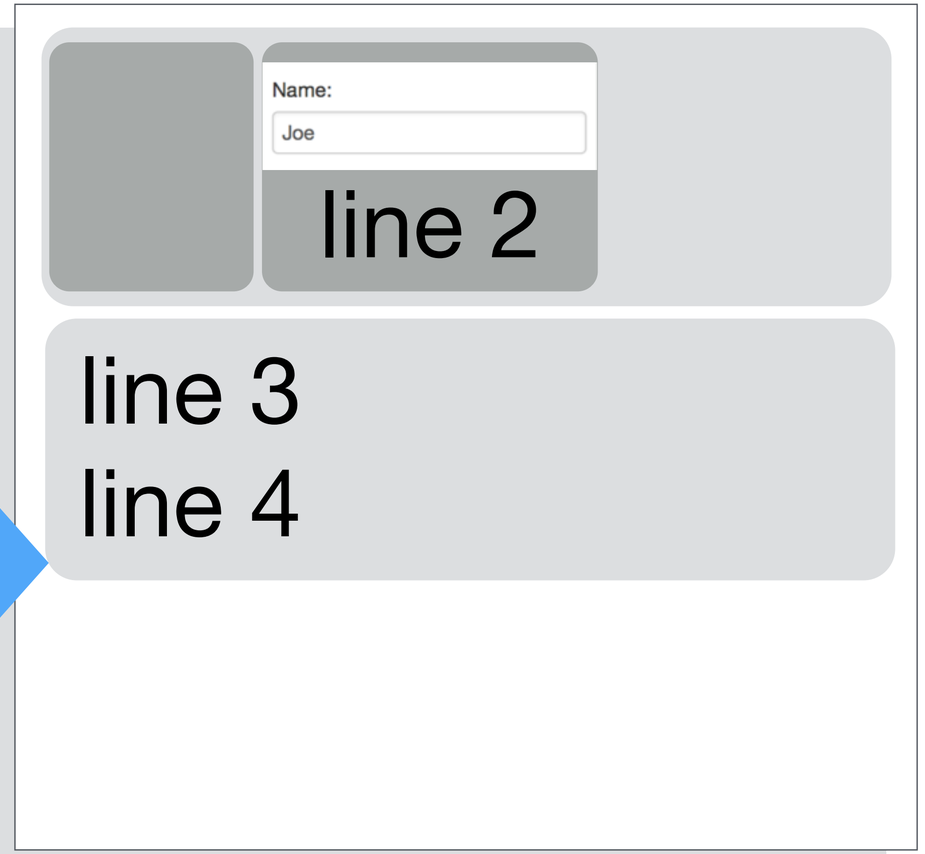
Specify the **width** and **offset** of each column out of 12

```
fluidPage(
  fluidRow(
    column(3),
    column(5)),
  fluidRow(
    column(4, offset = 8)
)
```



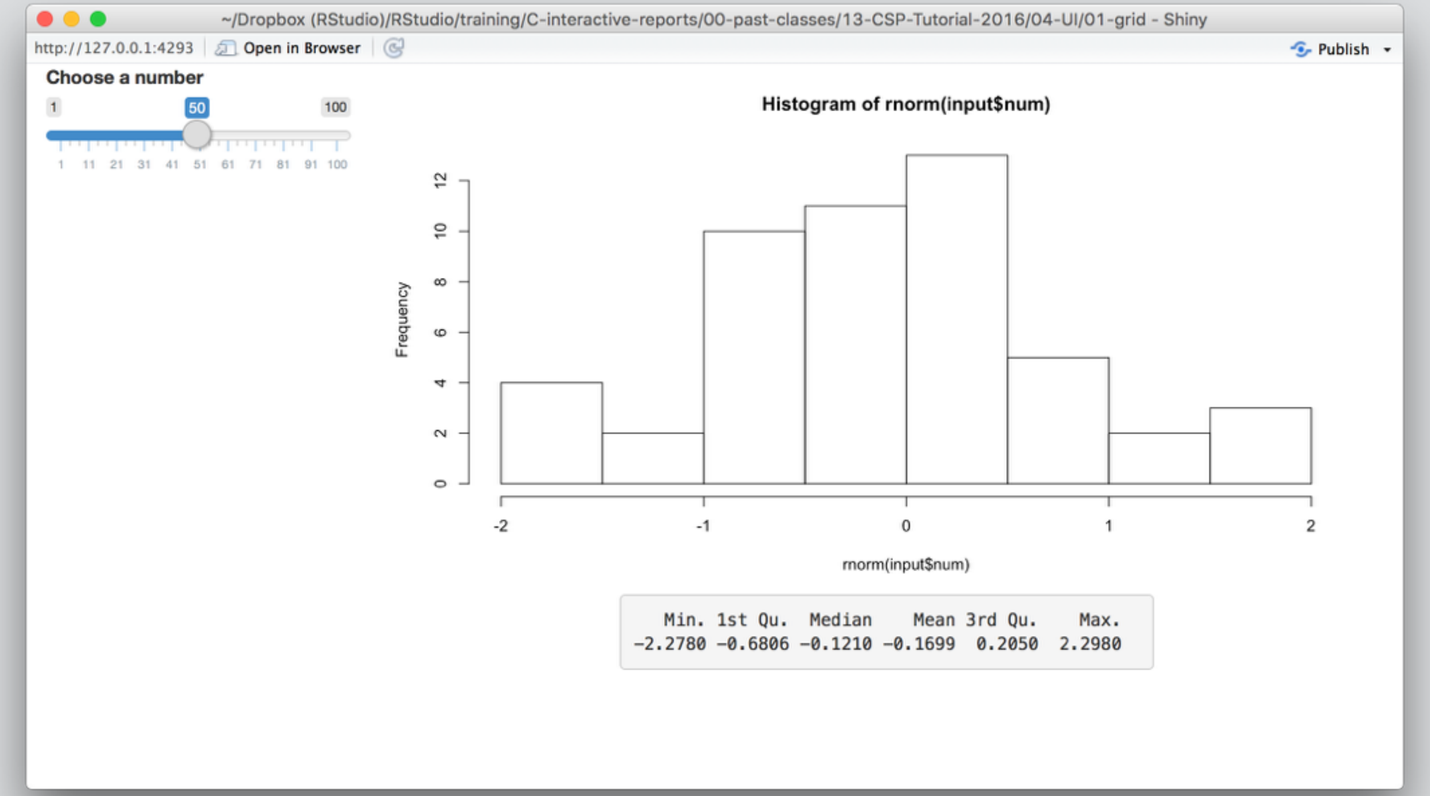
Shiny will fill columns and rows from top to bottom. Heights will expand to content.

```
fluidPage(
  fluidRow(
   column(3),
   column(5,
     textInput("n","Name:",value = "Joe"),
     "line 2"
   )
  ),
  fluidRow(
    "line3",
    "line4")
)
```



Your Turn

Open your `demos/06-basic-app.R`. Then use `fluidRow()` and `column()` to arrange your app like this.



Note: you will need to view the app in its own window.

Panels

Panel functions

absolutePanel()

Panel position set rigidly (absolutely), not fluidly

conditionalPanel()

A JavaScript expression determines whether panel is visible or not.

fixedPanel()

Panel is fixed to browser window and does not scroll with the page

headerPanel()

Panel for the app's title, used with pageWithSidebar()

inputPanel()

Panel with grey background, suitable for grouping inputs

mainPanel()

Panel for displaying output, used with pageWithSidebar()

navlistPanel()

Panel for displaying multiple stacked tabPanels(). Uses sidebar navigation

sidebarPanel()

Panel for displaying a sidebar of inputs, used with pageWithSidebar()

tabPanel()

Stackable panel. Used with navlistPanel() and tabsetPanel()

tabsetPanel()

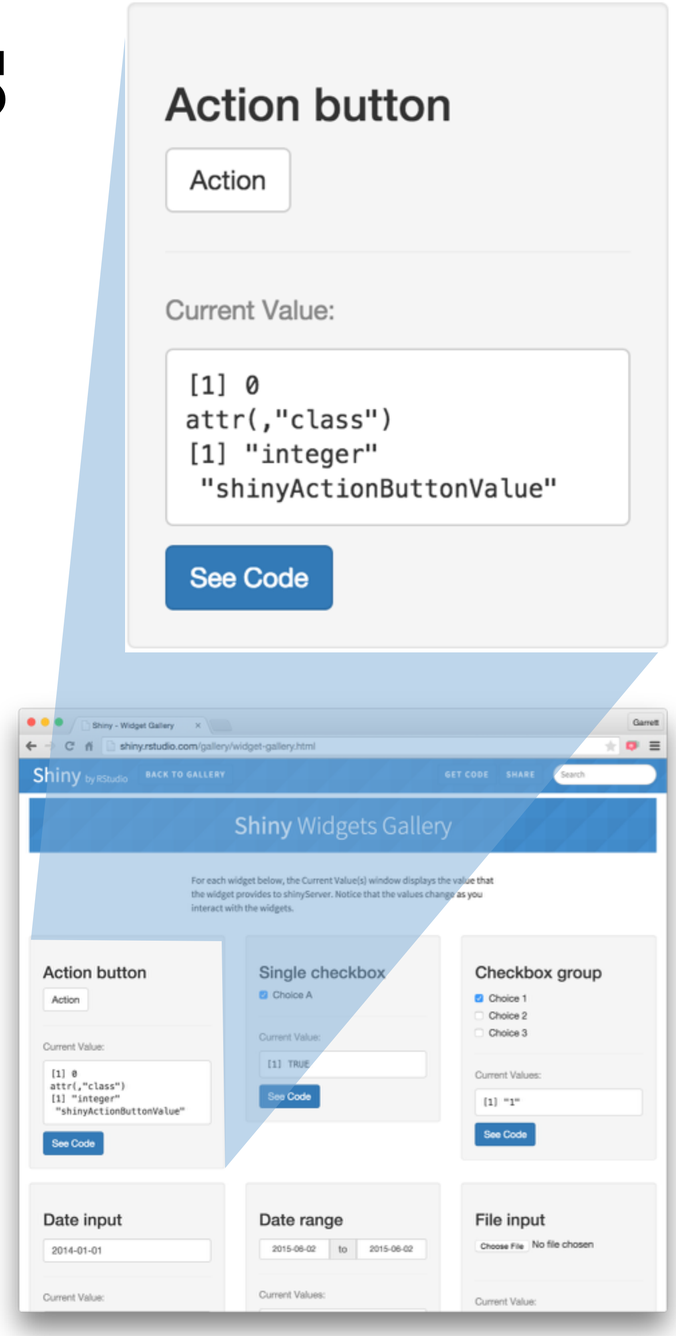
Panel for displaying multiple stacked tabPanels(). Uses tab navigation

titlePanel()

Panel for the app's title, used with pageWithSidebar()

wellPanel()

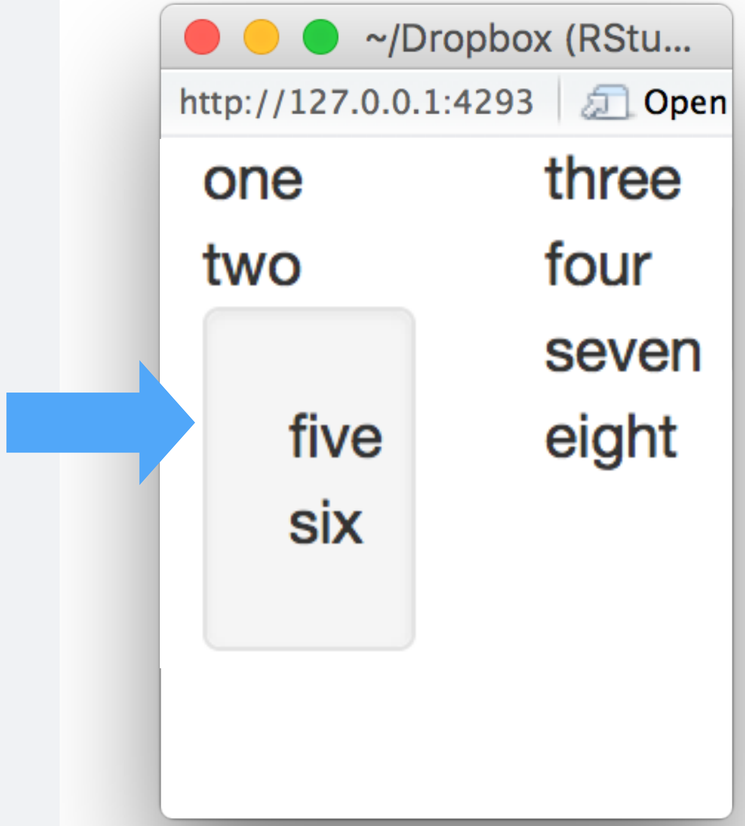
Panel with grey background.



wellPanel

Visually groups arguments into a grey field.

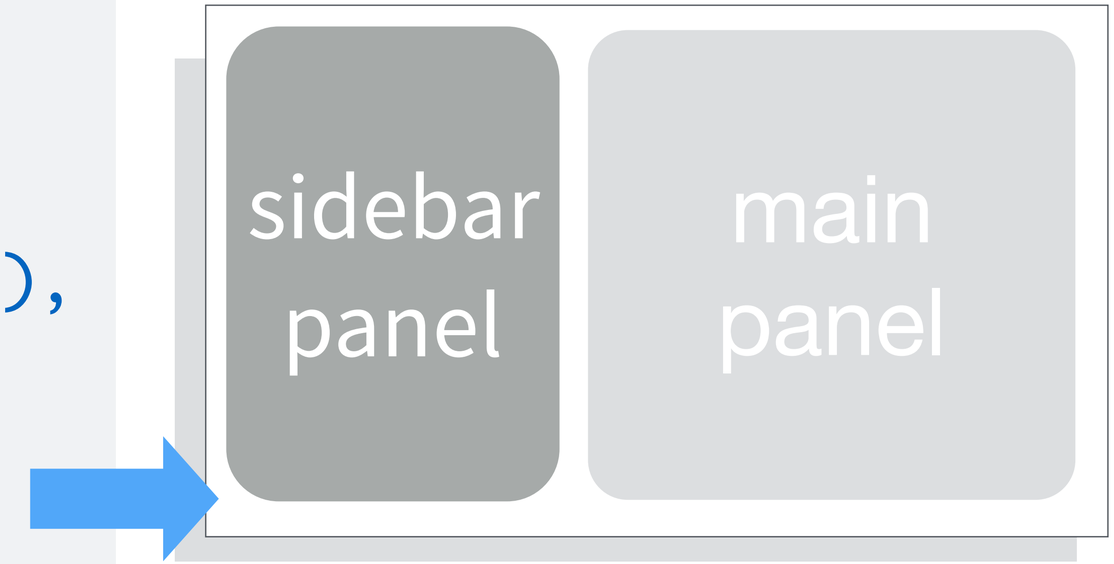
```
fluidPage(
  fluidRow(
    column(1, "one", "two"),
    column(1, "three", "four")
  ),
  fluidRow(
    column(1,
      wellPanel("five", "six")
    ),
    column(1, "seven", "eight")
  )
)
```

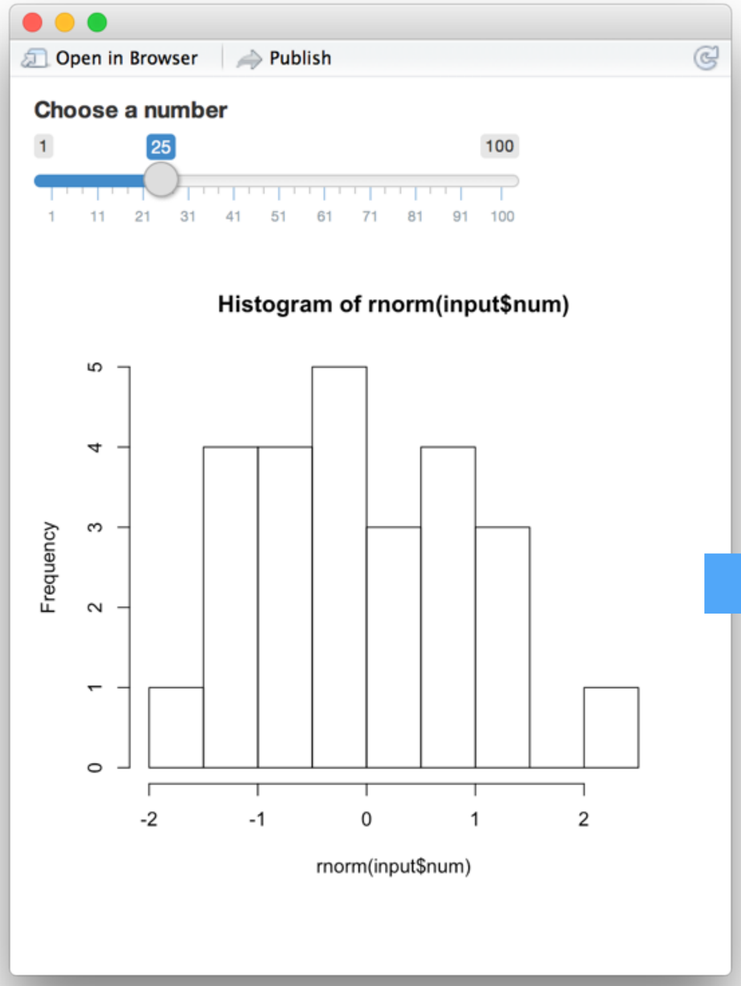


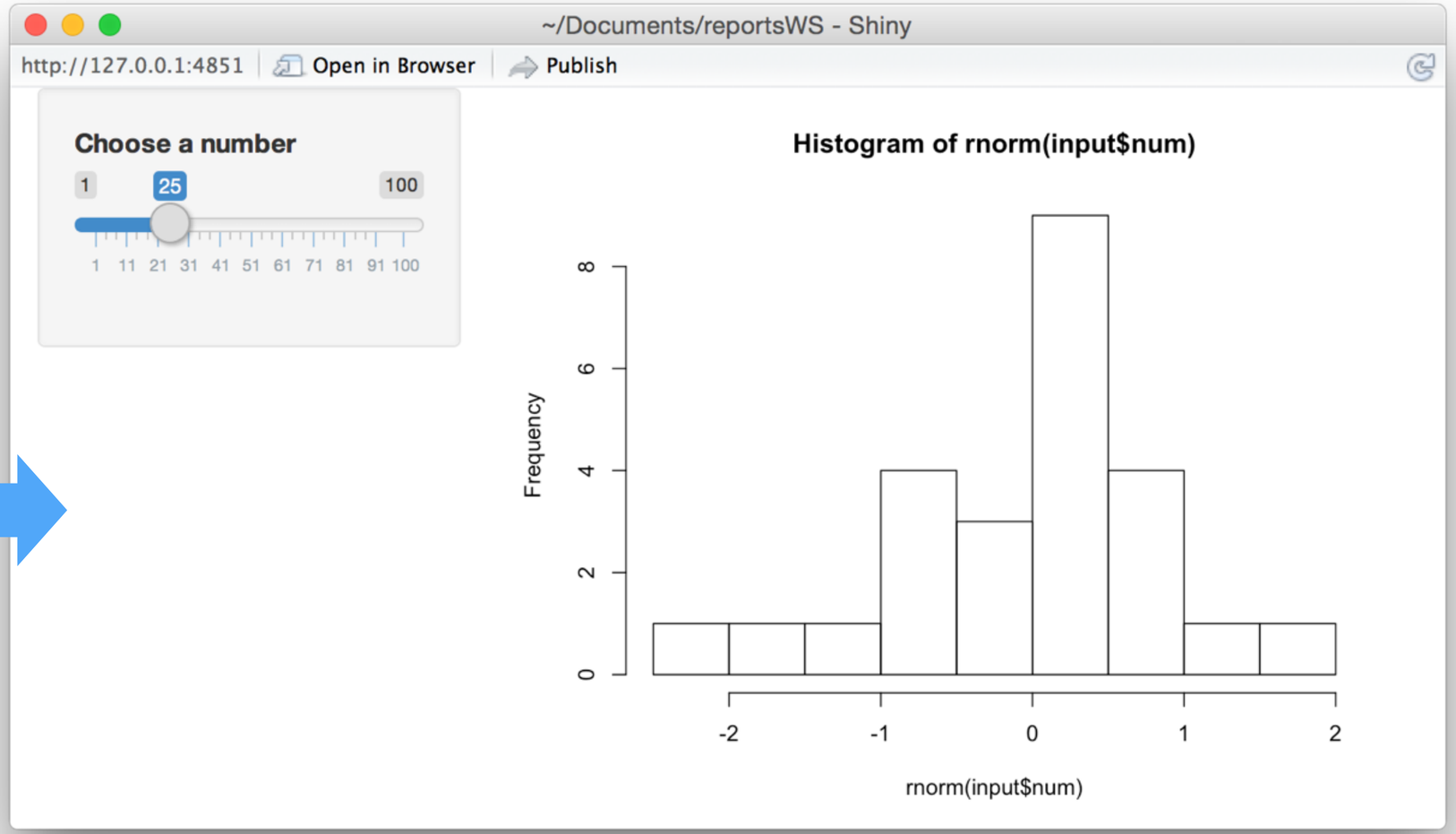
sidebarLayout()

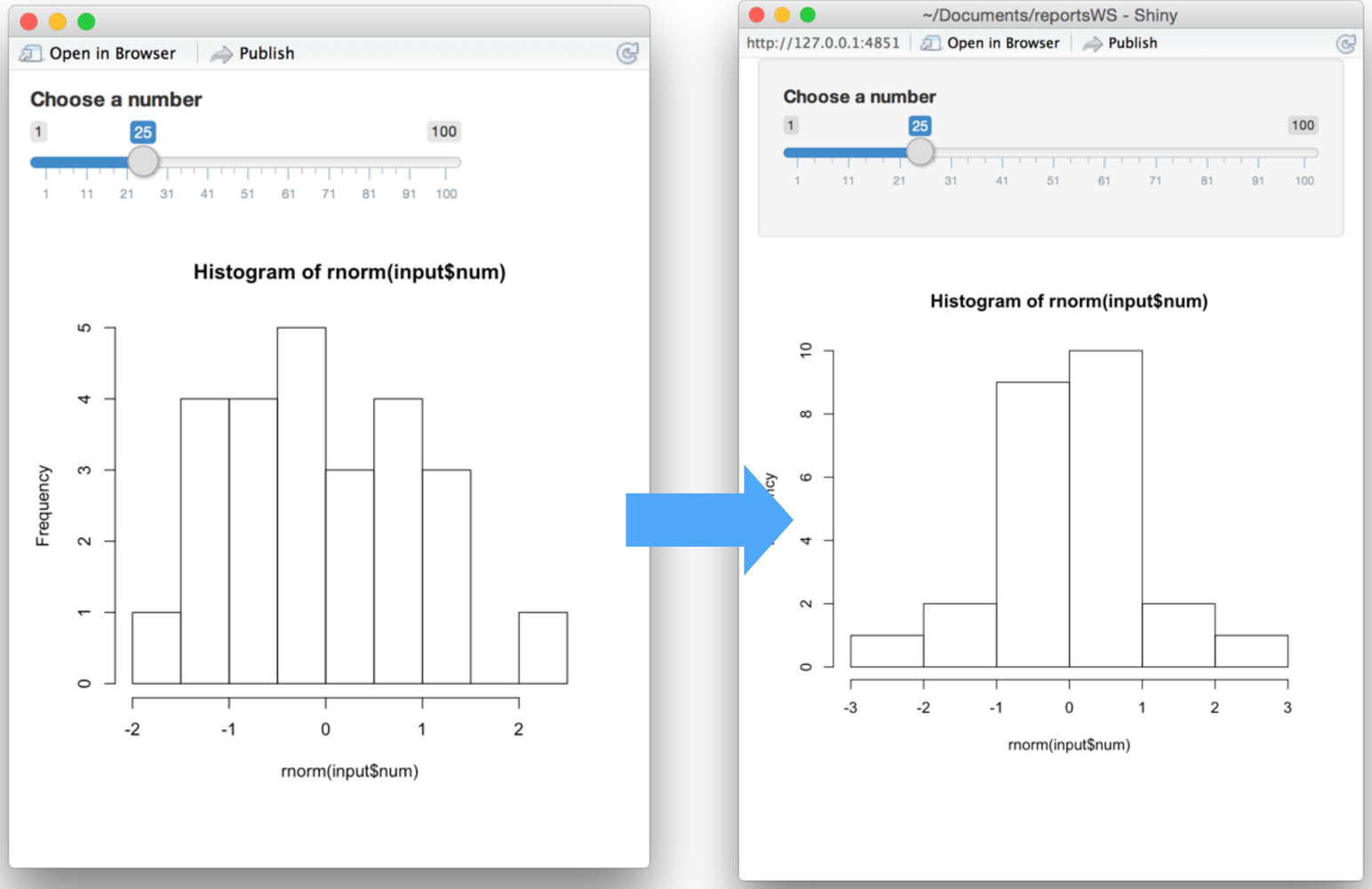
The most simple way to layout a Shiny app. Use with sidebarPanel() and mainPanel()

```
fluidPage(
 sidebarLayout(
  sidebarPanel(),
    mainPanel()
  )
)
```









Your Turn

```
ui <- fluidPage(
  fluidRow(
    column(3,
      sliderInput("num", "Choose a number", 1, 100, 50)
    ),
    column(9,
      plotOutput("hist")
    )
  ),
  fluidRow(
    column(5, offset = 5,
      verbatimTextOutput("sum")
    )
  )
)
server <- function(input, output) {
  data <- reactive({rnorm(input$num)})
  output$hist <- renderPlot({
    hist(data())
  })
  output$sum <- renderPrint({
    summary(data())
  })
}
shinyApp(ui = ui, server = server)
```

Replace fluidRow()'s and column()'s with

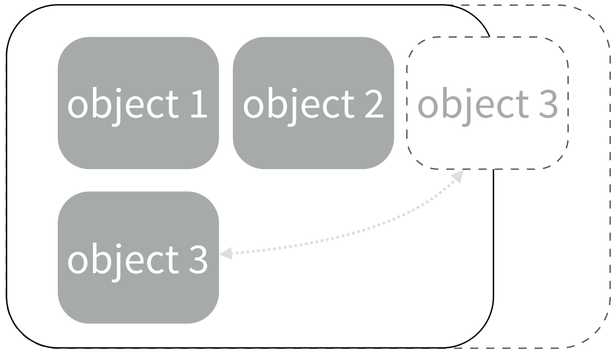
```
sidebarLayout(
  sidebarPanel(),
  mainPanel()
)
```

to give your last app a sidebar layout. **Save this app.**

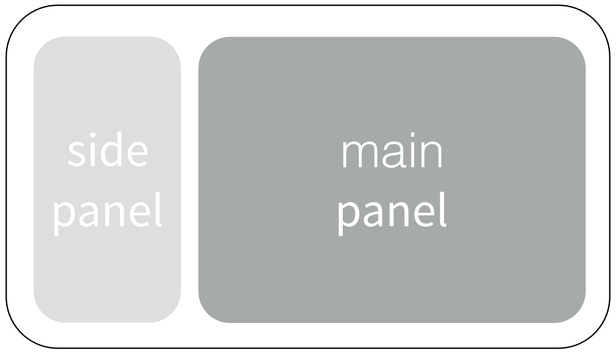
Layers

Layout functions

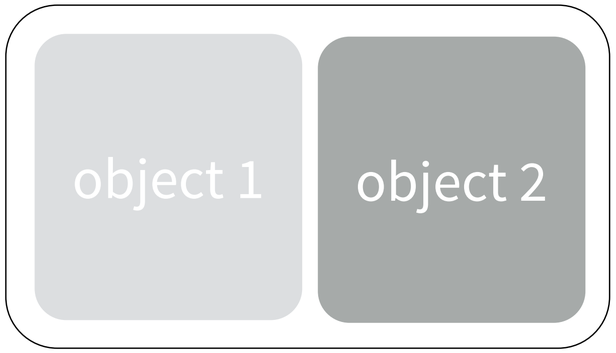
flowLayout()



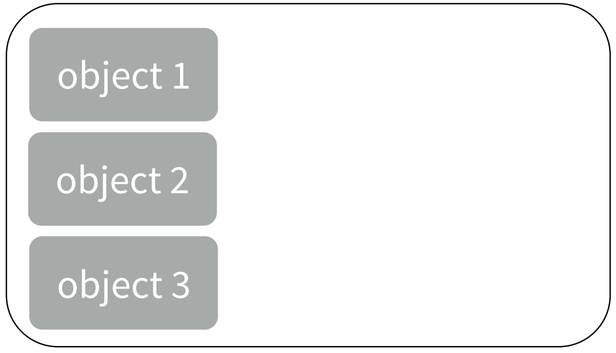
sidebarLayout()



splitLayout()

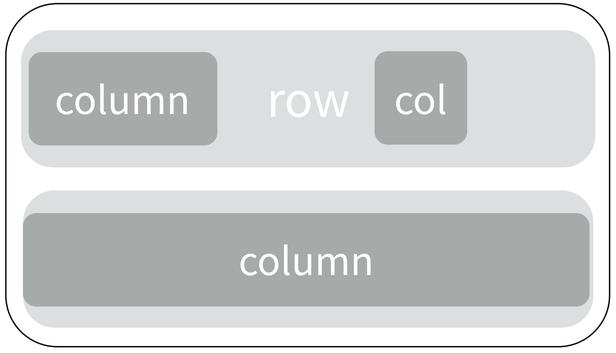


verticalLayout()

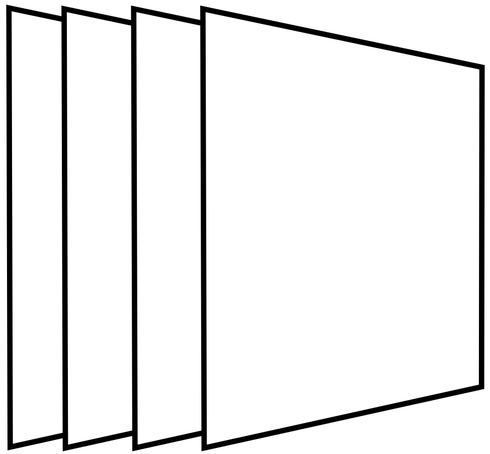


fluidRow()

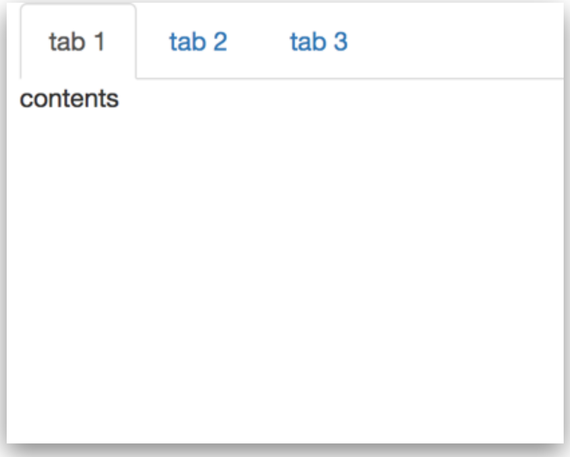
column()



"Layering" functions

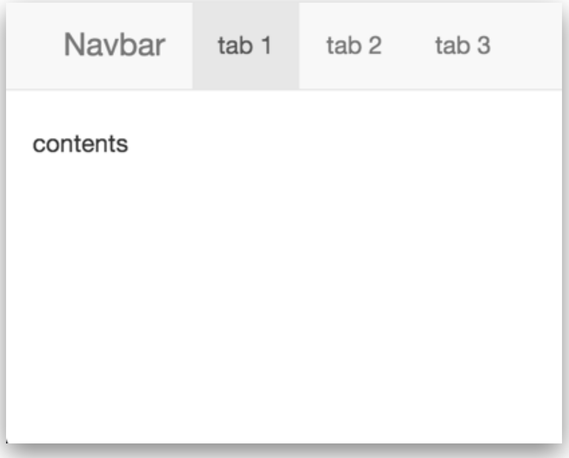


tabPanel()s



tabsetPanel()

navlistPanel()



navbarPage()

tabPanel()

`tabPanel()` creates a tab that hides elements until a user navigates to them. Each tab is like a small UI of its own.

```
tabPanel("Tab 1", …)
```

A title
(for navigation)

elements to
appear in the tab

Use tabPanel()'s with one of:

- tabsetPanel()
- navlistPanel()
- navbarPage()

tabsetPanel()

`tabsetPanel()` combines tabs into a single *panel*.

Use *tabs* to navigate between tabs.

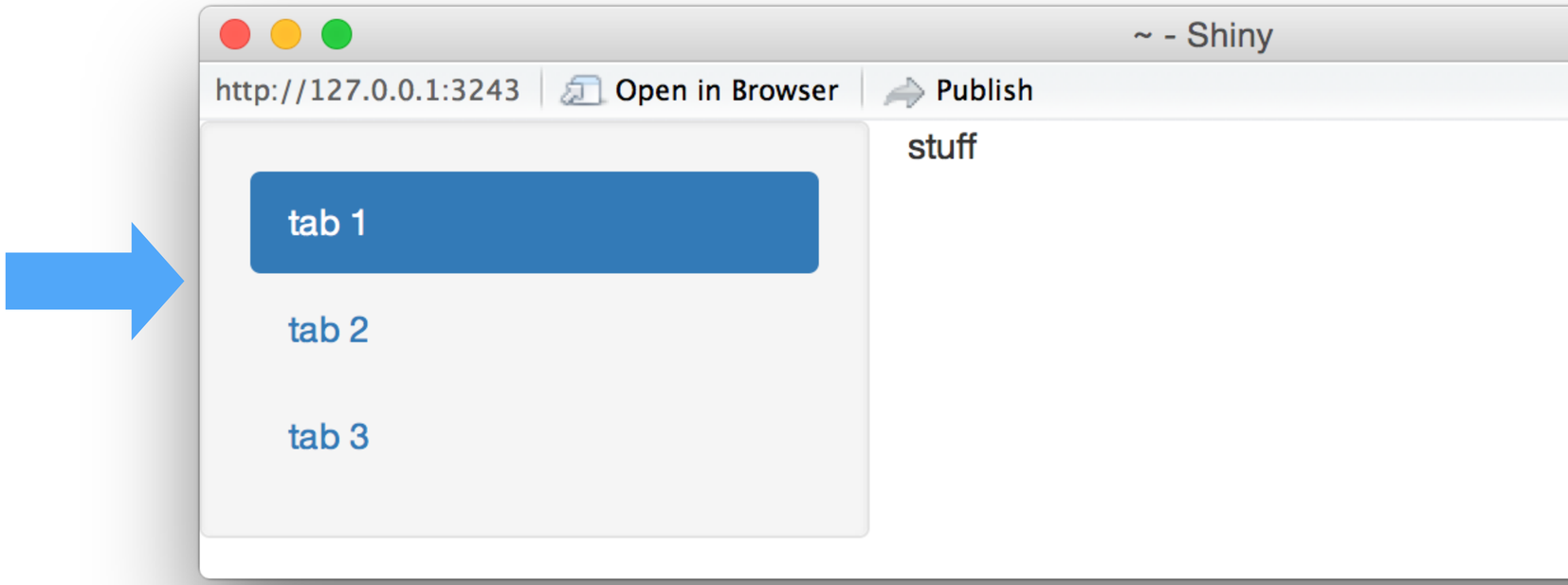
```
fluidPage(
  tabsetPanel(
   tabPanel("tab 1", "stuff"),
   tabPanel("tab 2", "stuff"),
   tabPanel("tab 3", "stuff")
  )
)
```

navlistPanel()

`navlistPanel()` combines tabs into a single *panel*.

Use *links* to navigate between tabs.

```
fluidPage(
  navlistPanel(
   tabPanel("tab 1", "stuff"),
   tabPanel("tab 2", "stuff"),
   tabPanel("tab 3", "stuff")
  )
)
```

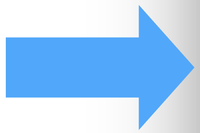


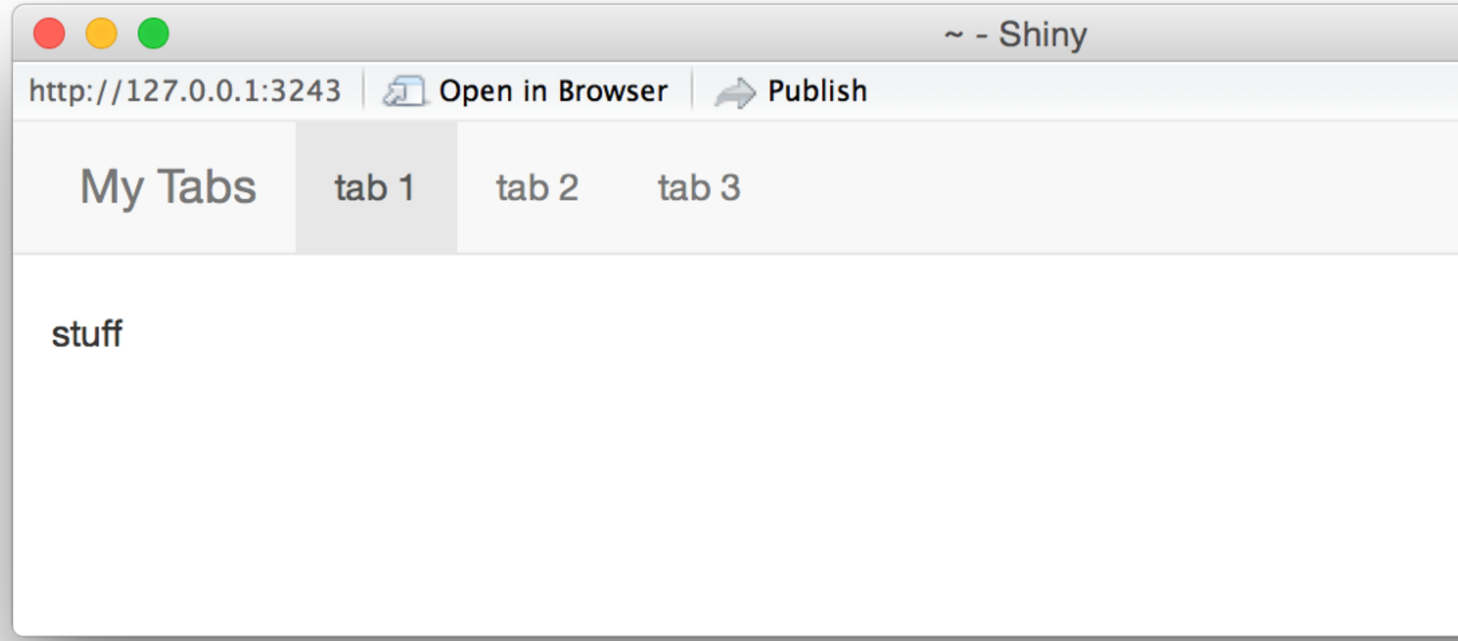
navbarPage()

`navbarPage()` combines tabs into a single *page*.

navbarPage() replaces fluidPage().

```
navbarPage("My Tabs",
    tabPanel("tab 1", "stuff"),
    tabPanel("tab 2", "stuff"),
    tabPanel("tab 3", "stuff")
)
```

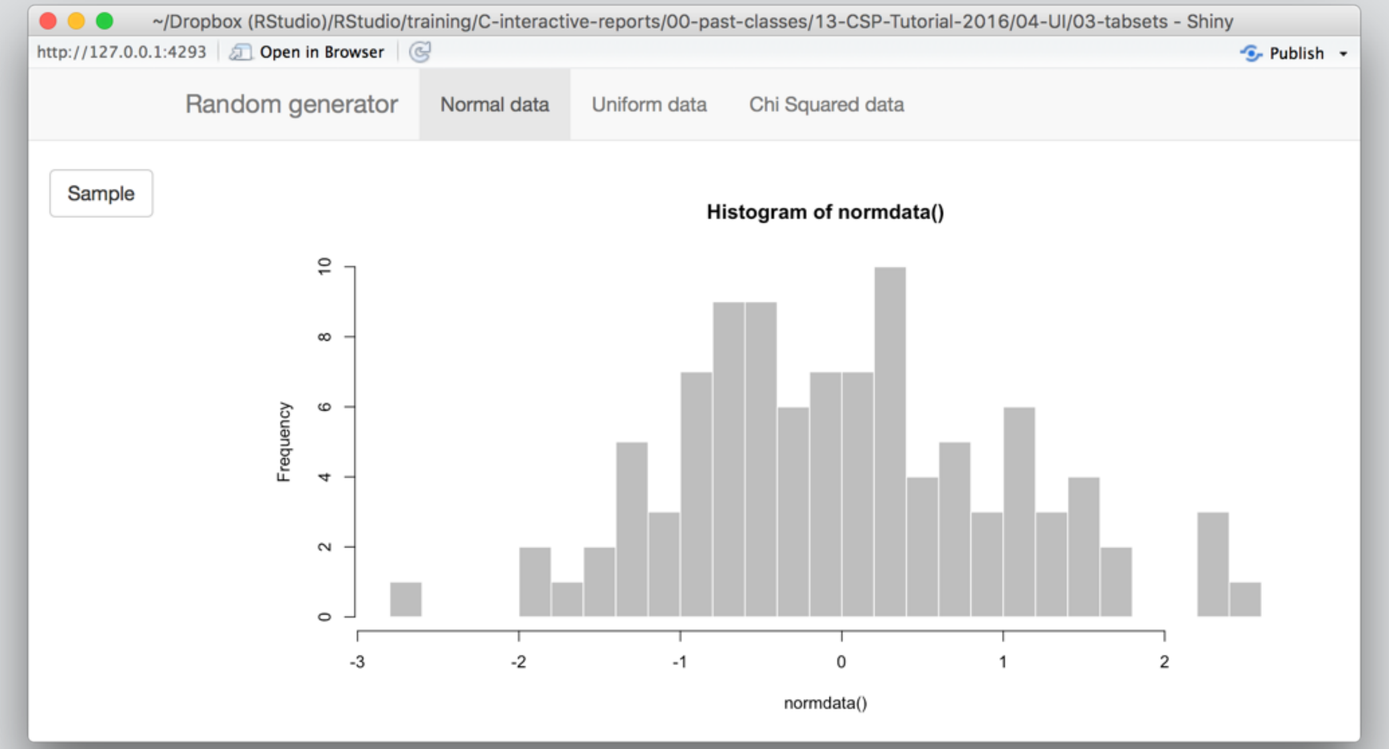




Your Turn

Reorganize the app in `demos/09-layers.R`

into three tab panels connected by a navbar, like this.



formatting

tags

Each element of the tags list is a function that recreates an html tag.

```
names(tags)
##  [1] "a"          "abbr"       "address"    "area"
##  [5] "article"    "aside"      "audio"      "b"
##  [9] "base"       "bdi"        "bdo"        "blockquote"
## [13] "body"       "br"         "button"     "canvas"
## [17] "caption"    "cite"       "code"       "col"
## [21] "colgroup"   "command"    "data"       "datalist"
## [25] "dd"         "del"        "details"    "dfn"
## [29] "div"        "dl"         "dt"         "em"
## [33] "embed"      "eventsource" "fieldset"  "figcaption"
## [37] "figure"     "footer"     "form"       "h1"
## [41] "h2"         "h3"         "h4"         "h5"
```

Shiny HTML tag functions

Shiny provides R functions to recreate HTML tags.

`tags$h1()` ⟷ `<h1></h1>`

`tags$a()` ⟷ `<a></a>`

tags syntax

```
tags$a(href = "www.rstudio.com", "RStudio")
```

the list named tags

the function/tag name (followed by parentheses)

named arguments appear as tag attributes (set boolean attributes to NA)

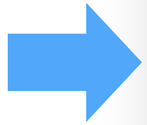
unnamed arguments appear inside the tags (call tags$…() to create nested tags)

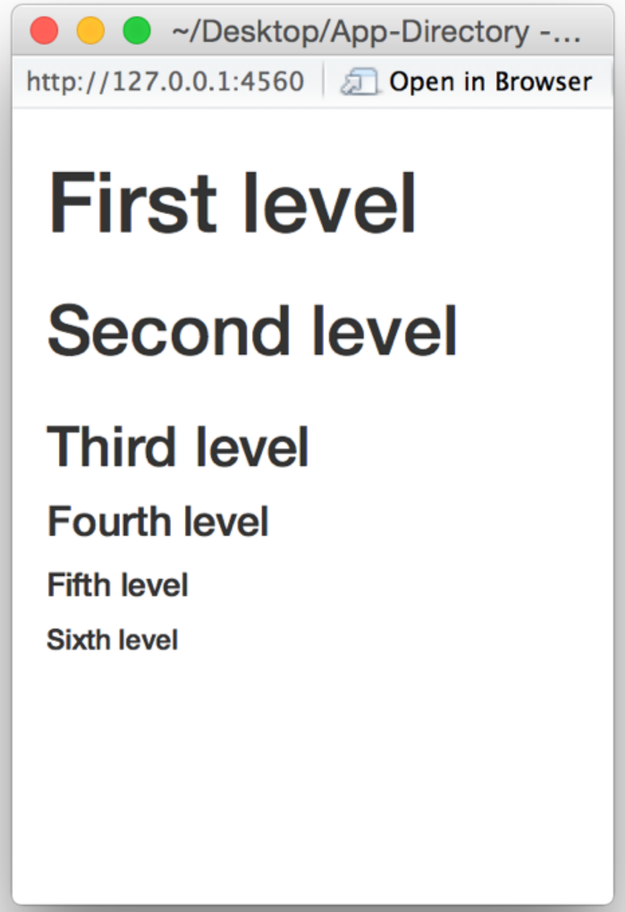
```
<a href="www.rstudio.com">RStudio</a>
```

h1() - h6()

Headers

```
fluidPage(
  tags$h1("First level"),
  tags$h2("Second level"),
  tags$h3("Third level"),
  tags$h4("Fourth level"),
  tags$h5("Fifth level"),
  tags$h6("Sixth level")
)
```

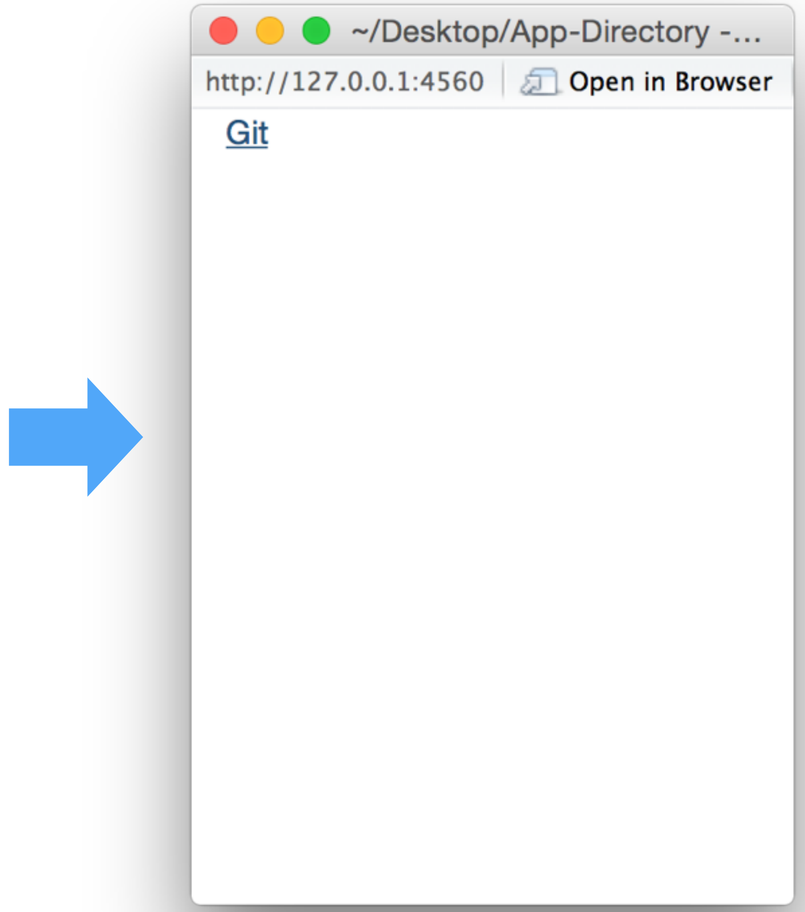




a()

hyperlink with the href argument

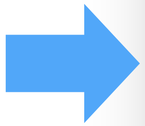
```
fluidPage(
  tags$a(href= "http://www.git.com",
    "Git")
)
```

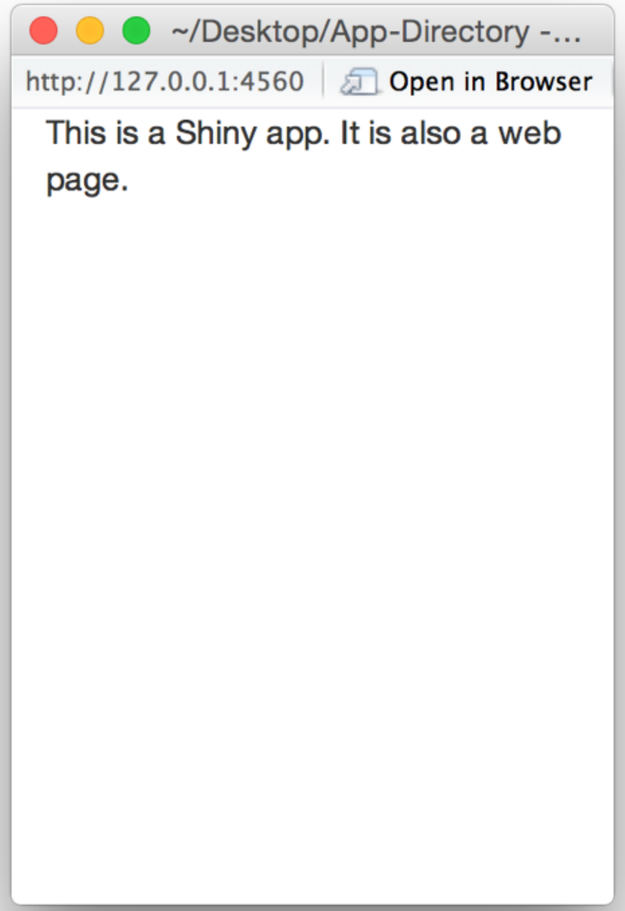


text

Character strings do not need a tag.

```
fluidPage(
 "This is a Shiny app.",
 "It is also a web page."
)
```

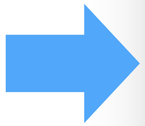


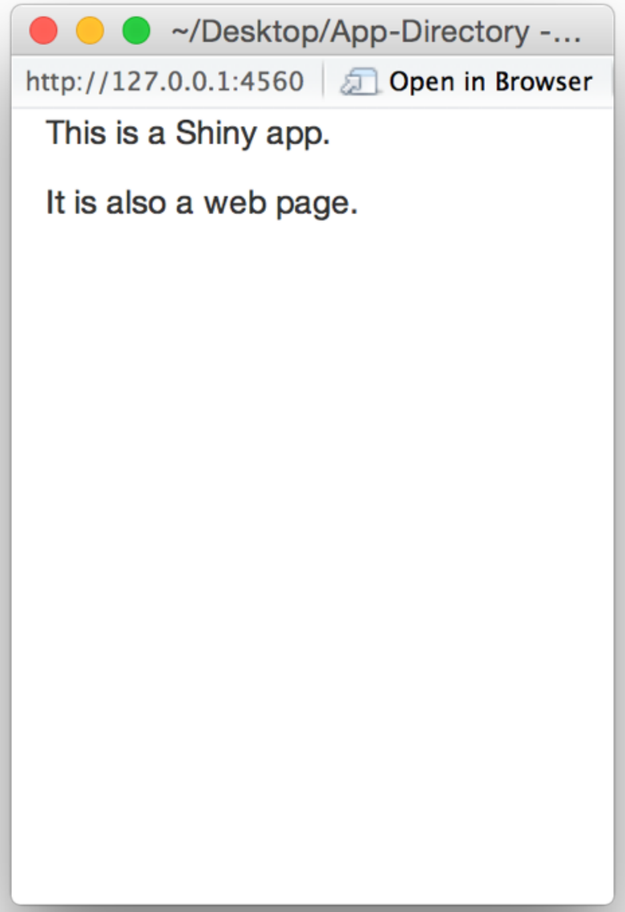


p()

A new paragraph

```
fluidPage(
 tags$p("This is a Shiny app."),
 tags$p("It is also a web page.")
)
```

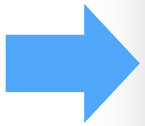


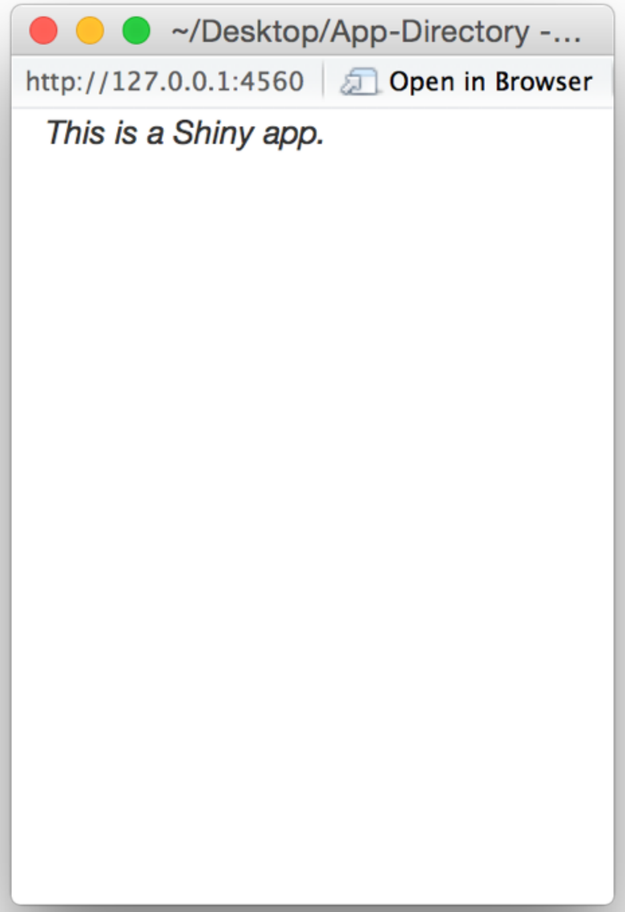


em()

Emphasized (*italic*) text

```
fluidPage(
  tags$em("This is a Shiny app.")
)
```

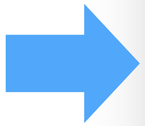


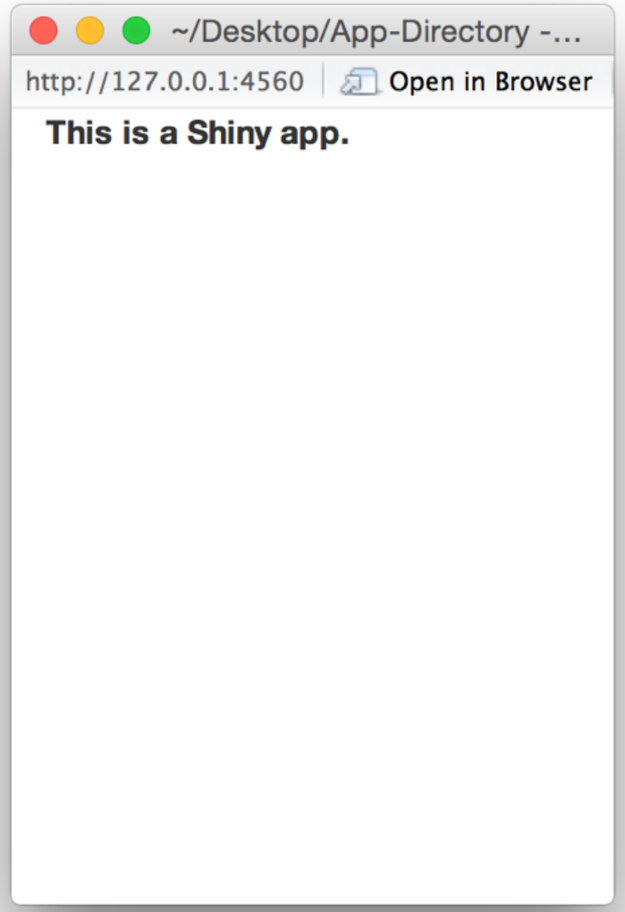


strong()

Strong (**bold**) text

```
fluidPage(
  tags$strong("This is a Shiny app.")
)
```

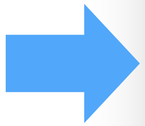


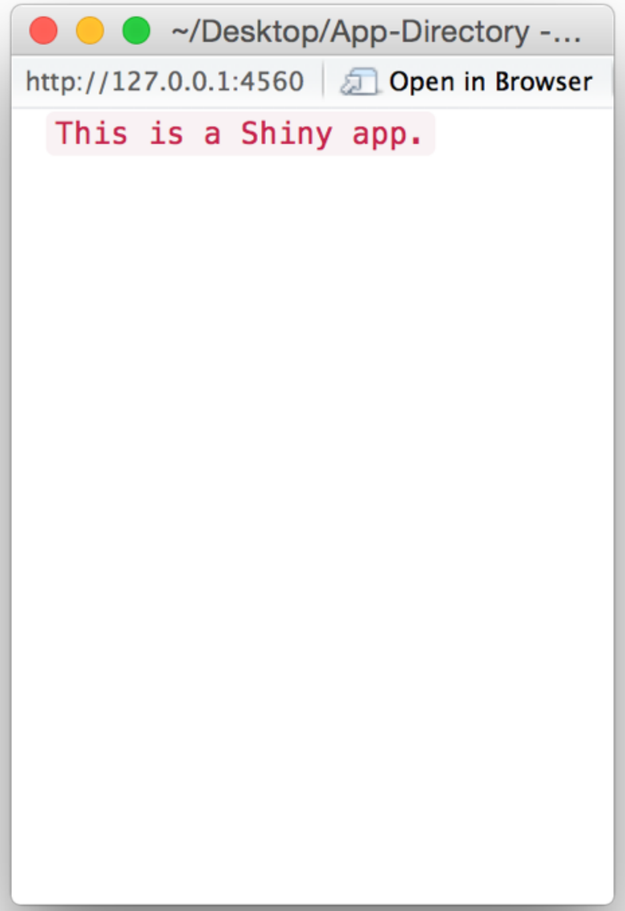


code()

Monospaced text (code)

```
fluidPage(
  tags$code("This is a Shiny app.")
)
```

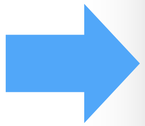


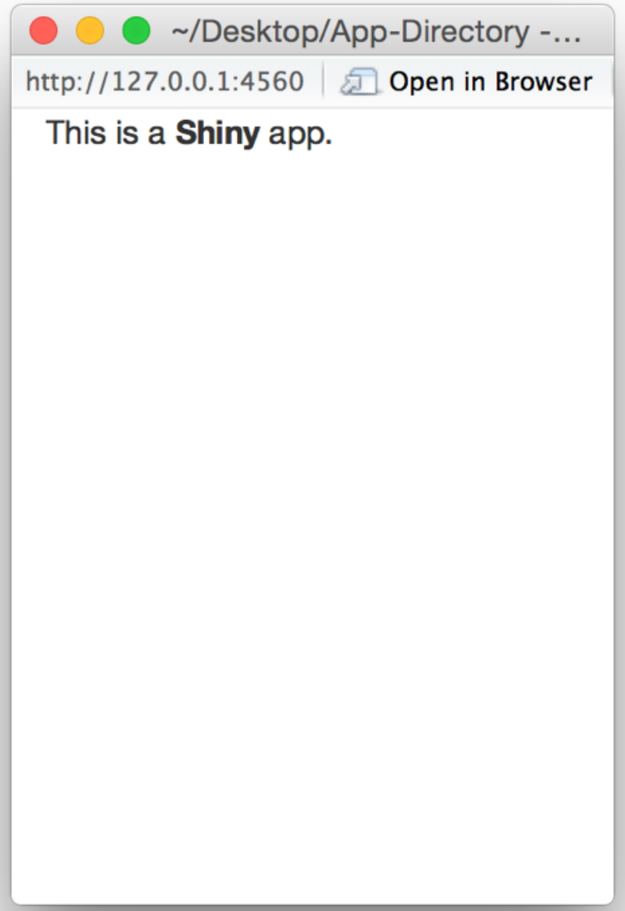


nesting

You can also nest functions inside of others

```
fluidPage(
  tags$p("This is a",
    tags$strong("Shiny"),
    "app.")
)
```

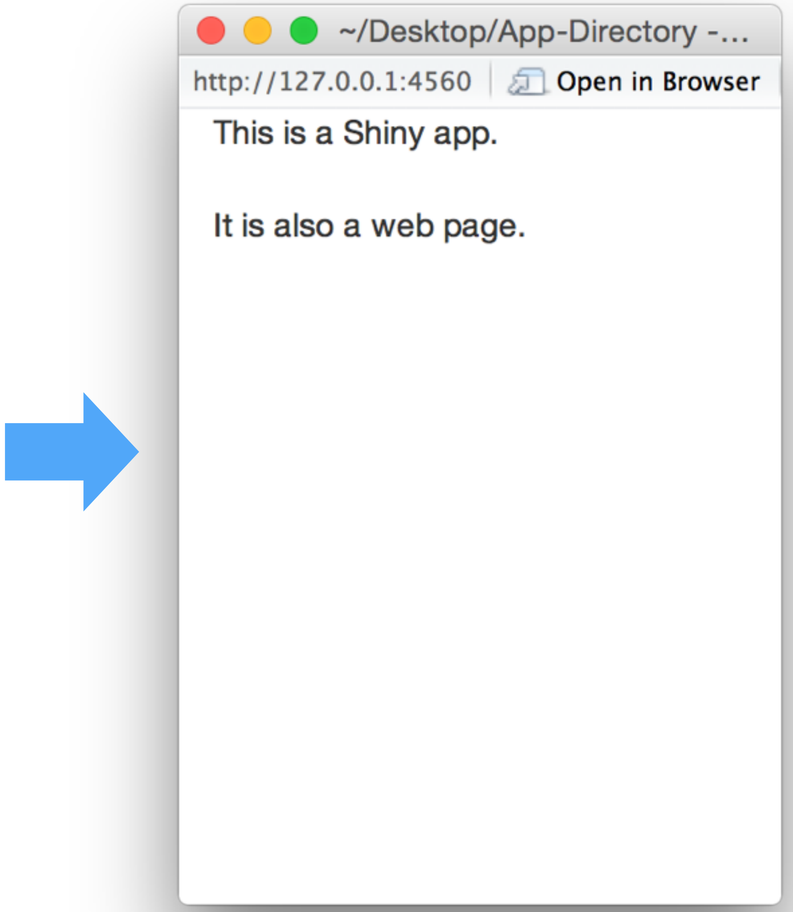




br()

A line break

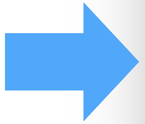
```
fluidPage(
  "This is a Shiny app.",
  tags$br(),
  "It is also a web page."
)
```

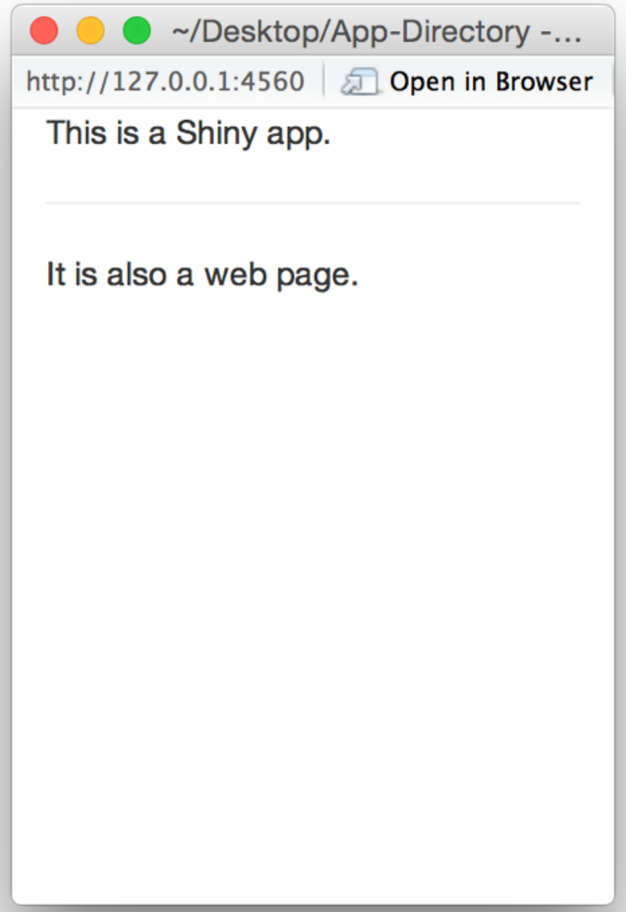


hr()

A horizontal rule

```
fluidPage(
  "This is a Shiny app.",
  tags$hr(),
  "It is also a web page."
)
```

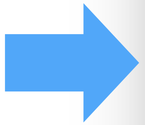




img()

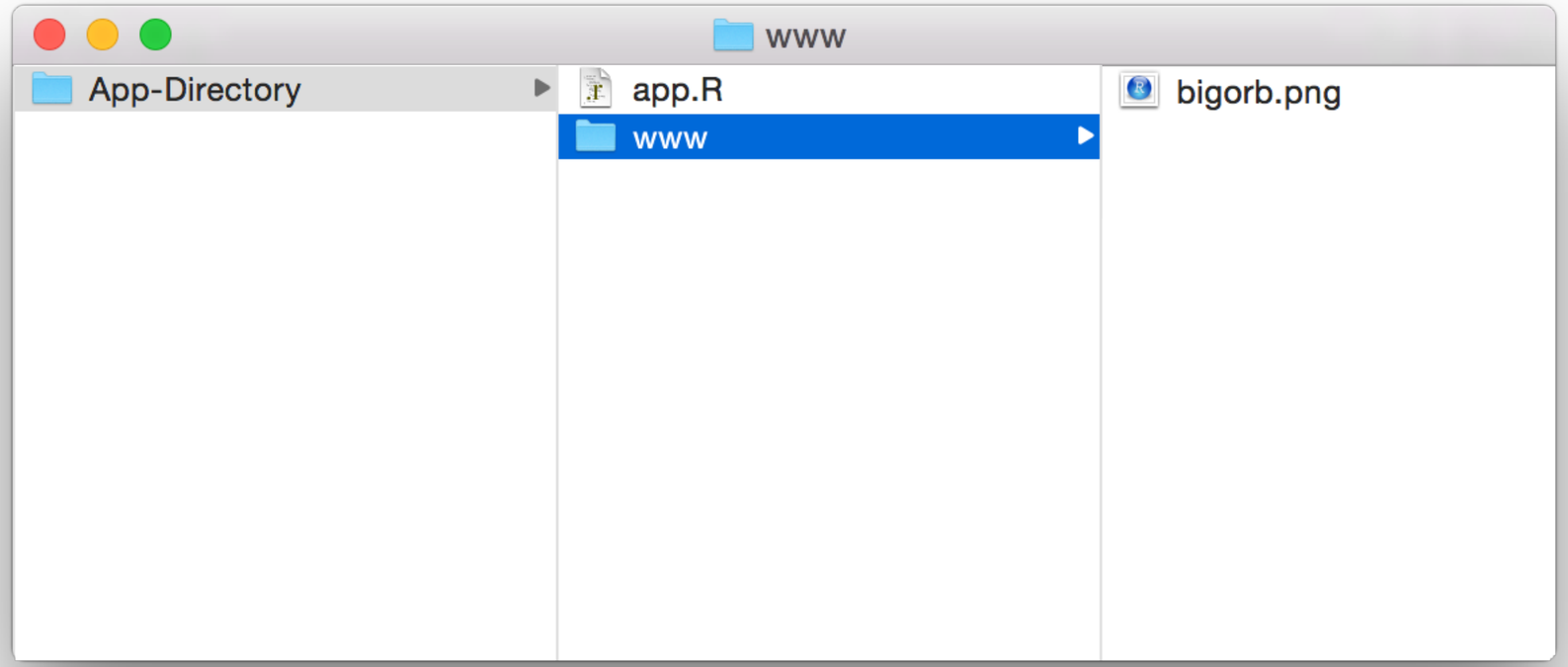
Add an image. Use **src** argument to point to the image URL.

```
fluidPage(
  tags$img(height = 100,
           width = 100,
           src = "http://www.rstudio.com/
                  images/RStudio.2x.png")
)
```



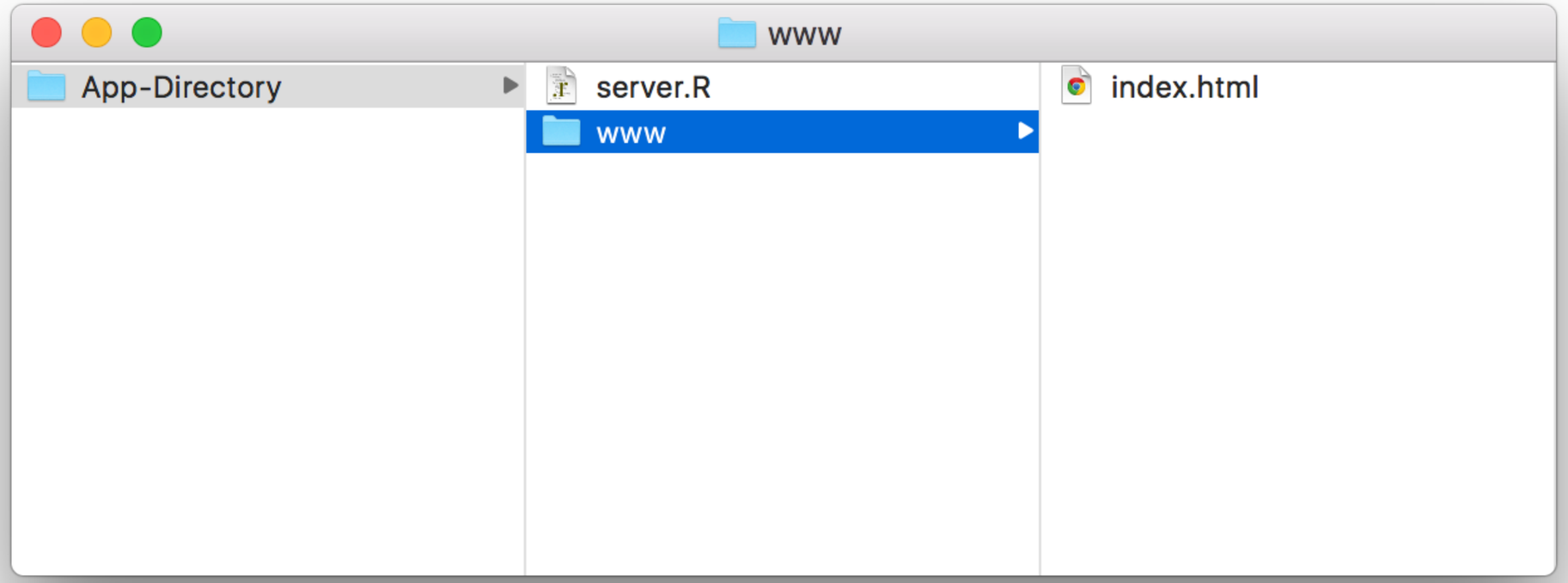
Adding Images

To add an image from a file, save the file in a subdirectory named **www**



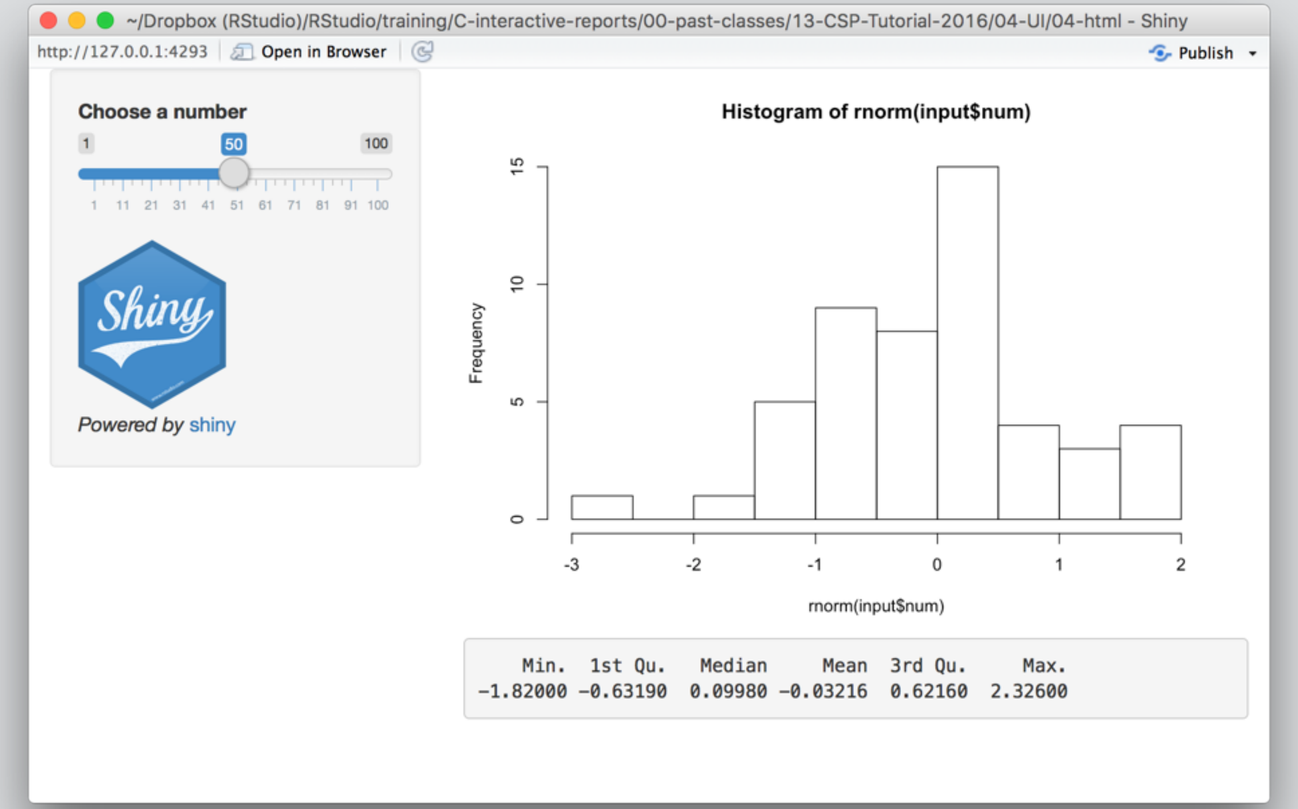
www

To share a file with your user's browser, put it in a subdirectory named **www**.



Your Turn

Use `tags$` and the image in `demos/www` to add an image, italic text, and a weblink (to shiny.rstudio.com) to your app.



Thank you

Learn more at
shiny.rstudio.com

Ask questions at
groups.google.com/forum/#!forum/shiny-discuss