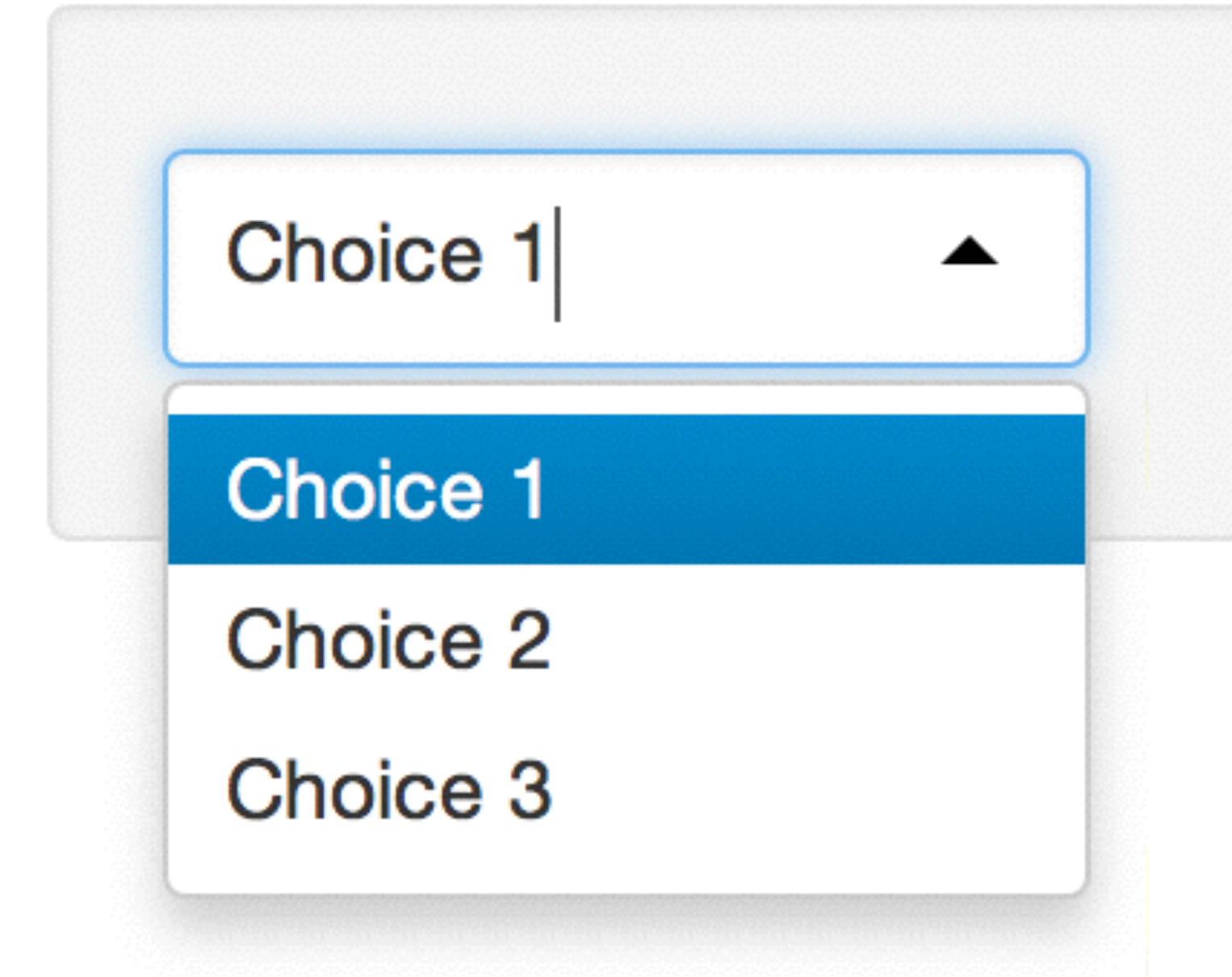


All Training materials are provided "as is" and without warranty and RStudio disclaims any and all express and implied warranties including without limitation the implied warranties of title, fitness for a particular purpose, merchantability and noninfringement.

The Training Materials are licensed under the Creative Commons Attribution-Noncommercial 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

How to start with Shiny, Part 1

How to build a Shiny App



Garrett Grolemund

Data Scientist and Master Instructor
May 2015
Email: garrett@rstudio.com

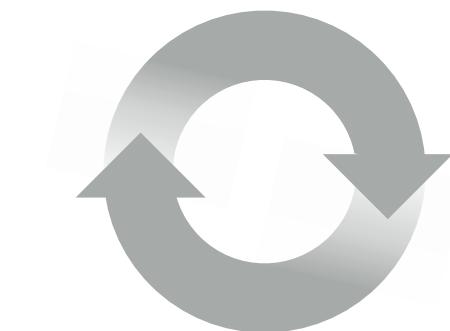
Code and slides at:

bit.ly/shiny-quickstart-1

How to start with Shiny

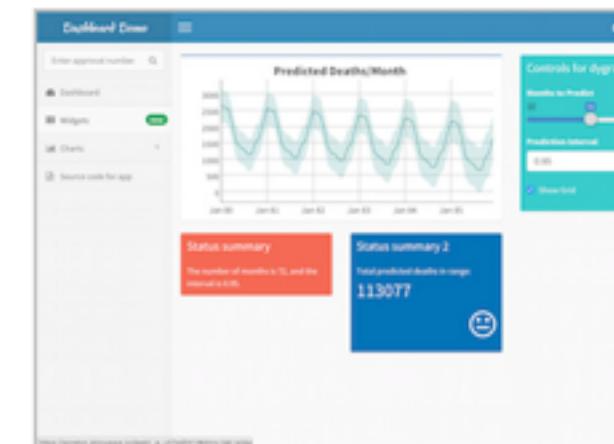


1. How to build a Shiny app (Today)
2. How to customize reactions (May 27)
3. How to customize appearance (June 3)



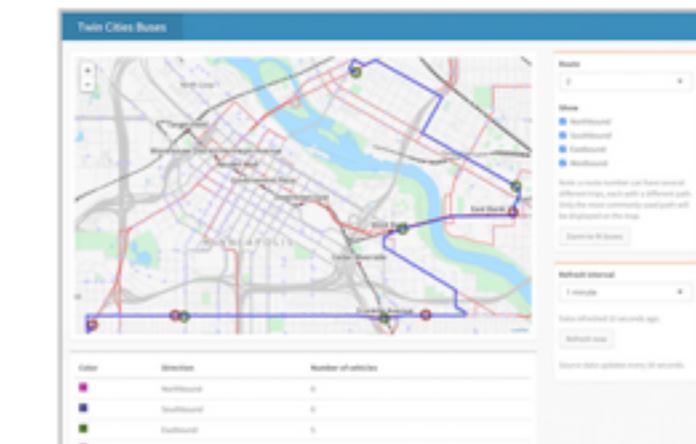


Shiny Apps for the Enterprise



[Shiny Dashboard Demo](#)

A dashboard built with Shiny.



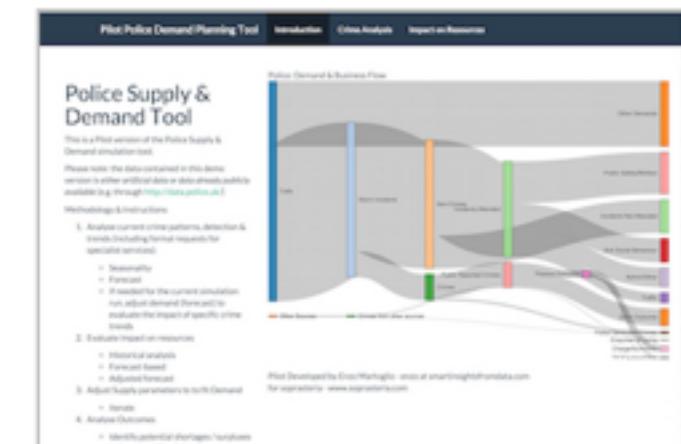
[Location tracker](#)

Track locations over time with streaming data.



[Download monitor](#)

Streaming download rates visualized as a bubble chart.



[Supply and Demand](#)

Forecast demand to plan resource allocation.

Shiny Showcase

www.rstudio.com/products/shiny/shiny-user-showcase/

Industry Specific Shiny Apps



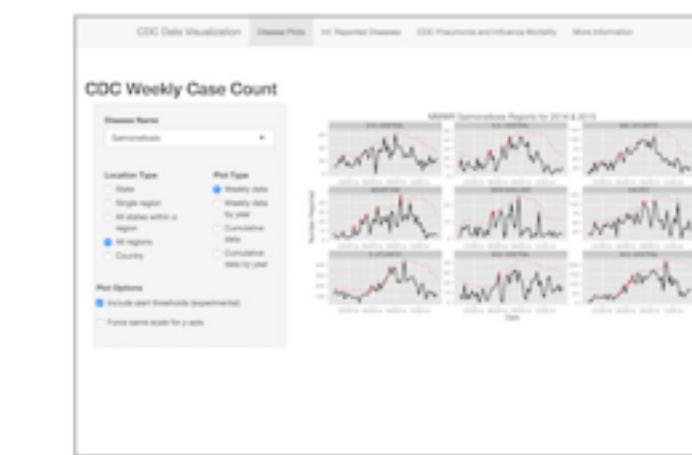
[Economic Dashboard](#)

Economic forecasting with macroeconomic indicators.



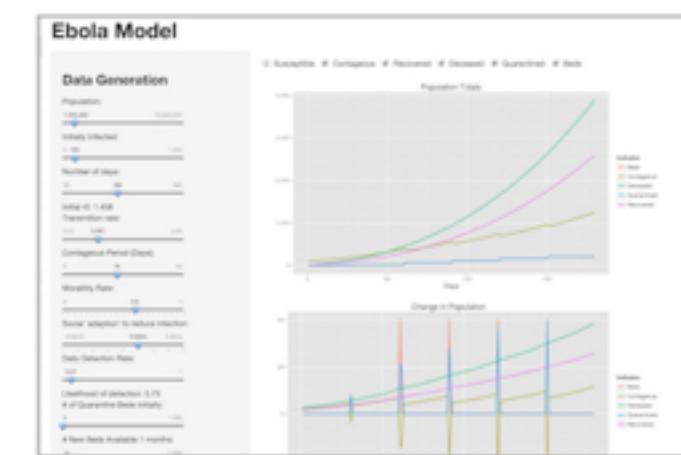
[ER Optimization](#)

An app that models patient flow.



[CDC Disease Monitor](#)

Alert thresholds and automatic weekly updates.



[Ebola Model](#)

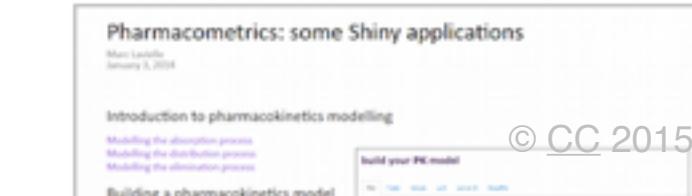
An epidemiological simulation.



[Pharmacometrics](#)



[Pharmacokinetics modelling](#)

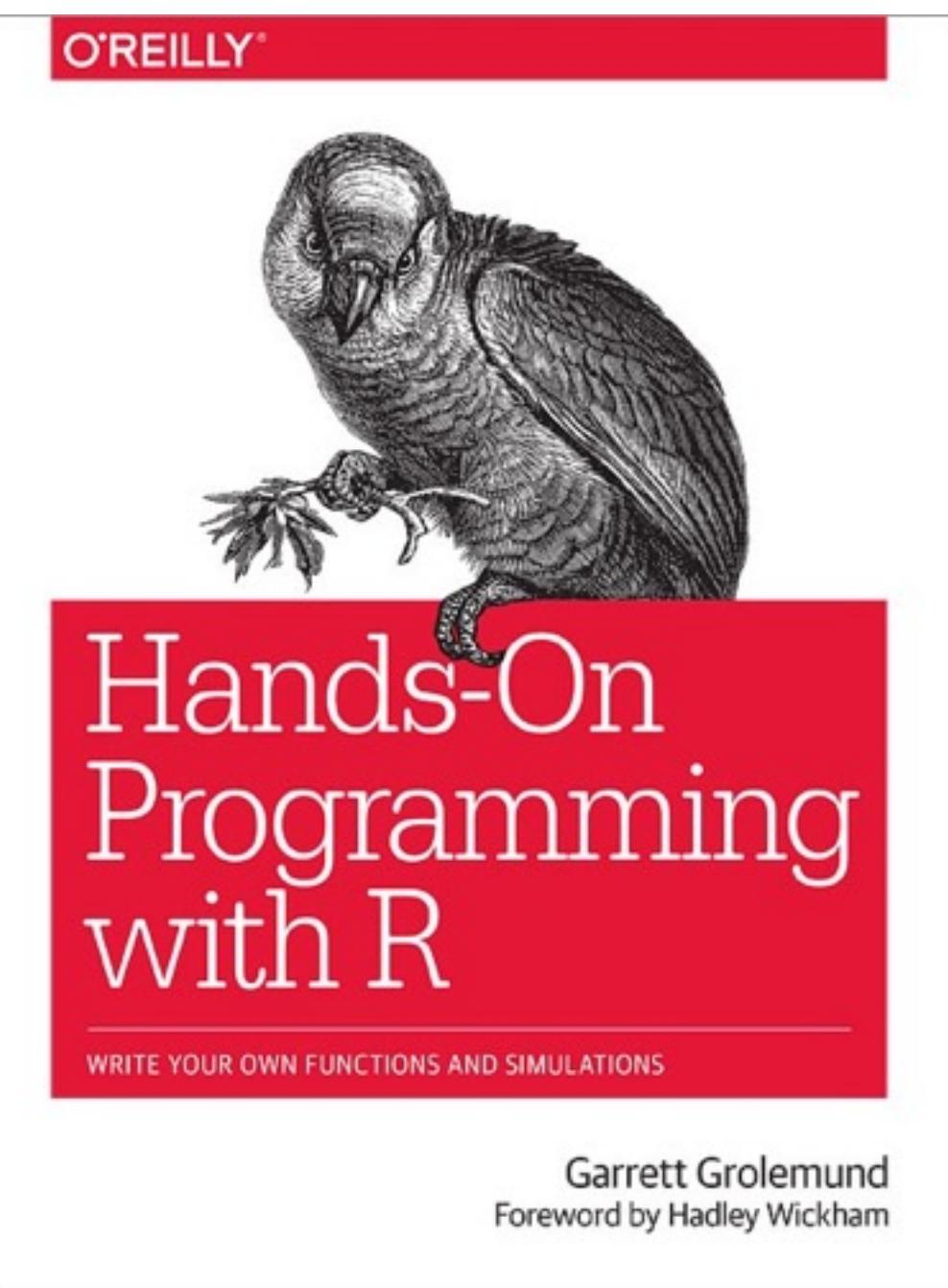


[Lake Erie Biological Station - Western Basin Trawl Survey](#)

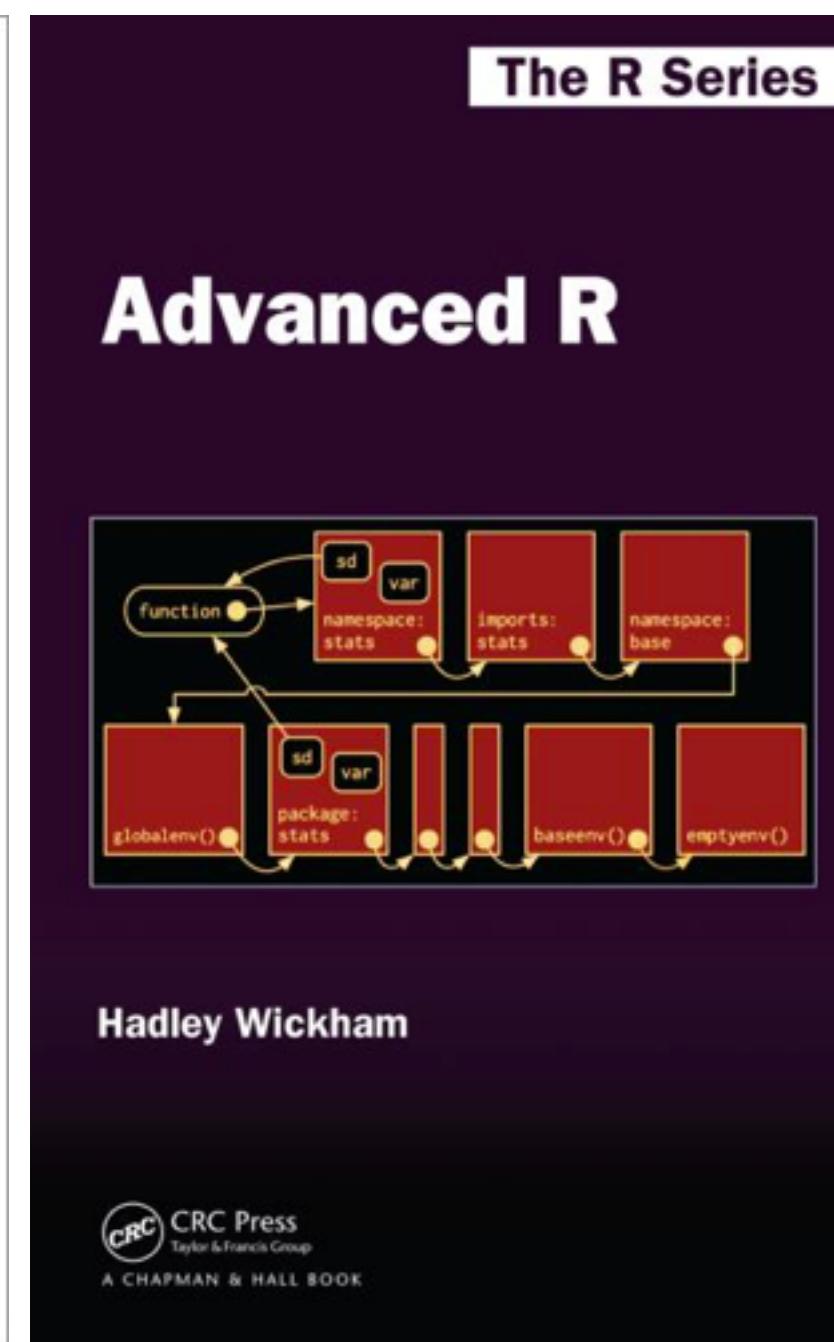


Learn R

Books



[shop.oreilly.com/product/
0636920028574.do](http://shop.oreilly.com/product/0636920028574.do)

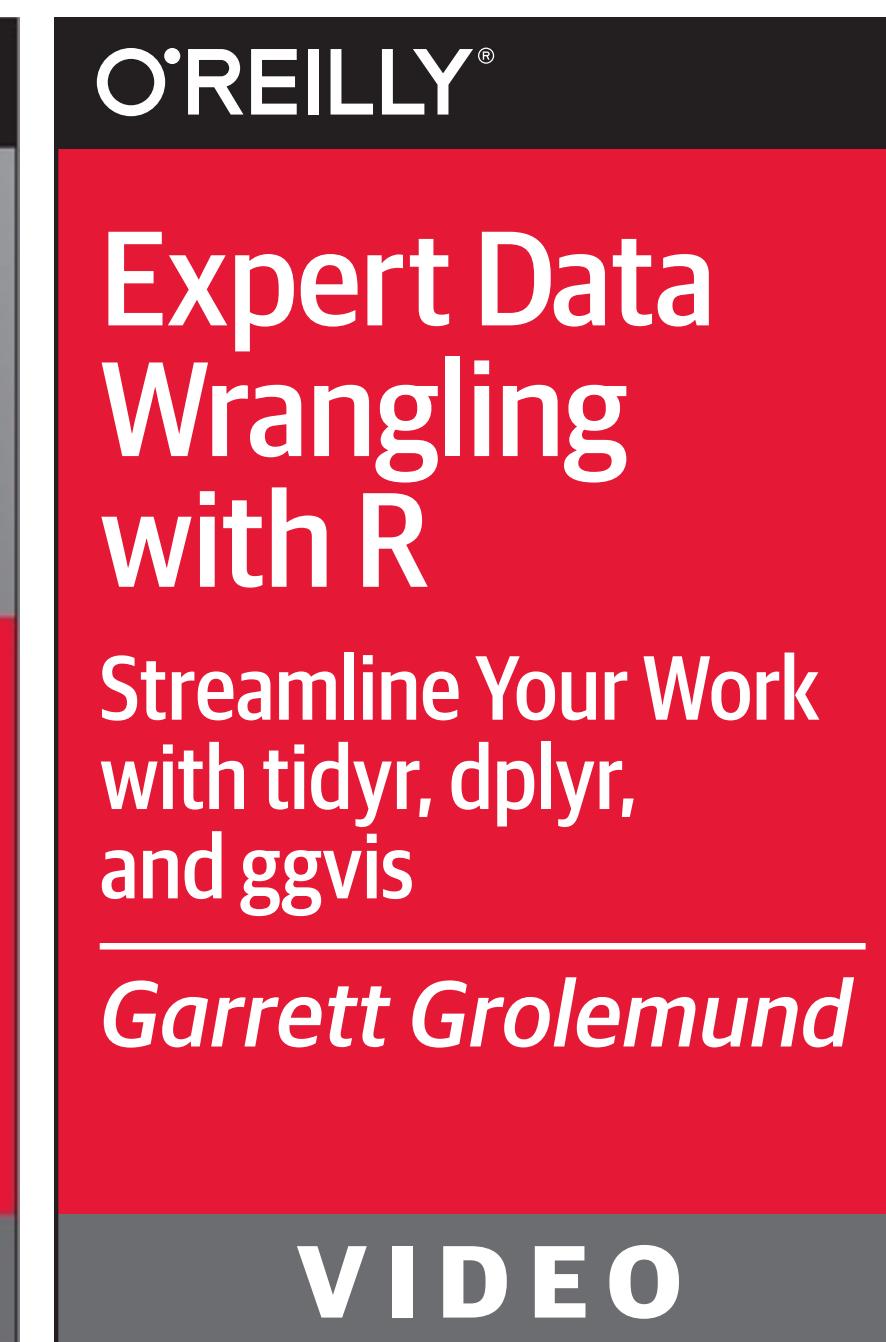


adv-r.had.co.nz/



[shop.oreilly.com/product/
0636920034834.do](http://shop.oreilly.com/product/0636920034834.do)

Videos



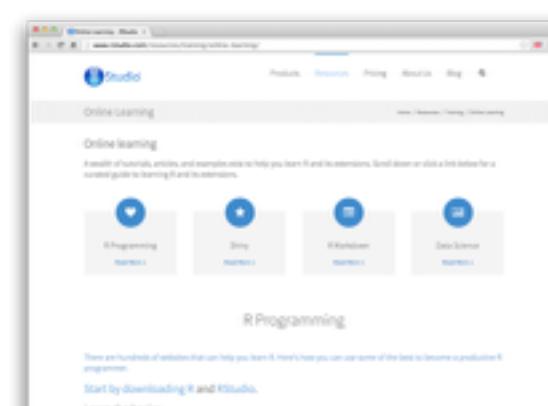
[shop.oreilly.com/product/
0636920035992.do](http://shop.oreilly.com/product/0636920035992.do)

Interactive tutorials



DataCamp

www.datacamp.com

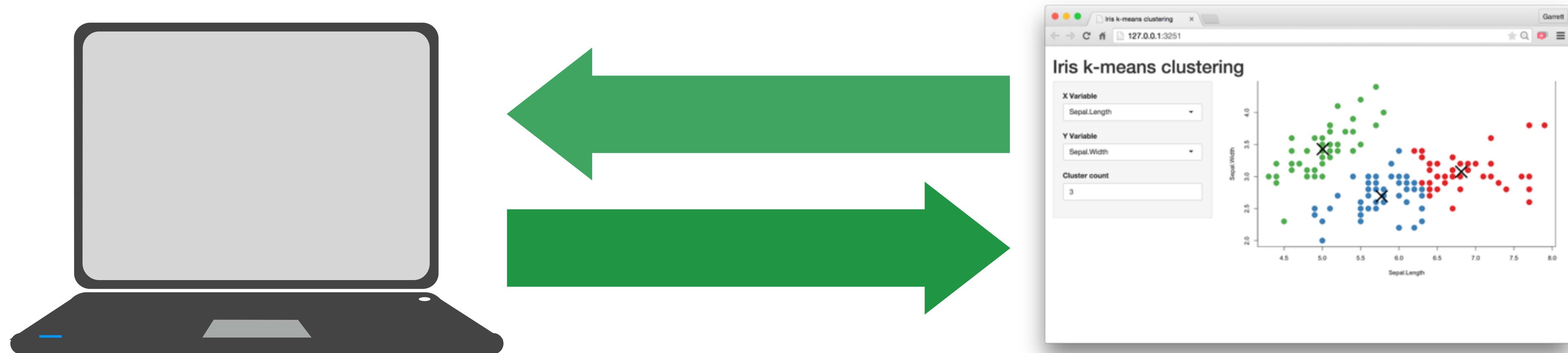


More at

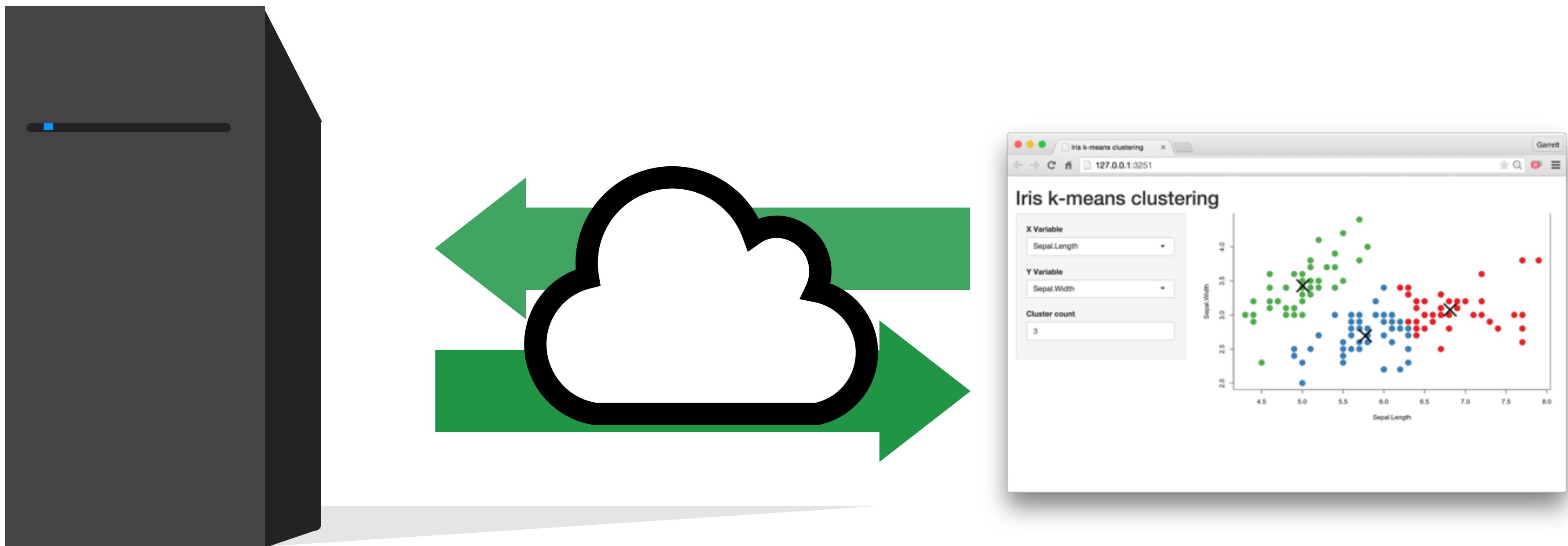
www.rstudio.com/resources/training/online-learning/

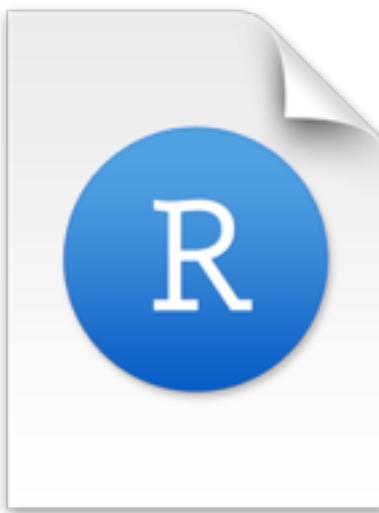
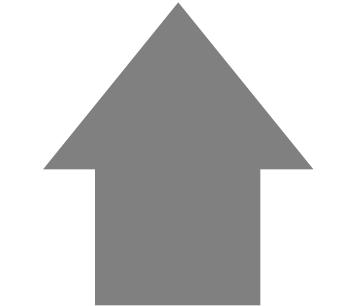
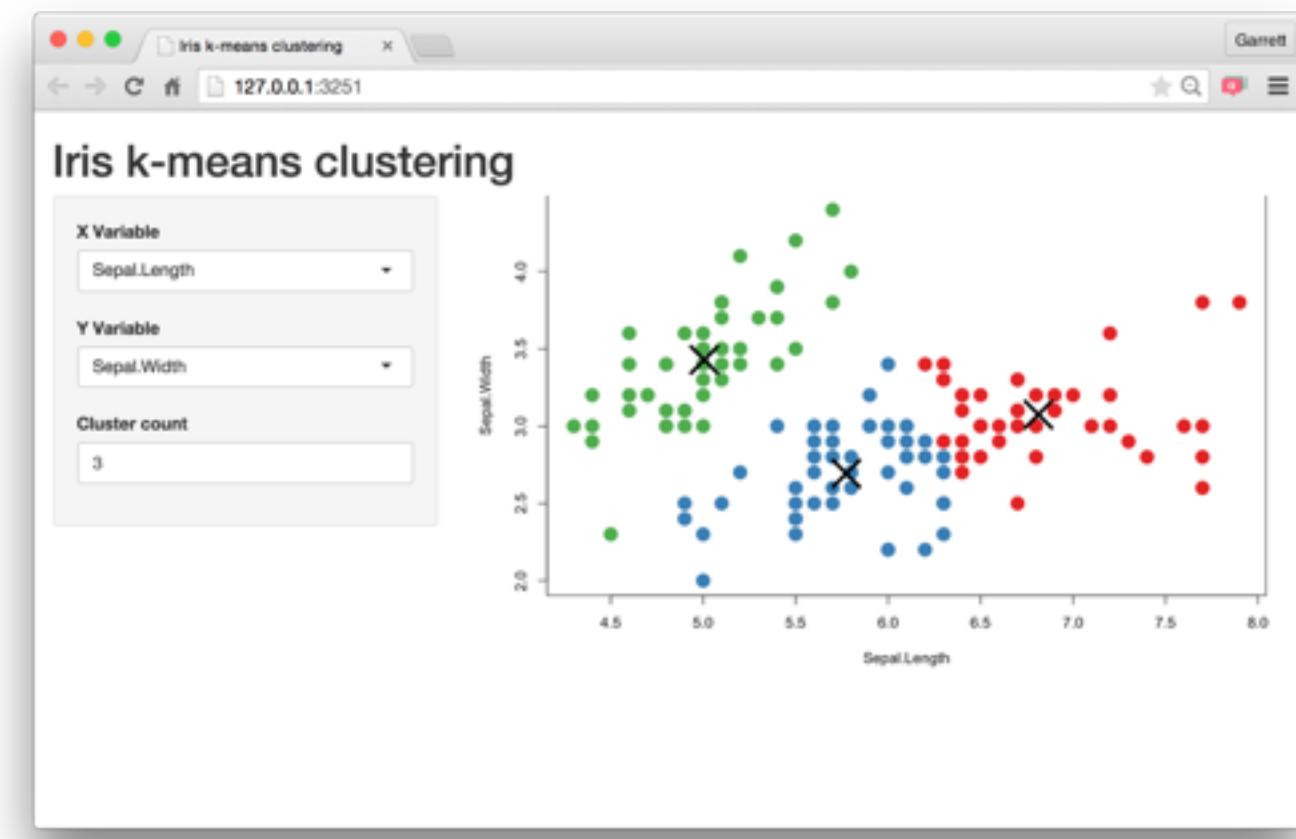
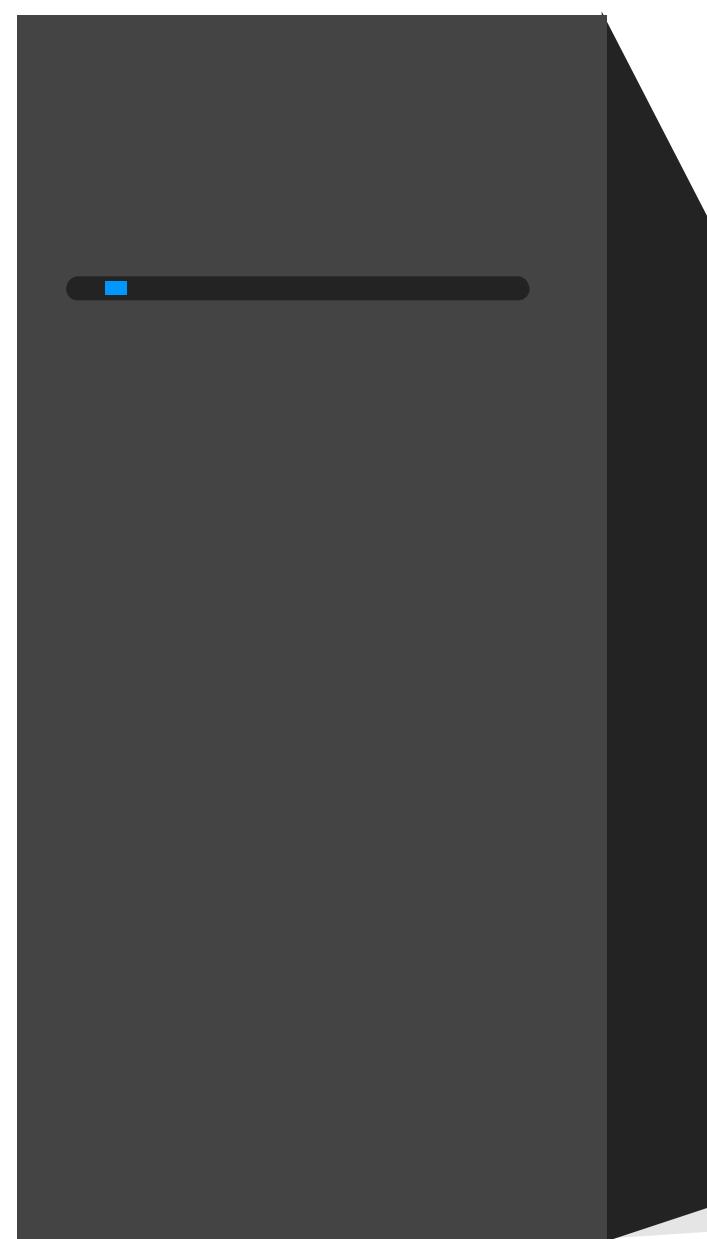
**Understand the
architecture**

Every Shiny app is maintained by a computer running R

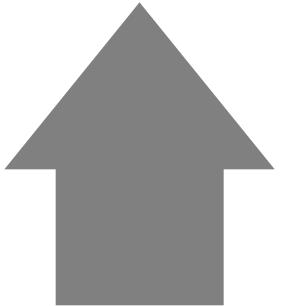


Every Shiny app is maintained by a computer running R





Server Instructions



User Interface (UI)

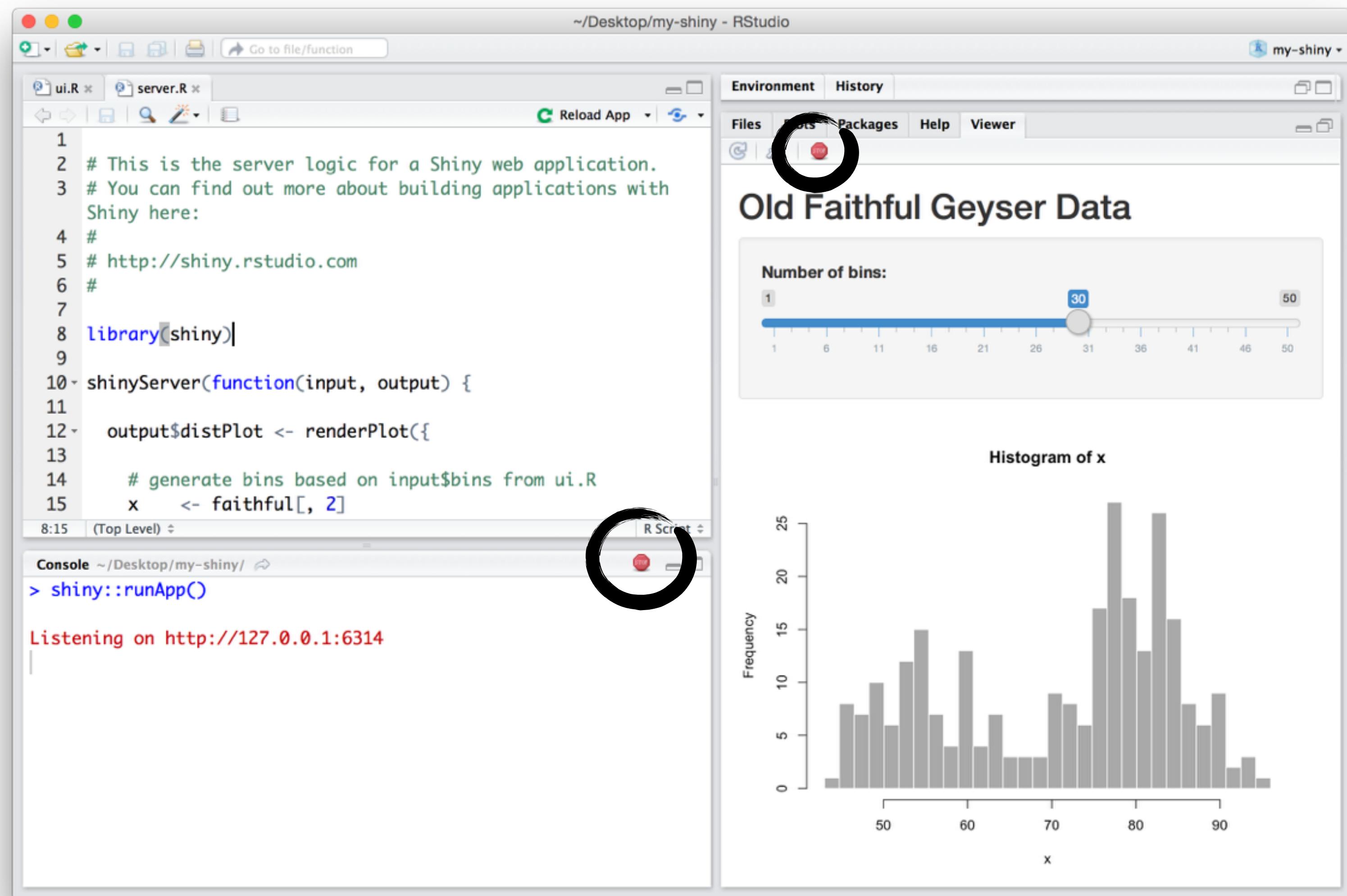
**Use the
template**

App template

The shortest viable shiny app

```
library(shiny)  
  
ui <- fluidPage()  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```

Close an app

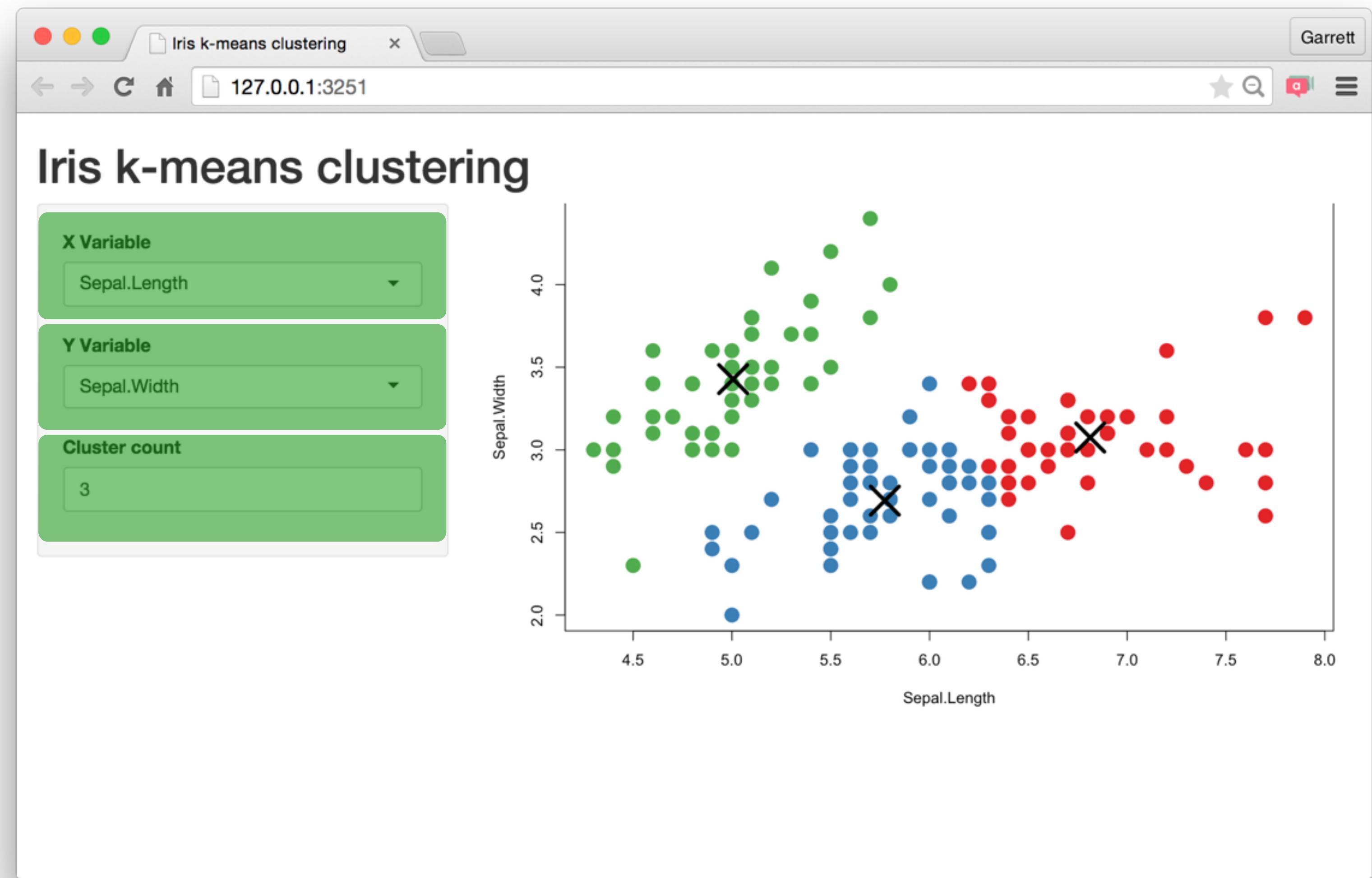


Add elements to your app as arguments to `fluidPage()`

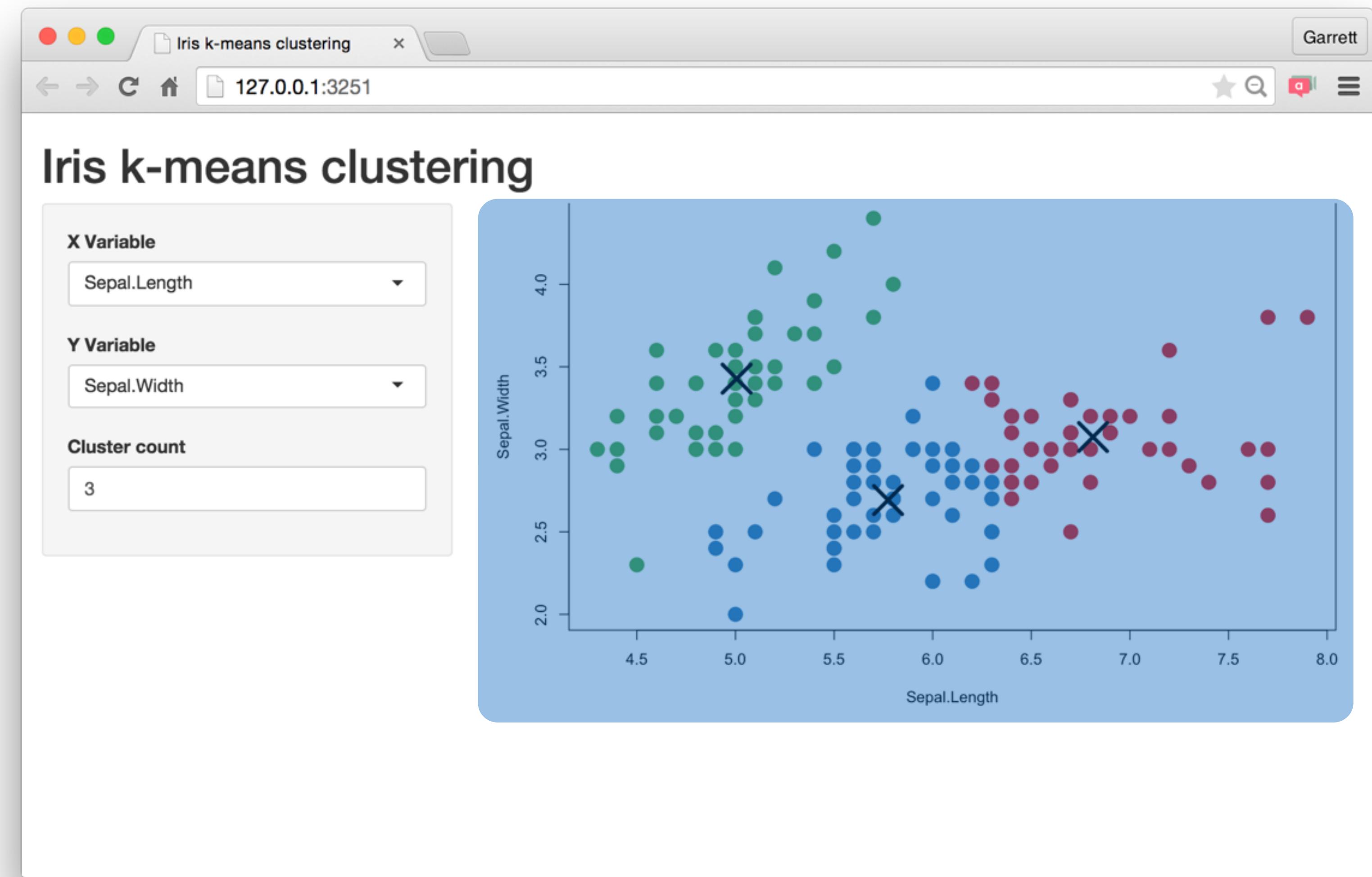
```
library(shiny)  
  
ui <- fluidPage("Hello World")  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```

**Build your app around
Inputs and
Outputs**

Build your app around **inputs** and **outputs**



Build your app around **inputs** and **outputs**



Add elements to your app as arguments to
`fluidPage()`

```
ui <- fluidPage(  
  # *Input() functions,  
  # *Output() functions  
)
```

Inputs

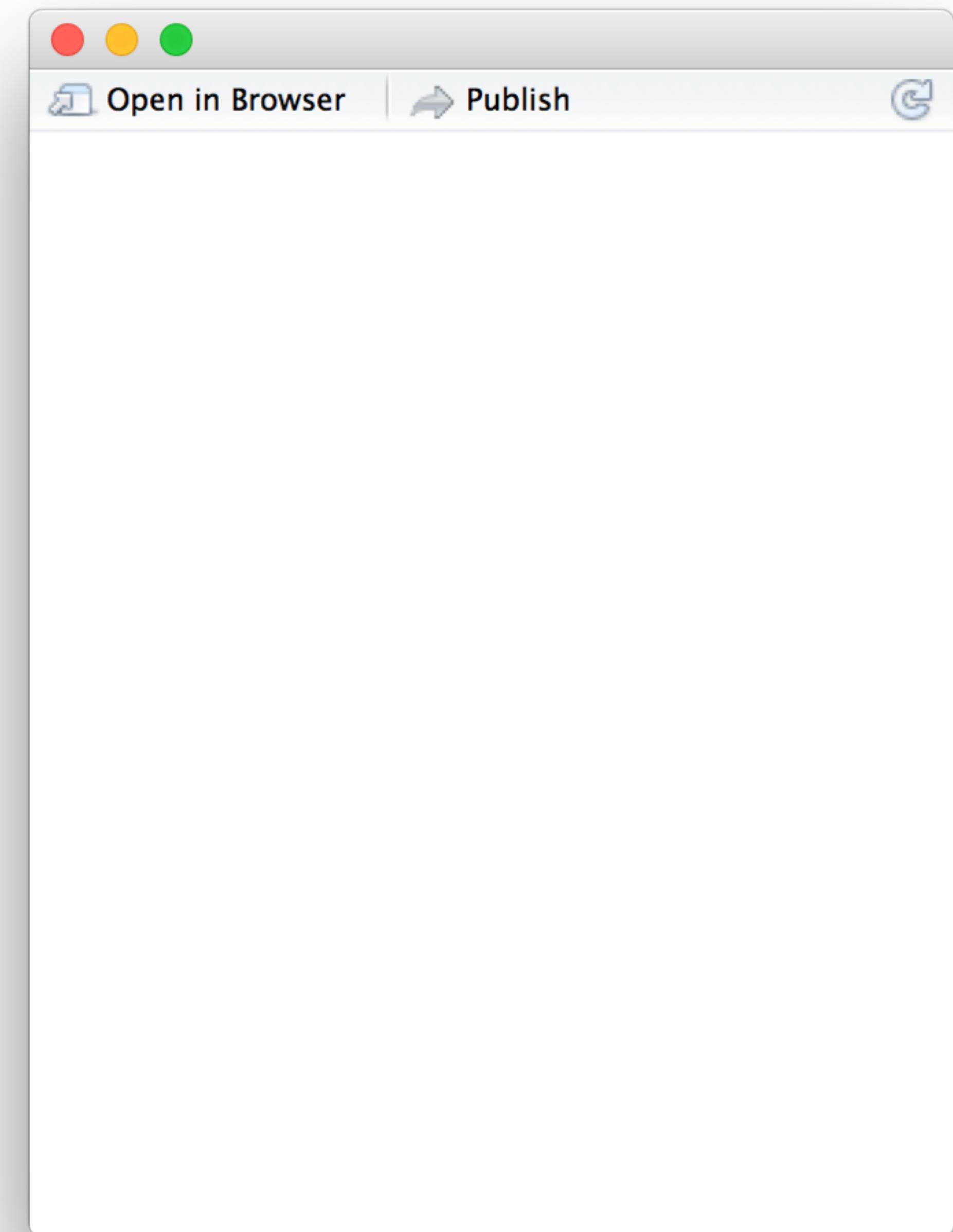
Create an input with an ***Input()** function.

```
sliderInput(inputId = "num",
  label = "Choose a number",
  value = 25, min = 1, max = 100)
```

```
<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-prettyify-separator="," data-keyboard="true"
    data-keyboard-step="1.010101010101"/>
</div>
```

Create an input with an input function.

```
library(shiny)  
ui <- fluidPage(  
  
)  
  
server <- function(input, output) {}  
  
shinyApp(server = server, ui = ui)
```

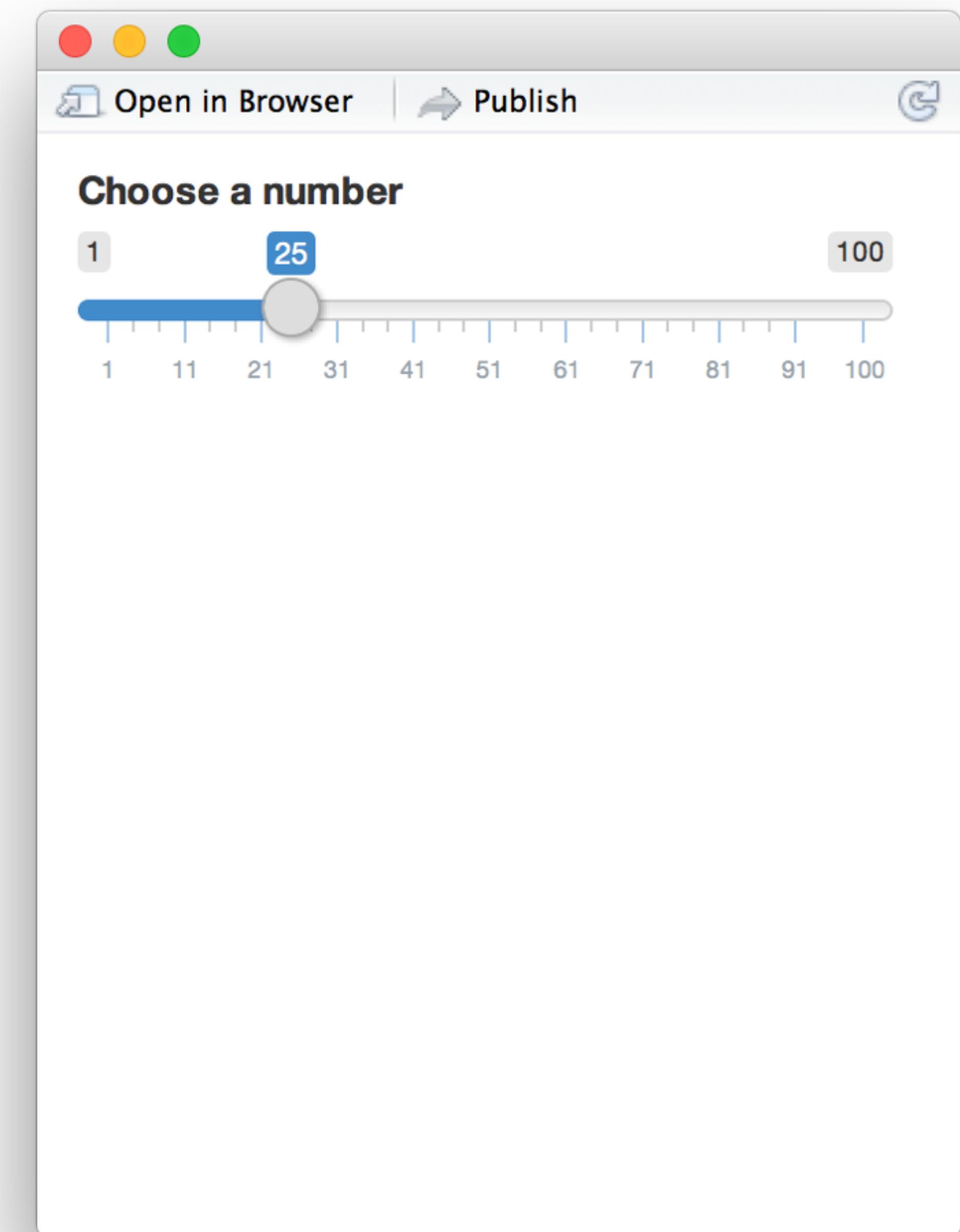


Create an input with an input function.

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

```
server <- function(input, output) {}

shinyApp(server = server, ui = ui)
```



Buttons

Action

Submit

actionButton()
submitButton()

Date range

2014-01-24 to 2014-01-24

dateRangeInput()

Radio buttons

- Choice 1
- Choice 2
- Choice 3

radioButtons()

Single checkbox

Choice A

checkboxInput()

File input

No file chosen

fileInput()

Select box

Choice 1

selectInput()

Checkbox group

Choice 1

Choice 2

Choice 3

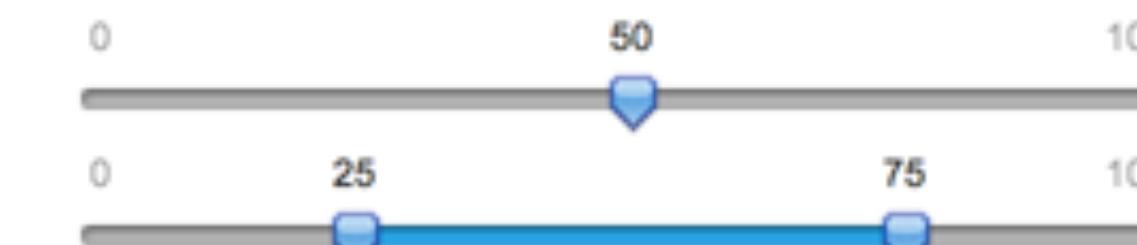
checkboxGroupInput() dateInput()

Numeric input

1

numericInput()

Sliders



sliderInput()

Date input

2014-01-01

.....

passwordInput()

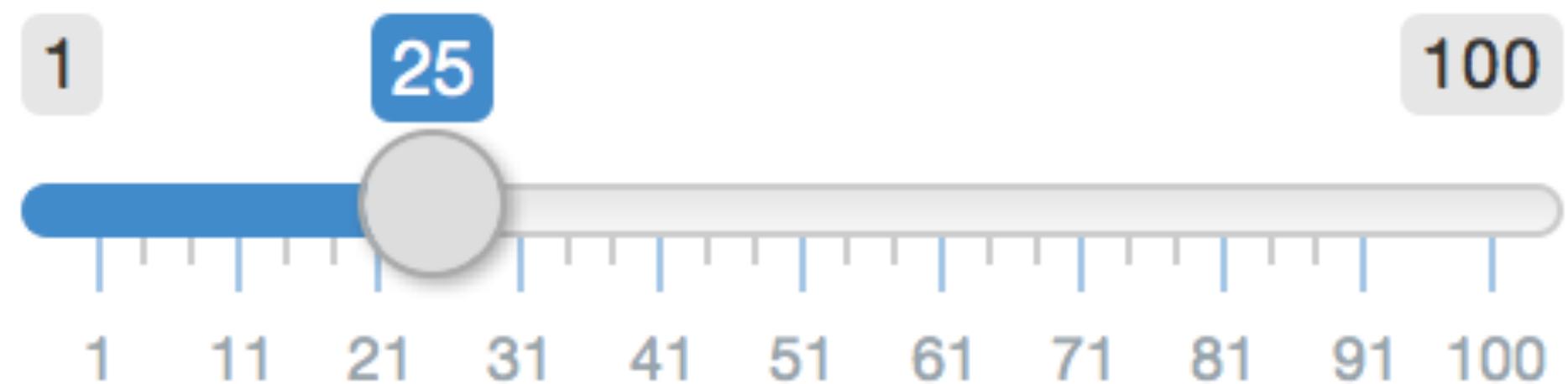
Text input

Enter text...

textInput()

Syntax

Choose a number



```
sliderInput(inputId = "num", label = "Choose a number", ...)
```

input name
(for internal use)

Notice:
Id not ID

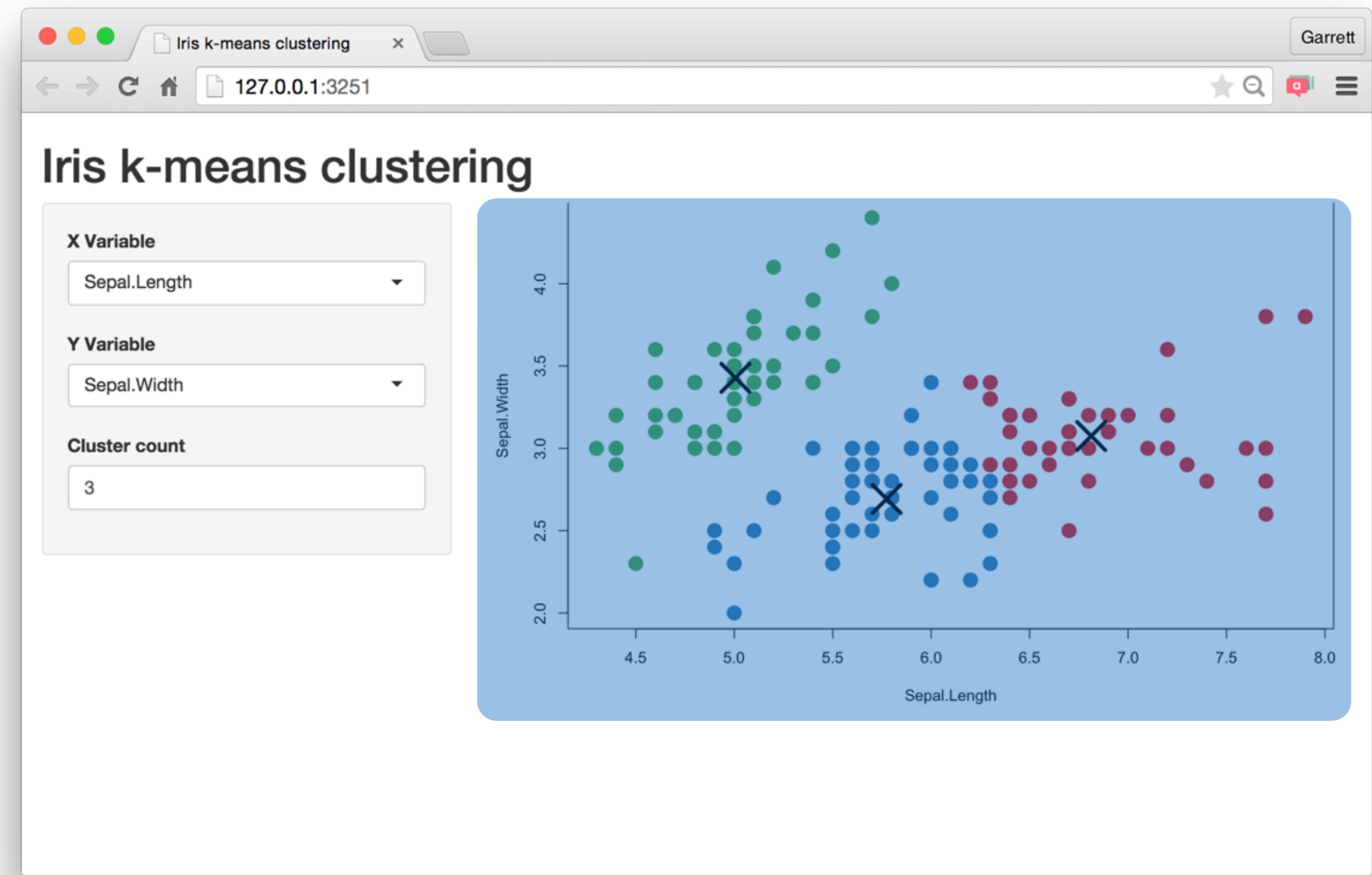
label to
display

input specific
arguments

?sliderInput

Outputs

Build your app around **inputs** and **outputs**



Function	Inserts
dataTableOutput()	an interactive table
htmlOutput()	raw HTML
imageOutput()	image
plotOutput()	plot
tableOutput()	table
textOutput()	text
uiOutput()	a Shiny UI element
verbatimTextOutput()	text

*Output()

To display output, add it to `fluidPage()` with an
`*Output()` function

```
plotOutput("hist")
```

the type of output
to display

name to give to the
output object

```
library(shiny)

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

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```

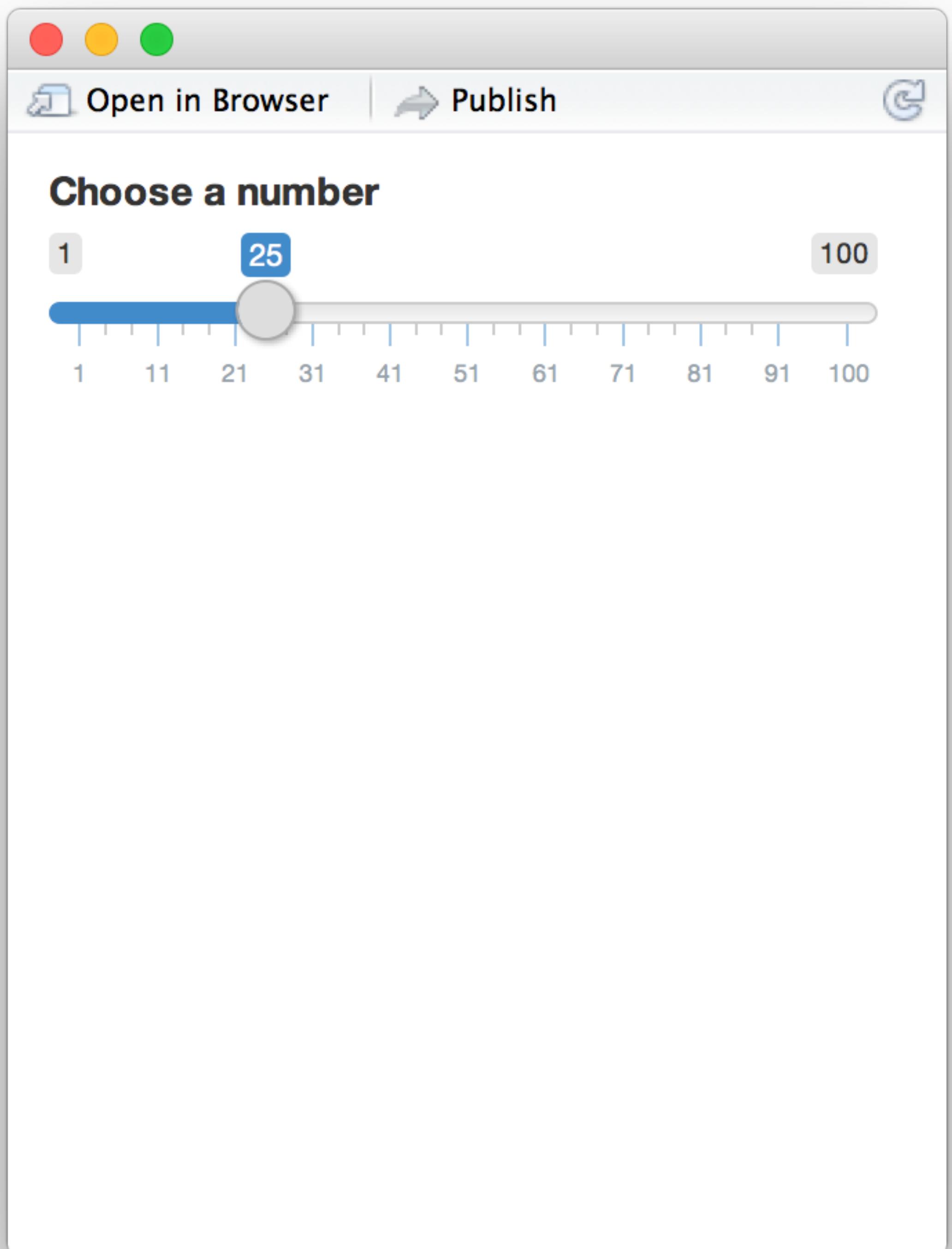
Comma between
arguments

```
library(shiny)

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

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```

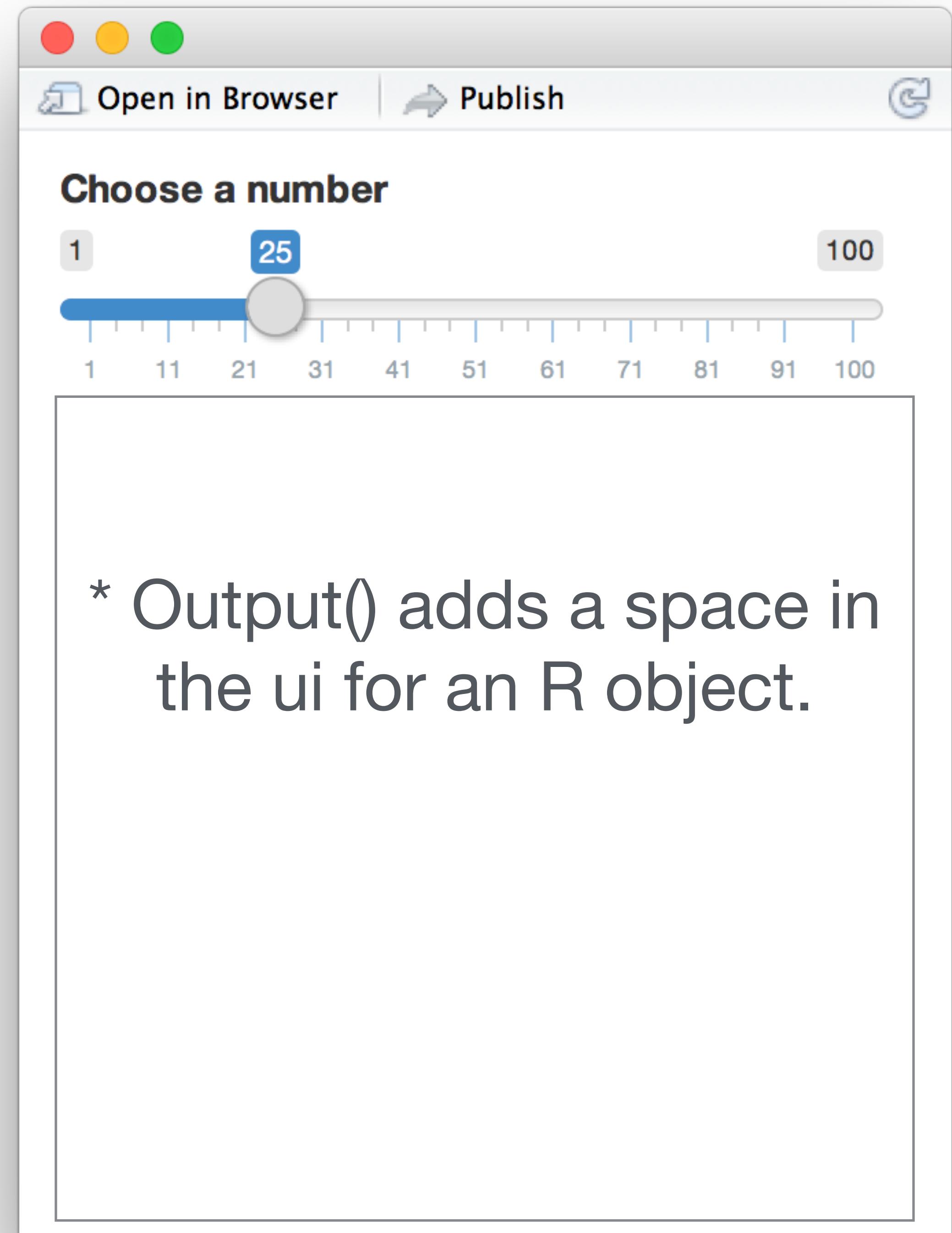


```
library(shiny)

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

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```

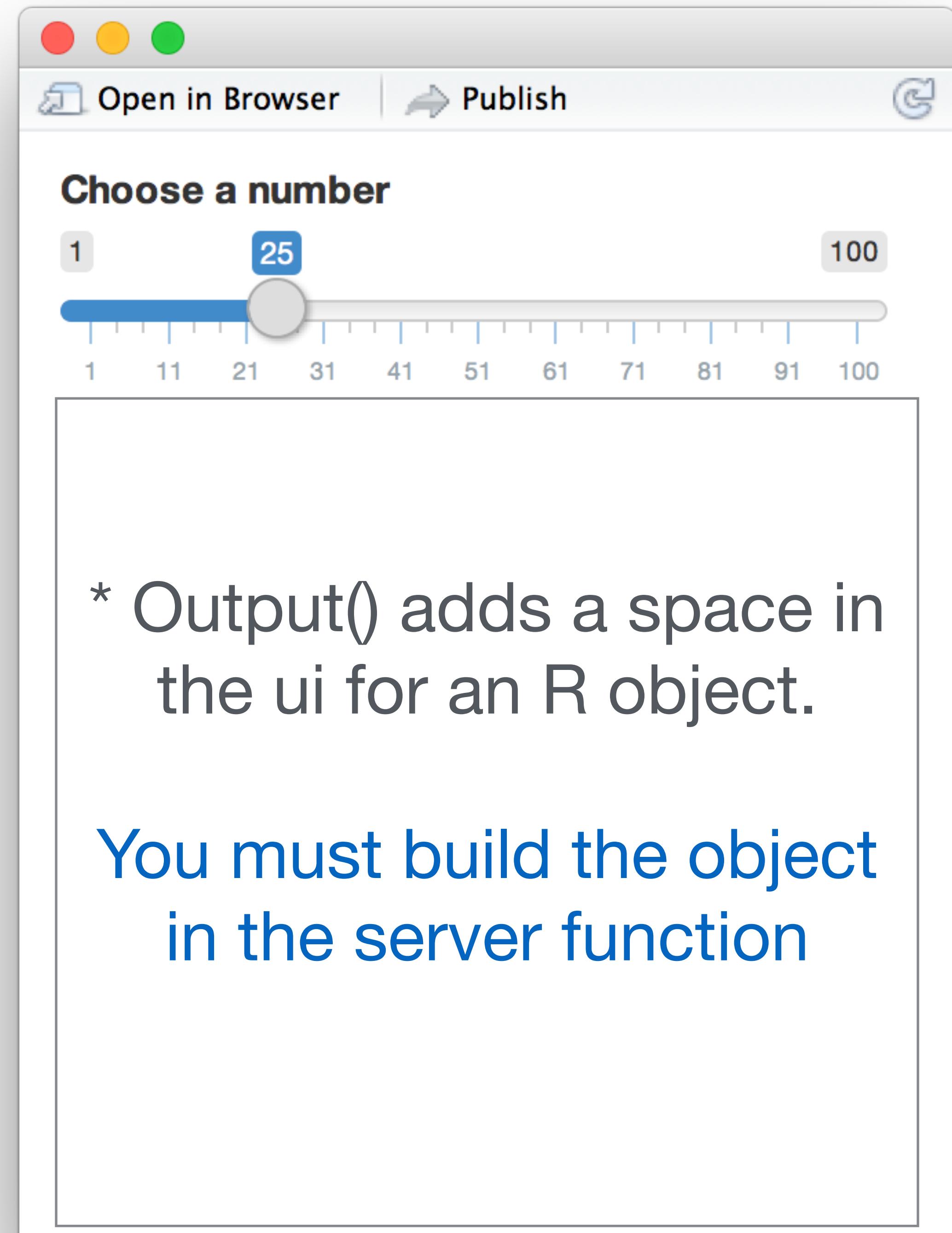


```
library(shiny)

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

server <- function(input, output) {}

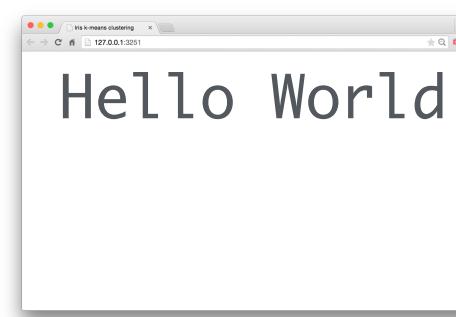
shinyApp(ui = ui, server = server)
```



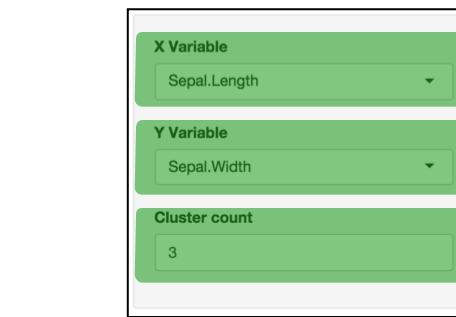
Recap

Begin each app with the template

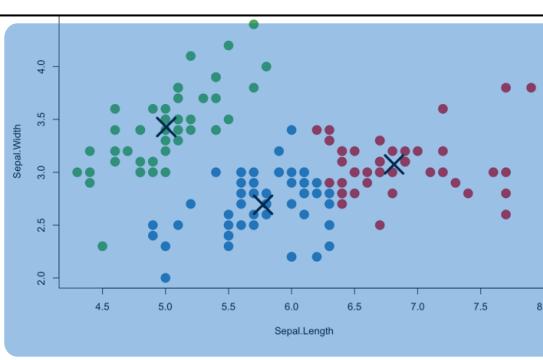
```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```



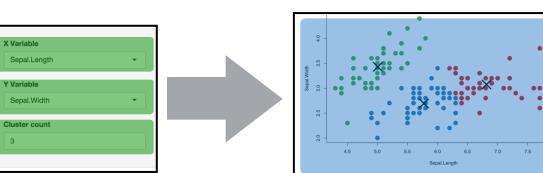
Add elements as arguments to **fluidPage()**



Create reactive inputs with an ***Input()** function



Display reactive results with an ***Output()** function



Assemble outputs from inputs in the server function

Tell the
server
how to assemble
inputs into outputs

Use **3 rules** to write the server function

```
server <- function(input, output) {  
}  
}
```

1

Save objects to display to output\$

```
server <- function(input, output) {  
  output$hist <- # code  
}
```

1

Save objects to display to output\$

output\$hist



plotOutput("hist")

2

Build objects to display with **render***()

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

Use the **render***() function that creates the type of output you wish to make.

function	creates
renderDataTable()	An interactive table <small>(from a data frame, matrix, or other table-like structure)</small>
renderImage()	An image (saved as a link to a source file)
renderPlot()	A plot
renderPrint()	A code block of printed output
renderTable()	A table <small>(from a data frame, matrix, or other table-like structure)</small>
renderText()	A character string
renderUI()	a Shiny UI element

render*()

Builds reactive output to display in UI

```
renderPlot({ hist(rnorm(100)) })
```

type of object to build

code block that builds the object

2

Build objects to display with **render***()

```
server <- function(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(100))  
  })  
}
```

2

Build objects to display with **render***()

```
server <- function(input, output) {  
  output$hist <- renderPlot({  
    title <- "100 random normal values"  
    hist(rnorm(100), main = title)  
  })  
}
```

3

Access **input** values with **input\$**

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

3

Access **input** values with **input\$**

```
sliderInput(inputId = "num", ...)
```



input\$num

Input values

The input value changes whenever a user changes the input.

Choose a number

A slider input with a title "Choose a number". The slider has a blue track and a grey handle. The value "25" is displayed in a blue box above the slider. The slider scale shows tick marks at 1, 11, 21, 31, 41, 51, 61, 71, 81, 91, and 100.

input\$num = 25

Choose a number

A slider input with a title "Choose a number". The slider has a blue track and a grey handle. The value "50" is displayed in a blue box above the slider. The slider scale shows tick marks at 1, 11, 21, 31, 41, 51, 61, 71, 81, 91, and 100.

input\$num = 50

Choose a number

A slider input with a title "Choose a number". The slider has a blue track and a grey handle. The value "75" is displayed in a blue box above the slider. The slider scale shows tick marks at 1, 11, 21, 31, 41, 51, 61, 71, 81, 91, and 100.

input\$num = 75

Input values

The input value changes whenever a user changes the input.

Choose a number

A slider input with a title "Choose a number". The slider has a blue track and a grey handle. The value "25" is displayed in a blue box above the slider. The slider scale ranges from 1 to 100 with major tick marks every 10 units and minor tick marks every 1 unit.

input\$num = 25

Choose a number

A slider input with a title "Choose a number". The slider has a blue track and a grey handle. The value "50" is displayed in a blue box above the slider. The slider scale ranges from 1 to 100 with major tick marks every 10 units and minor tick marks every 1 unit.

input\$num = 50

Choose a number

A slider input with a title "Choose a number". The slider has a blue track and a grey handle. The value "75" is displayed in a blue box above the slider. The slider scale ranges from 1 to 100 with major tick marks every 10 units and minor tick marks every 1 unit.

input\$num =

Output will automatically update
if you follow the 3 rules

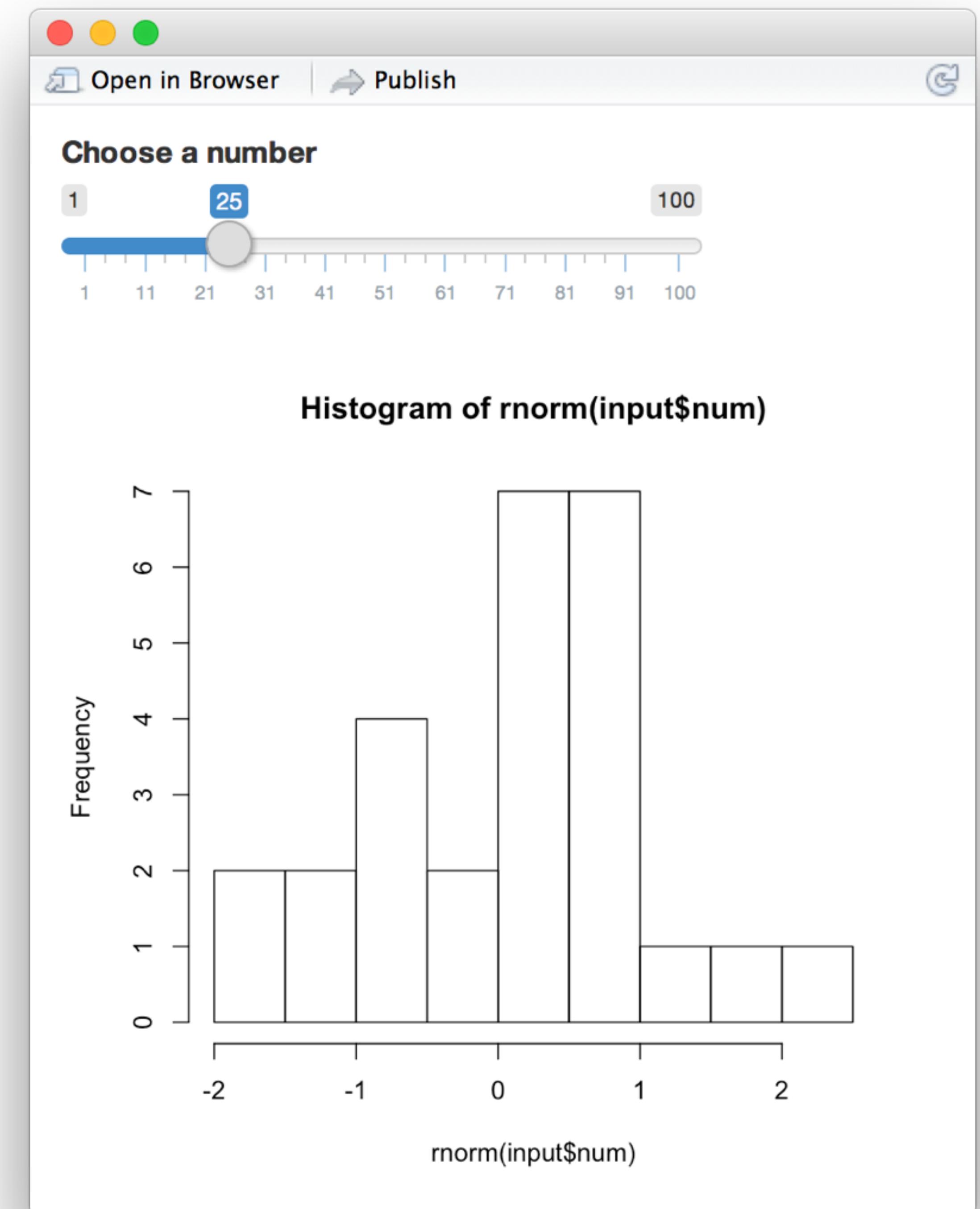
Reactivity 101

Reactivity automatically occurs whenever you use an input value to render an output object

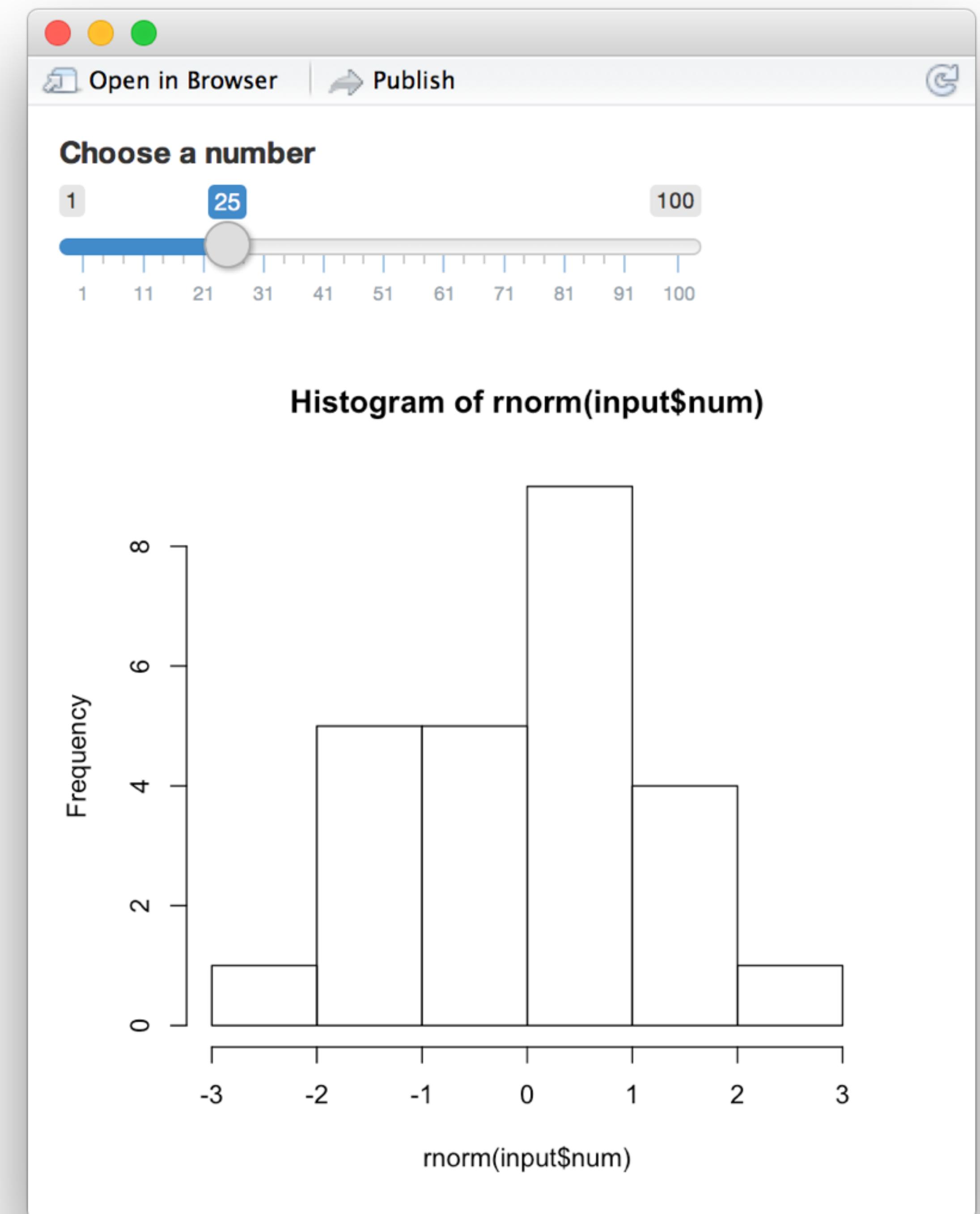
```
function(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(input$num))  
  })  
}
```

input\$num

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



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



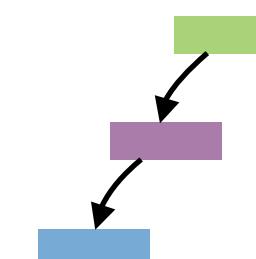
Recap: Server



`output$hist <-`

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

`input$num`



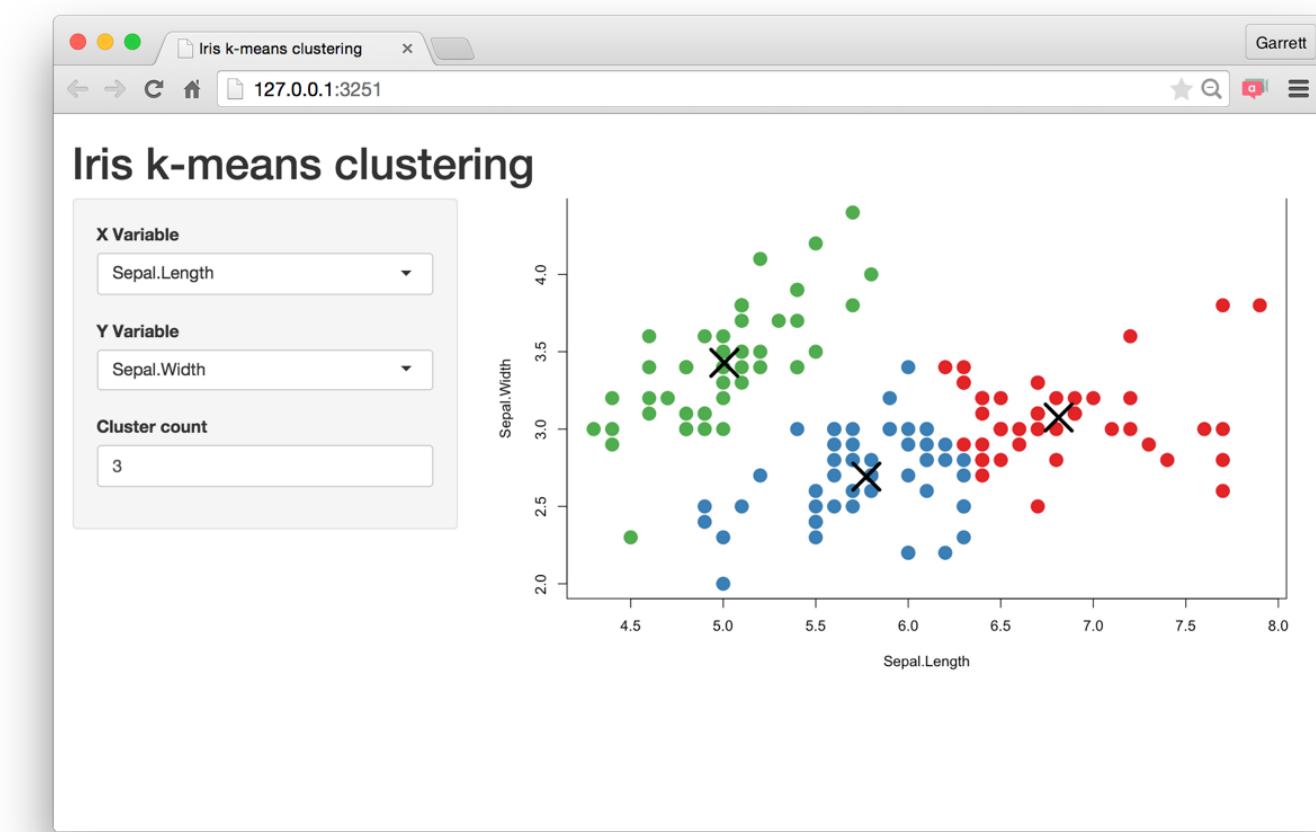
Use the `server` function to assemble inputs into outputs. Follow 3 rules:

1. Save the output that you build to `output$`
 2. Build the output with a `render*` function
 3. Access input values with `input$`
- Create reactivity by using **Inputs** to build **rendered Outputs**

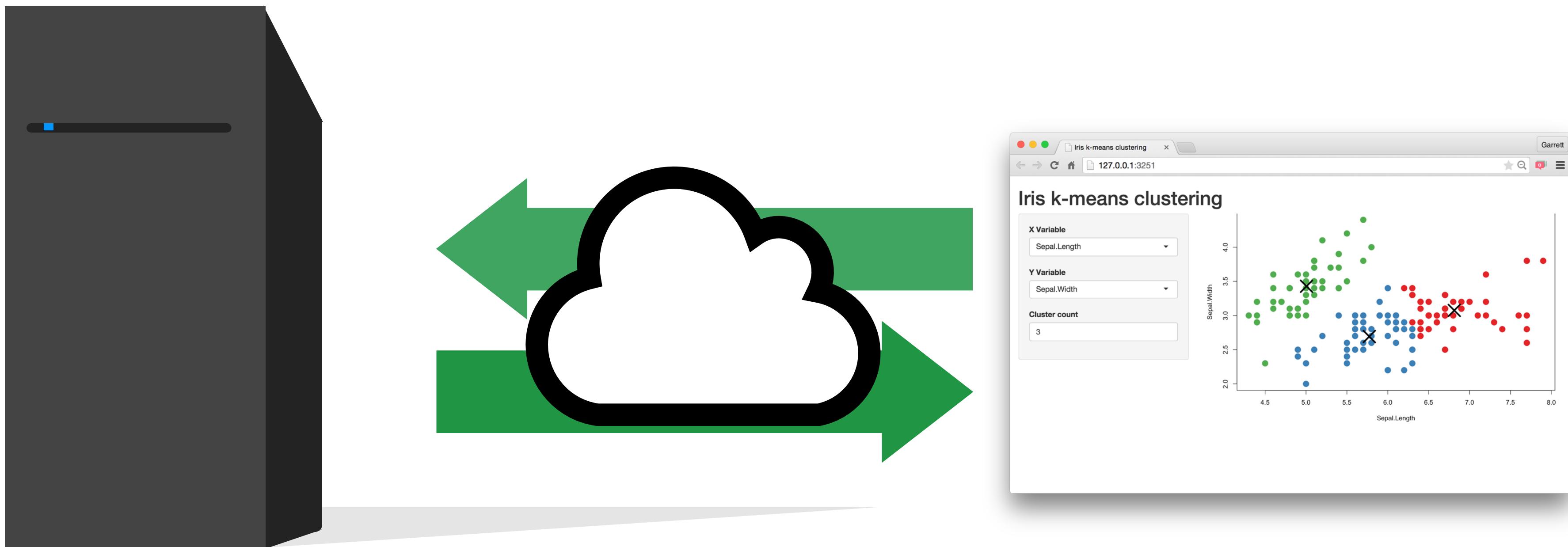
**Share
your app**



Every Shiny app is maintained by a computer running R



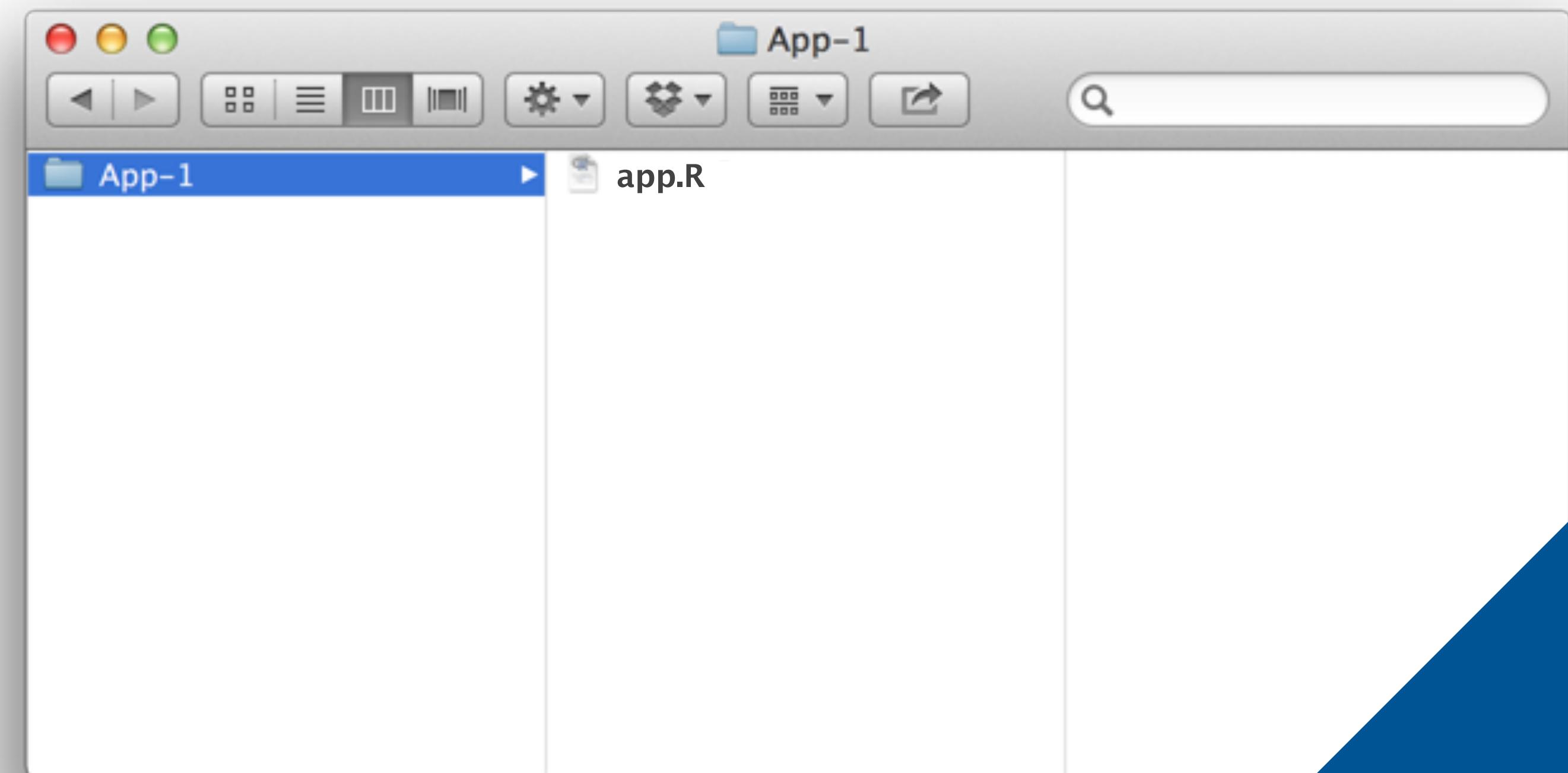
Every Shiny app is maintained by a computer running R



How to save your app

One directory with every file the app needs:

- **app.R** (*your script which ends with a call to shinyApp()*)
- **datasets, images, css, helper scripts, etc.**



You must use this
exact name (app.R)

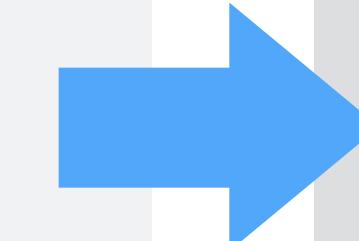
Two file apps

```
library(shiny)

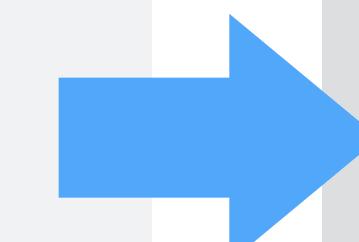
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
```

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

shinyApp(ui = ui, server = server)
```



```
# ui.R
library(shiny)
fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
```

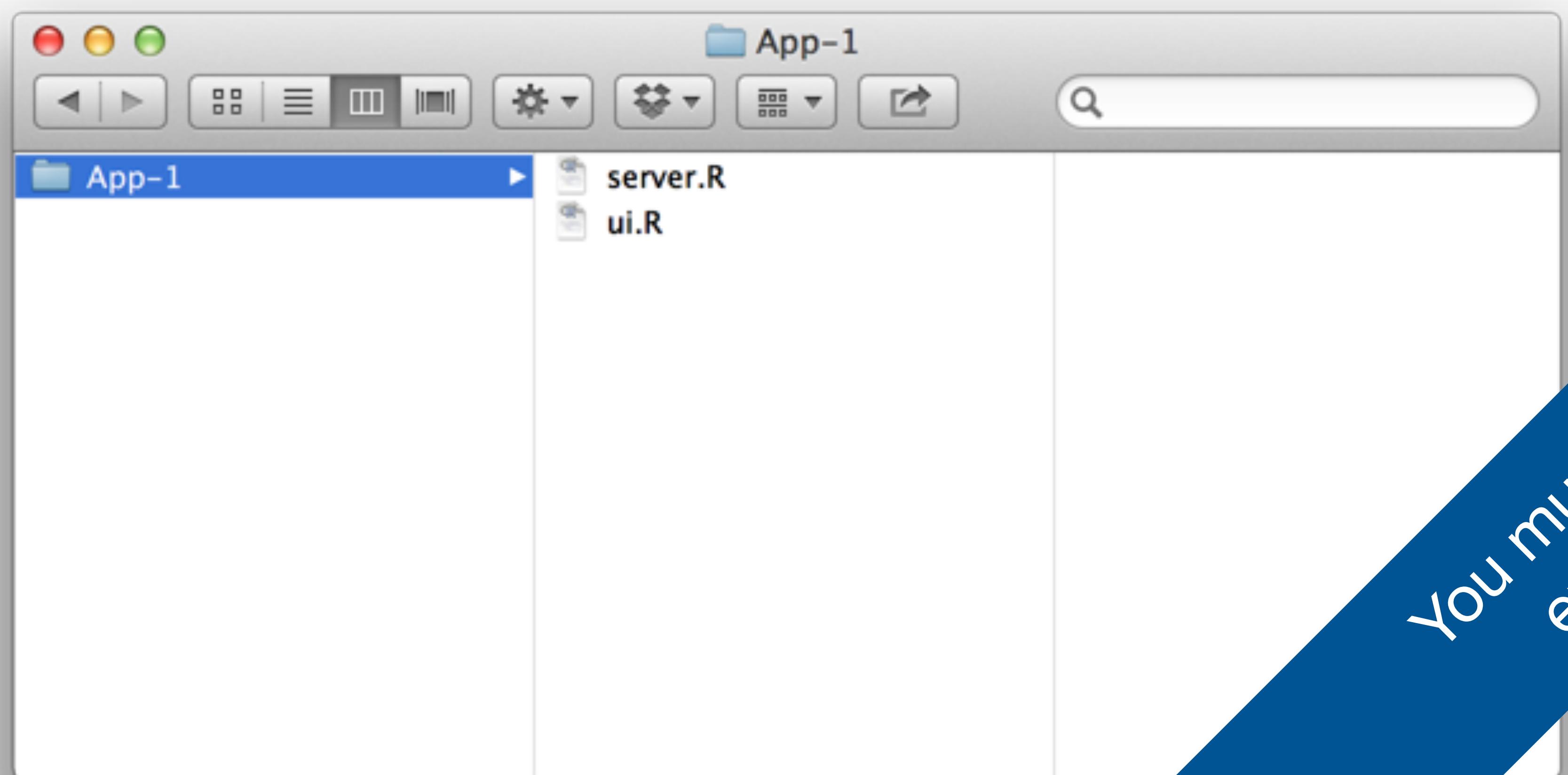


```
# server.R
library(shiny)
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```

Two file apps

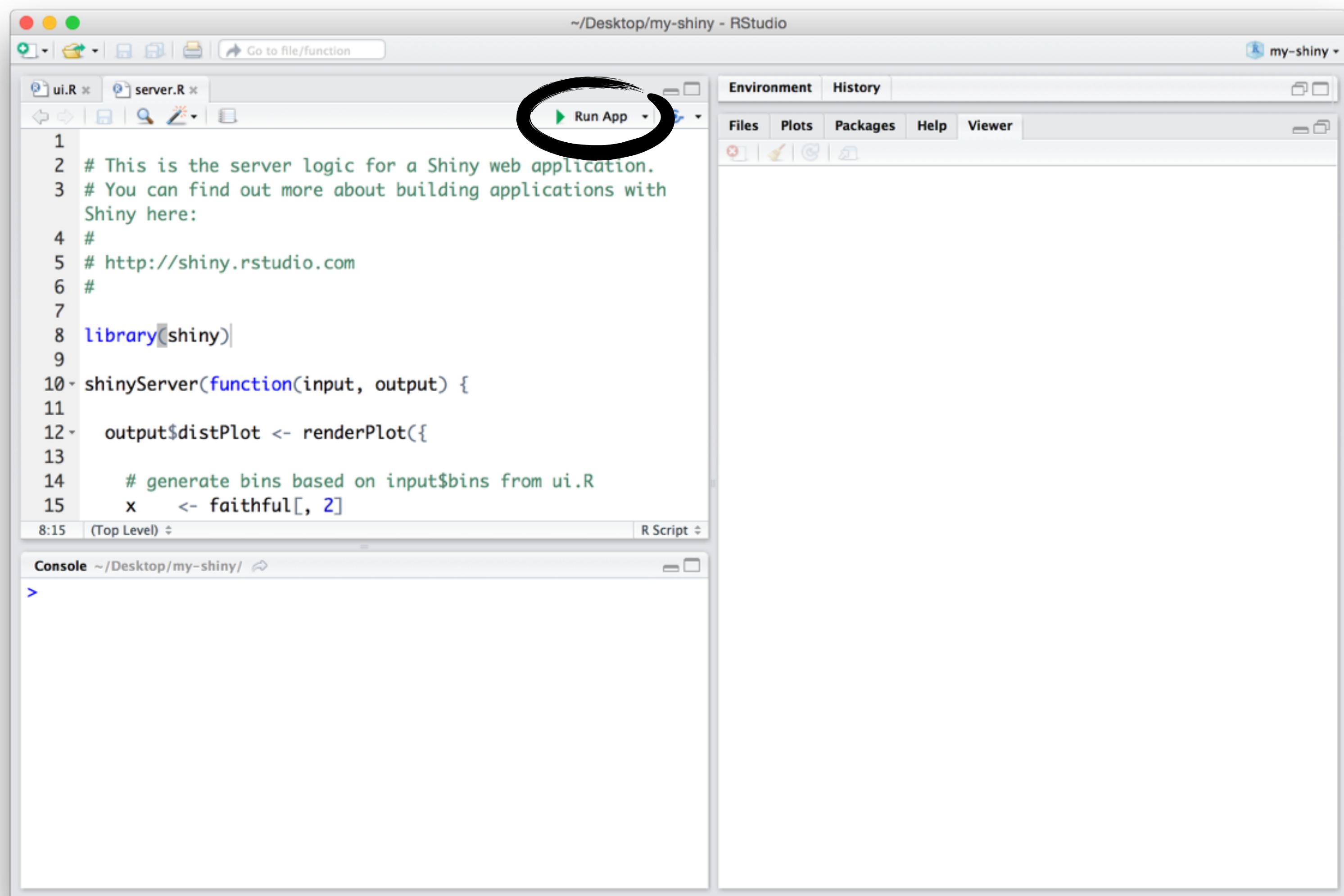
One directory with two files:

- `server.R`
- `ui.R`

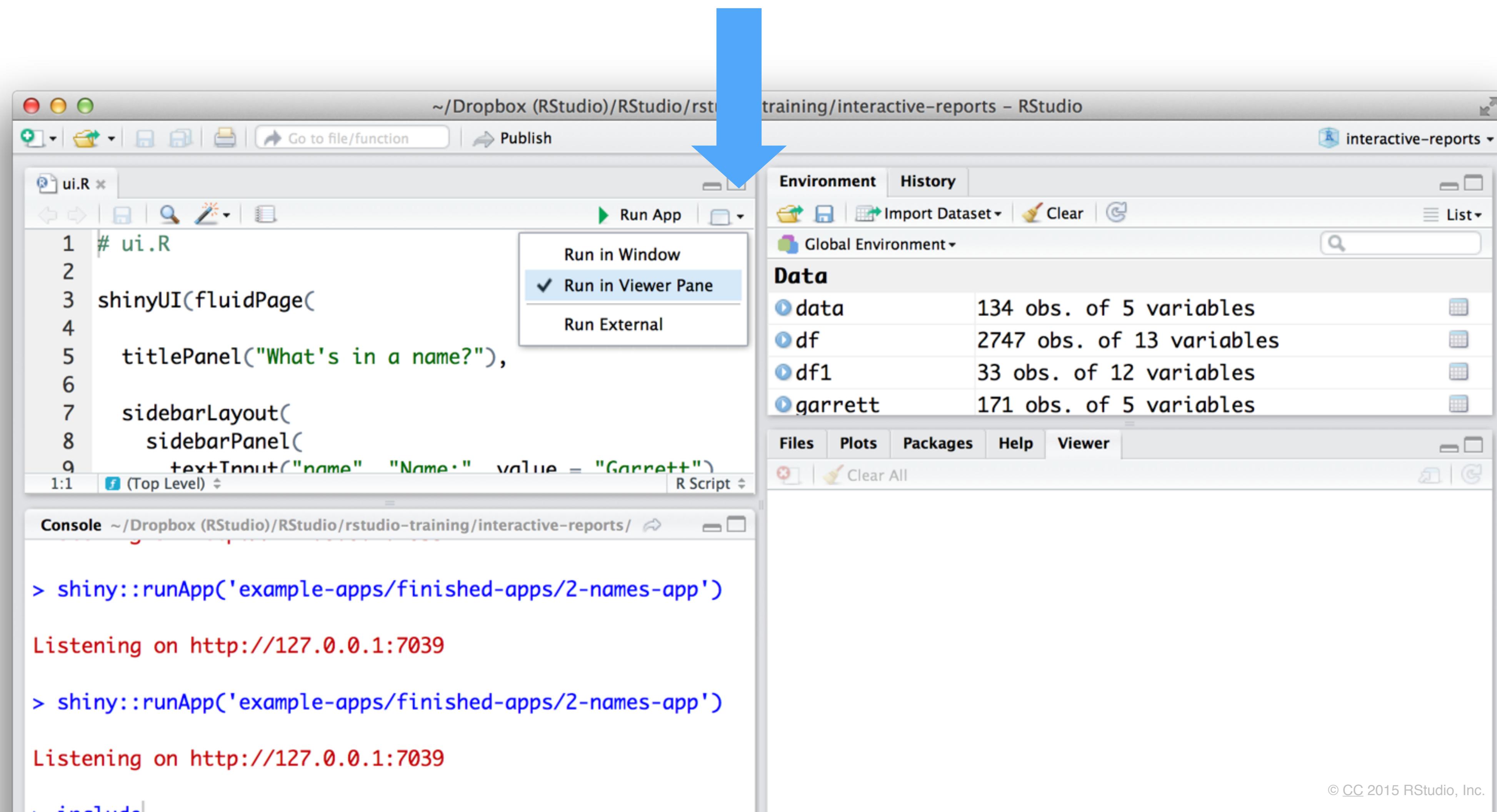


You must use these
exact names

Launch an app



Display options



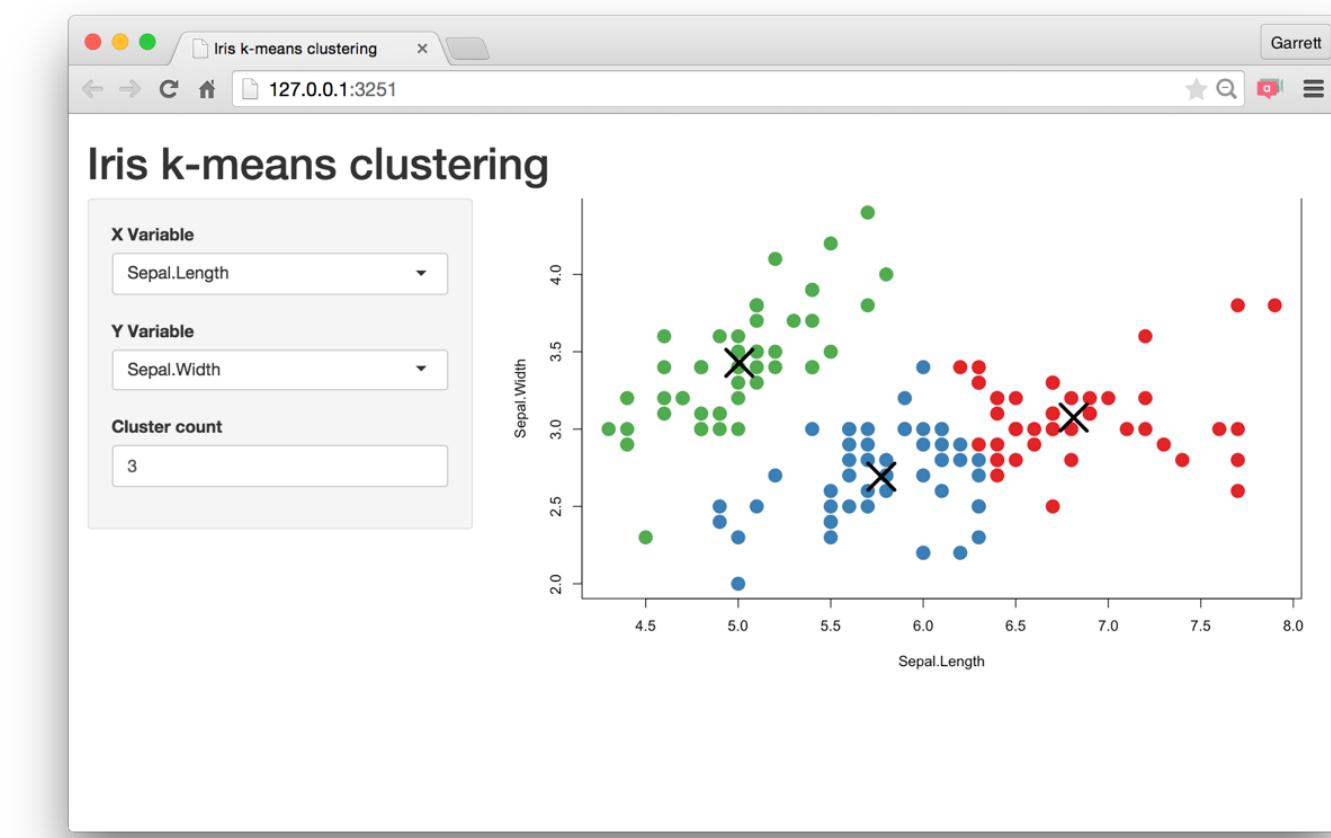
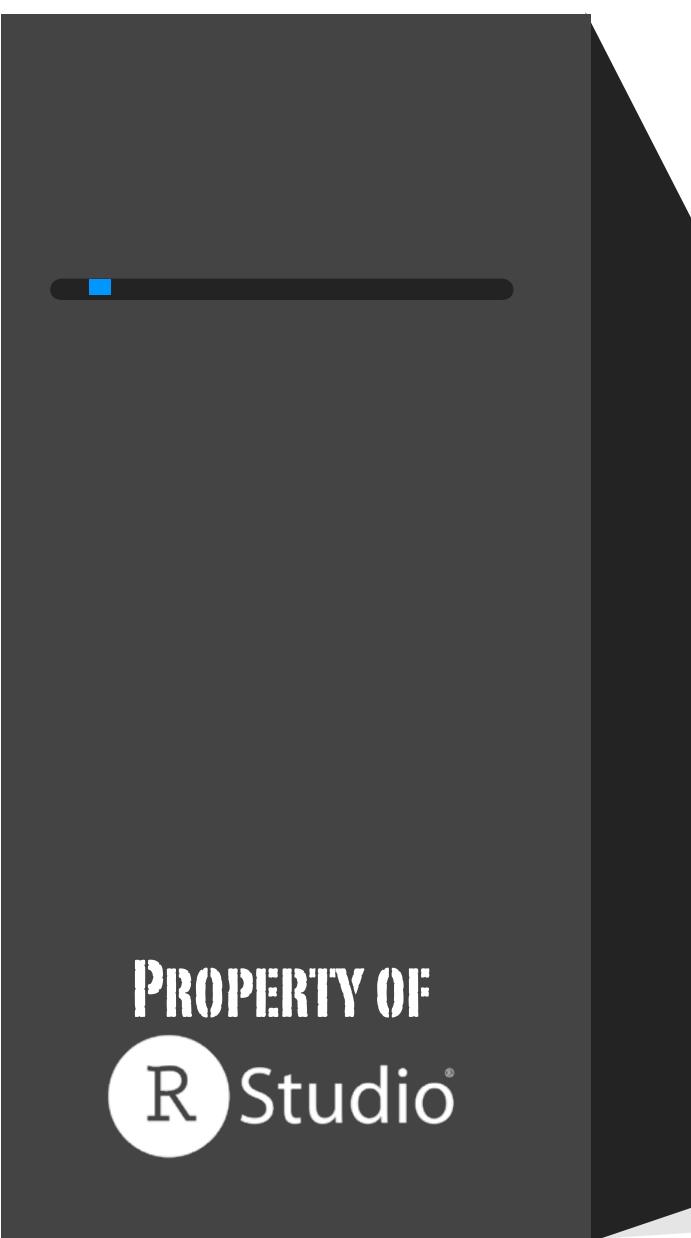
Use
shinyapps.io

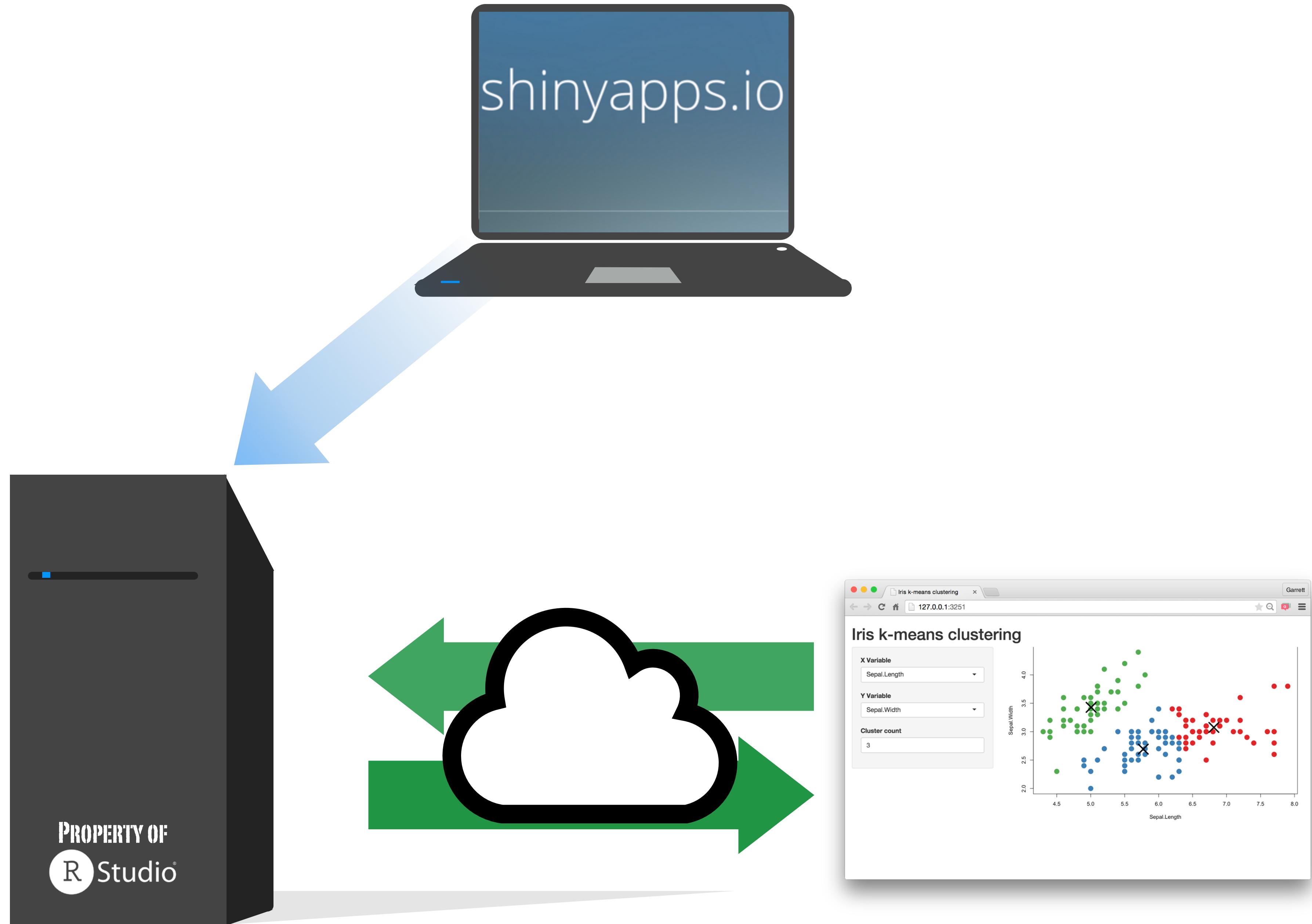


Shinyapps.io

A server maintained by RStudio

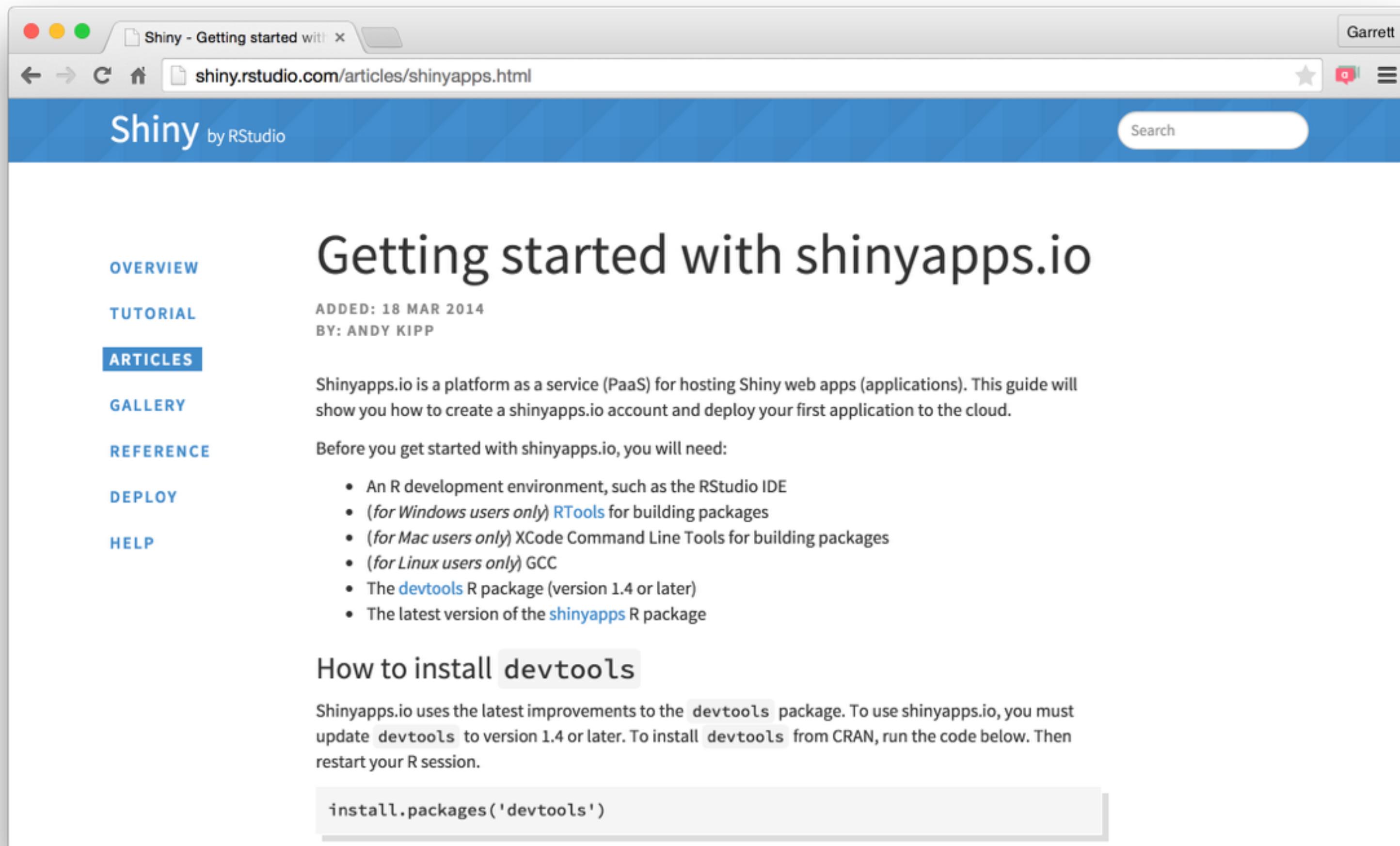
- free
- easy to use
- secure
- scalable





Getting started guide

shiny.rstudio.com/articles/shinyapps.html



The screenshot shows a web browser window with the title bar "Shiny - Getting started with" and the address bar containing the URL "shiny.rstudio.com/articles/shinyapps.html". The browser interface includes standard controls like back, forward, and search, along with a user profile "Garrett". The main content area is titled "Getting started with shinyapps.io". On the left, a sidebar menu lists "OVERVIEW", "TUTORIAL", "ARTICLES" (which is selected), "GALLERY", "REFERENCE", "DEPLOY", and "HELP". The main content area includes the date "ADDED: 18 MAR 2014" and author "BY: ANDY KIPP". It describes Shinyapps.io as a PaaS for hosting Shiny web apps and provides instructions for getting started, listing requirements such as an R development environment, RTools, XCode Command Line Tools, GCC, devtools, and shinyapps packages. It also includes a section on how to install devtools and a code snippet for doing so.

Getting started with shinyapps.io

ADDED: 18 MAR 2014
BY: ANDY KIPP

Shinyapps.io is a platform as a service (PaaS) for hosting Shiny web apps (applications). This guide will show you how to create a shinyapps.io account and deploy your first application to the cloud.

Before you get started with shinyapps.io, you will need:

- An R development environment, such as the RStudio IDE
- (for Windows users only) [RTools](#) for building packages
- (for Mac users only) XCode Command Line Tools for building packages
- (for Linux users only) GCC
- The [devtools](#) R package (version 1.4 or later)
- The latest version of the [shinyapps](#) R package

How to install `devtools`

Shinyapps.io uses the latest improvements to the `devtools` package. To use shinyapps.io, you must update `devtools` to version 1.4 or later. To install `devtools` from CRAN, run the code below. Then restart your R session.

```
install.packages('devtools')
```

FREE**\$ 0** /month

New to Shiny? Deploy your applications to the cloud for FREE. Perfect for teachers and students or those who want a place to learn and play. No credit card required.

5 Applications**25 Active Hours** Community Support RStudio Branding**BASIC****\$ 39** /month
(or \$440/year)

Take your users' experience to the next level. shinyapps.io Basic lets you scale your application performance by adding R processes dynamically as usage increases.

Unlimited Applications**250 Active Hours** Multiple Instances Email Support**STANDARD****\$ 99** /month
(or \$1,100/year)

Need password protection? shinyapps.io Standard lets you authenticate your application users.

Unlimited Applications**1000 Active Hours** Authentication Multiple Instances Email Support**PROFESSIONAL****\$ 299** /month
(or \$3,300/year)

shinyapps.io Professional has it all. Share an account with others in your business or change your shinyapps.io domain into a URL of your own.

Unlimited Applications**5000 Active Hours** Authentication Multiple Users Multiple Instances Custom Domains* Email Support

**Build
your own
server**



Shiny Server

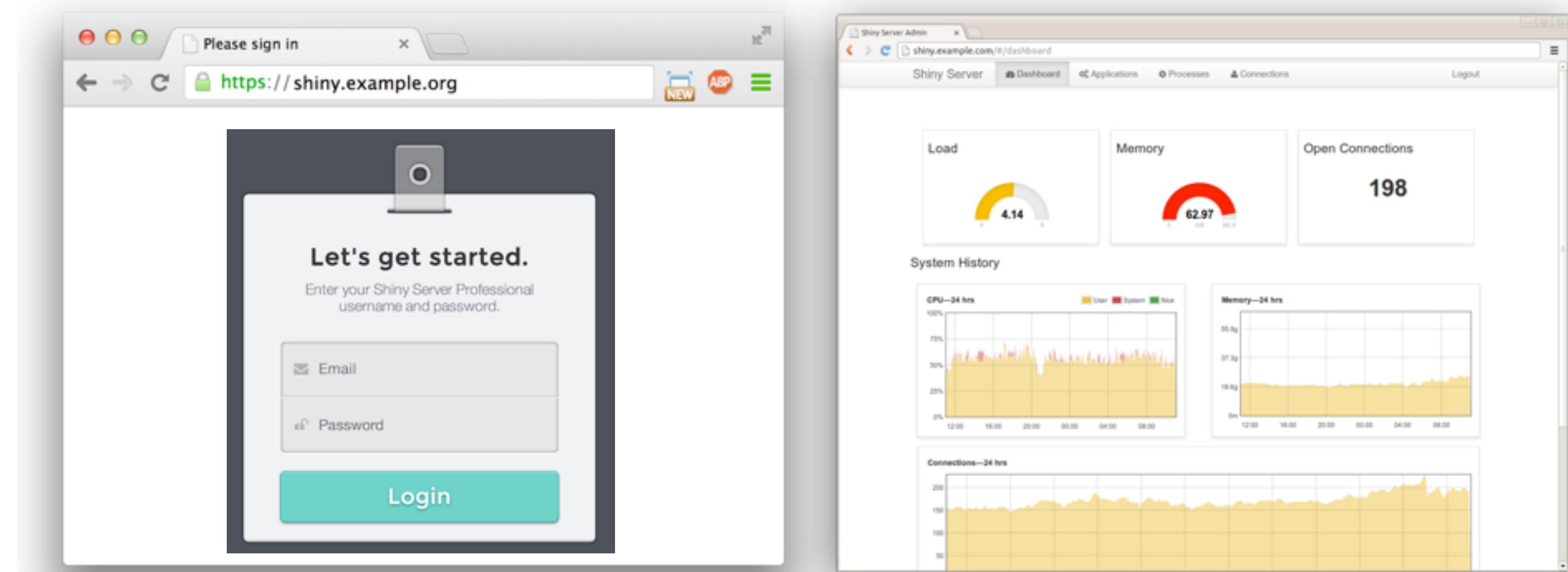
www.rstudio.com/products/shiny/shiny-server/

A back end program that builds a linux web server specifically designed to host Shiny apps.

Shiny Server Pro

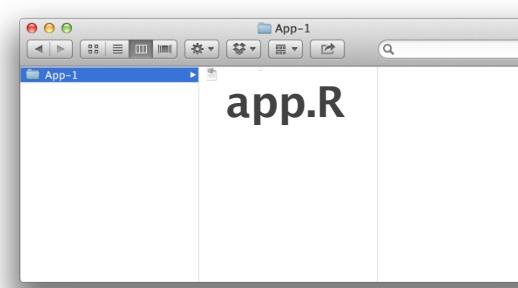
www.rstudio.com/products/shiny/shiny-server/

- ✓ **Secure access** - LDAP, GoogleAuth, SSL, and more
- ✓ **Performance** - fine tune at app and server level
- ✓ **Management** - monitor and control resource use
- ✓ **Support** - direct priority support



45 day
evaluation
free trial

Recap: Sharing



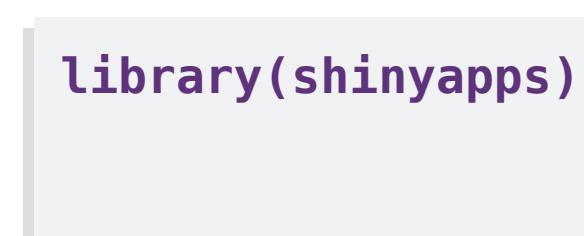
Save your app in its own directory as
app.R, or **ui.R** and **server.R**



Host apps at **shinyapps.io** by:



1. Sign up for a free **shinyapps.io** account



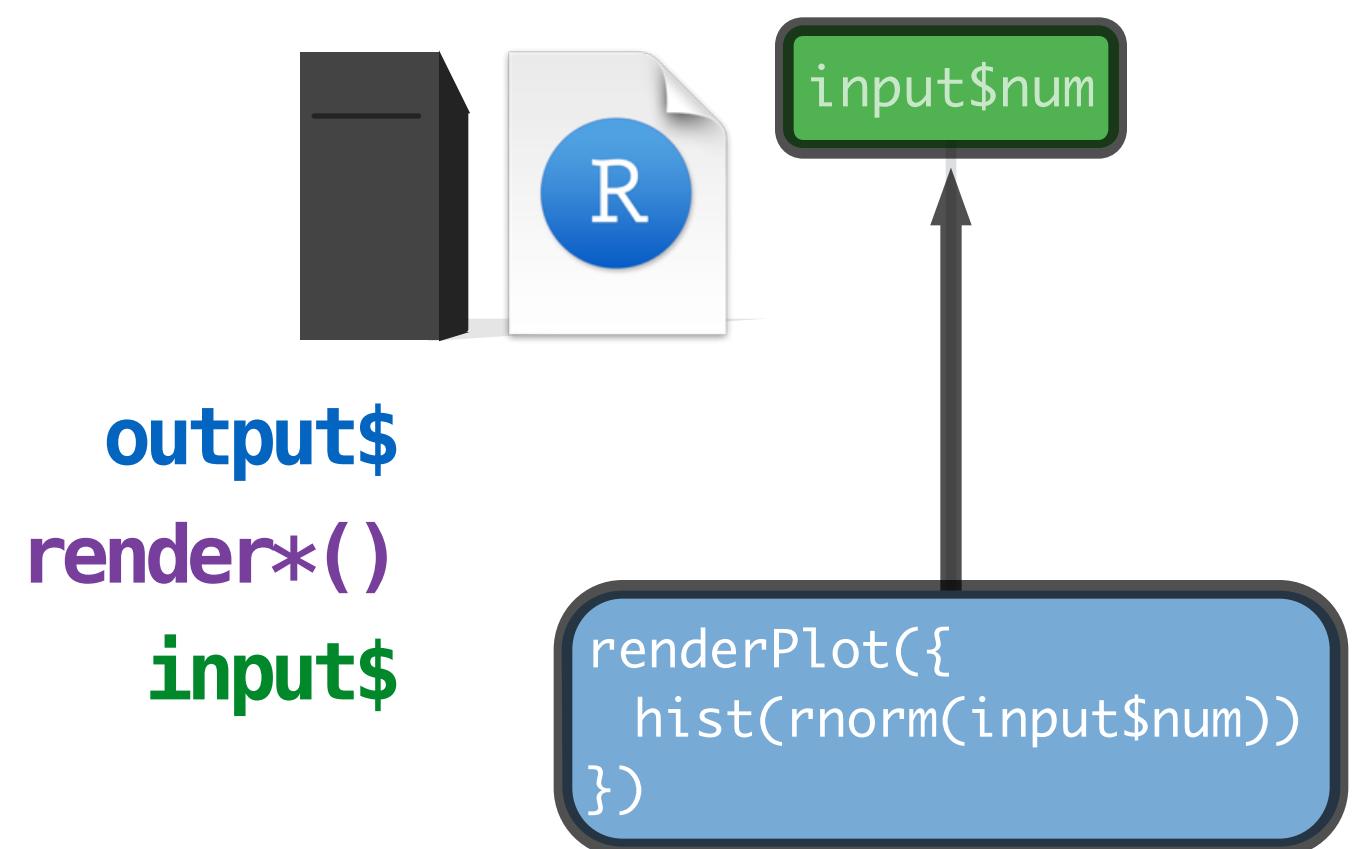
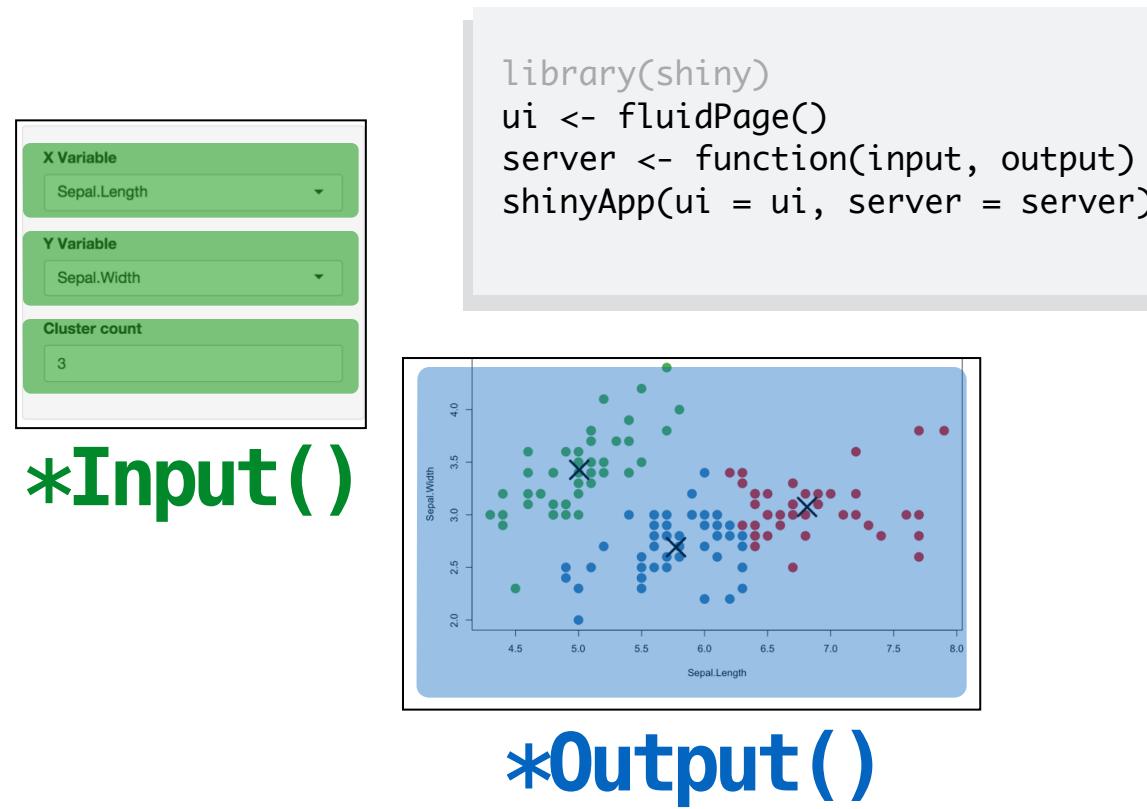
2. Install the **shinyapps** package



Build your own server with **Shiny Server**
or **Shiny Server Pro**

Learn
more

You now how to



Build an app

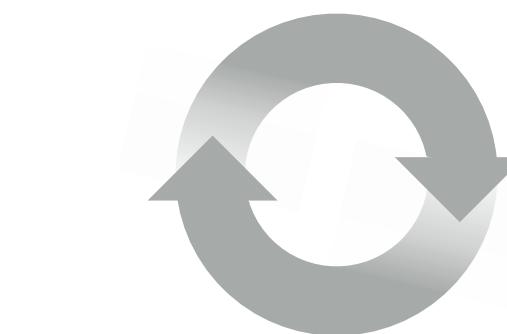
Create interactions

Share your apps

How to start with Shiny

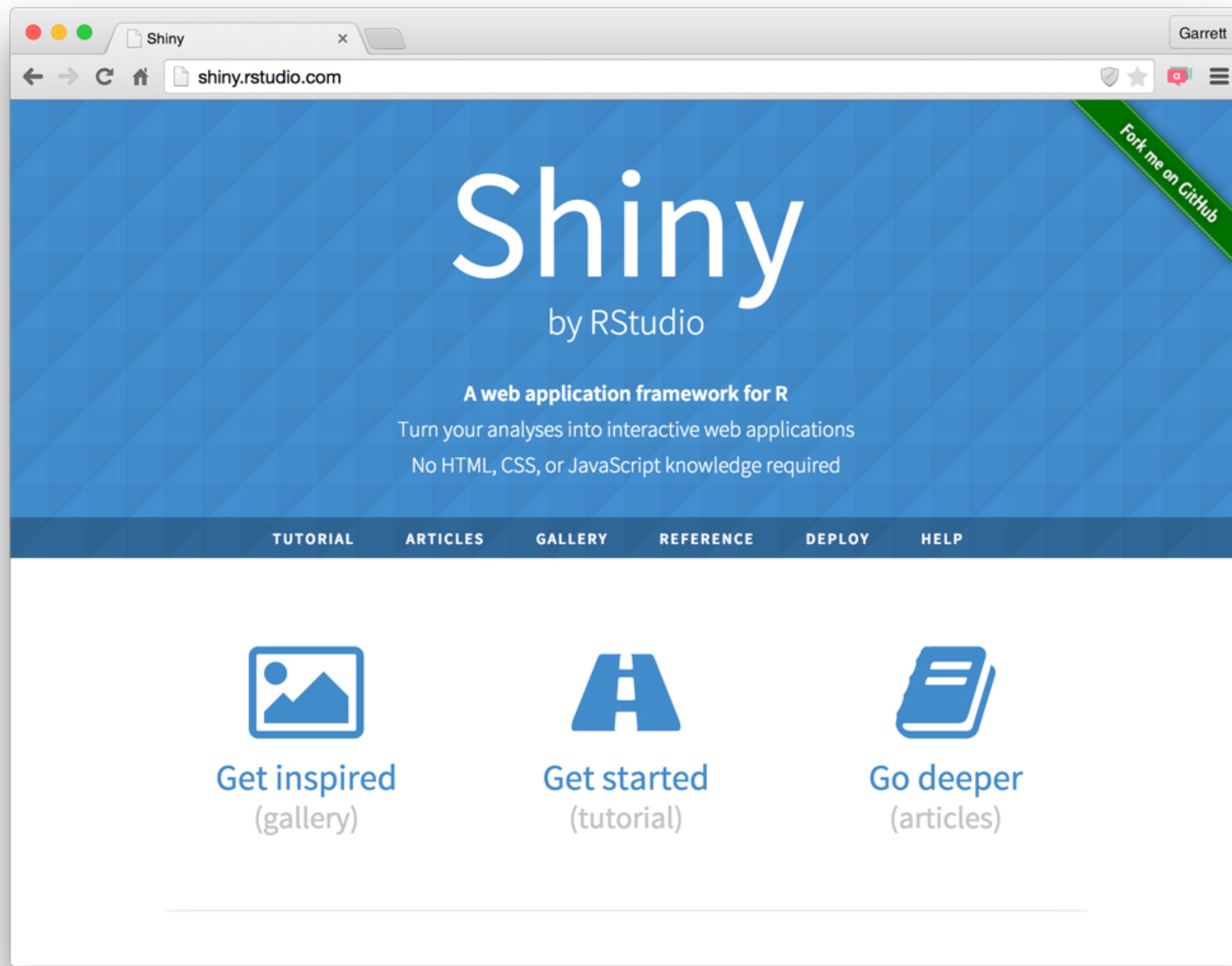


1. How to build a Shiny app (Today)
2. How to customize reactions (May 27)
3. How to customize appearance (June 3)



The Shiny Development Center

shiny.rstudio.com

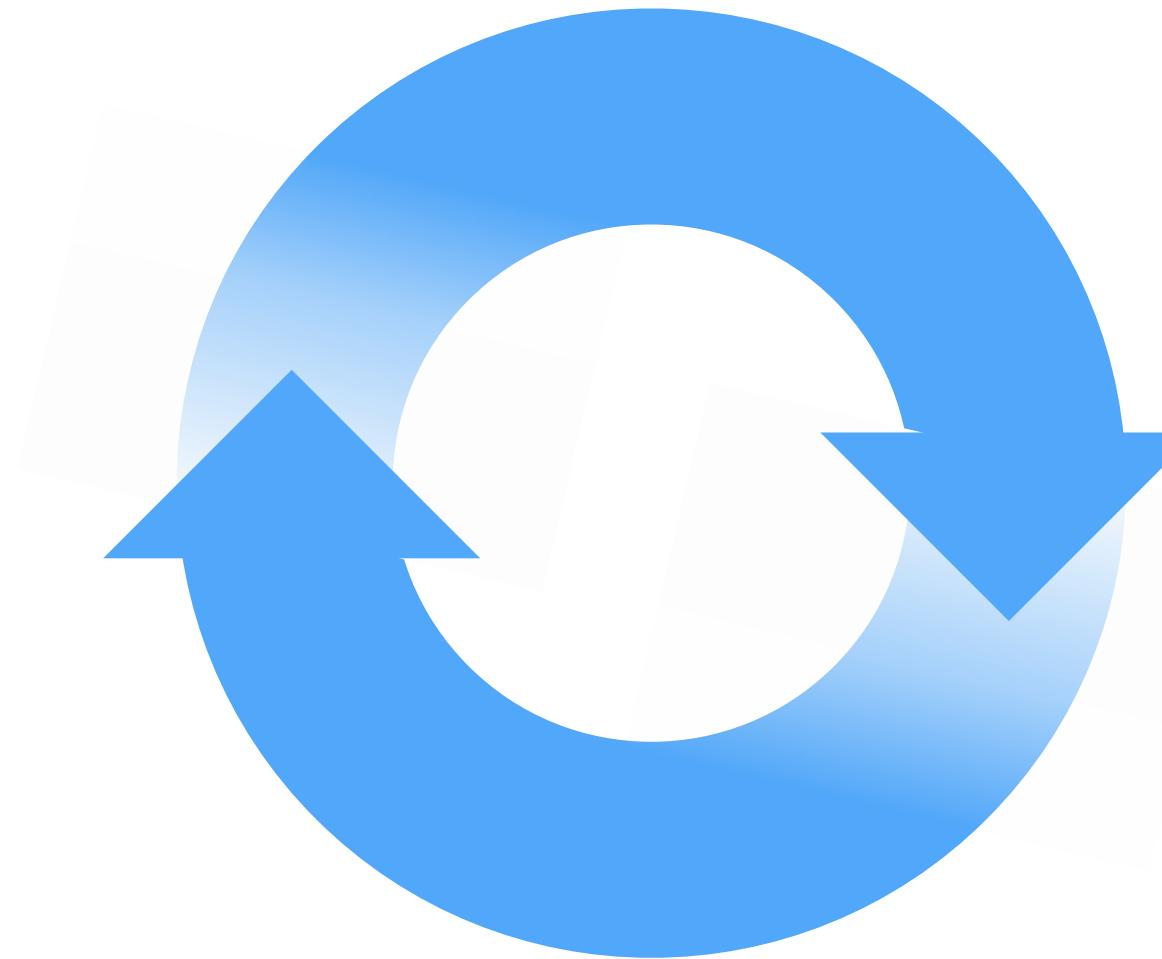


All Training materials are provided "as is" and without warranty and RStudio disclaims any and all express and implied warranties including without limitation the implied warranties of title, fitness for a particular purpose, merchantability and noninfringement.

The Training Materials are licensed under the Creative Commons Attribution-Noncommercial 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

How to start with Shiny, Part 2

How to customize reactions



Garrett Grolemund

Data Scientist and Master Instructor

May 2015

Email: garrett@rstudio.com

Twitter: @StatGarrett

Code and slides at:

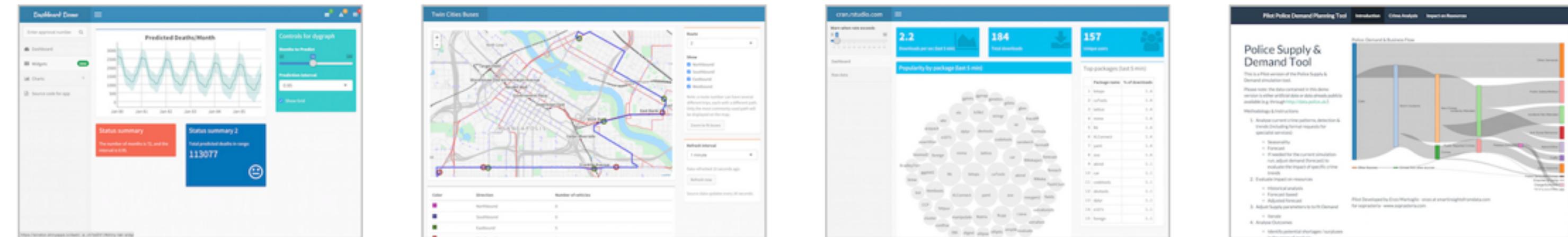
bit.ly/shiny-quickstart-2



Shiny Showcase

www.rstudio.com/products/shiny/shiny-user-showcase/

Shiny Apps for the Enterprise



[Shiny Dashboard Demo](#)

A dashboard built with Shiny.

[Location tracker](#)

Track locations over time with streaming data.

[Download monitor](#)

Streaming download rates visualized as a bubble chart.

[Supply and Demand](#)

Forecast demand to plan resource allocation.

Industry Specific Shiny Apps



[Economic Dashboard](#)

Economic forecasting with macroeconomic indicators.

[ER Optimization](#)

An app that models patient flow.

[CDC Disease Monitor](#)

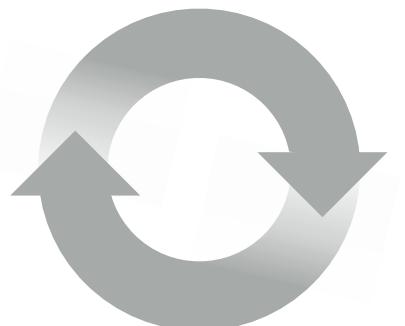
Alert thresholds and automatic weekly updates.

[Ebola Model](#)

An epidemiological simulation.



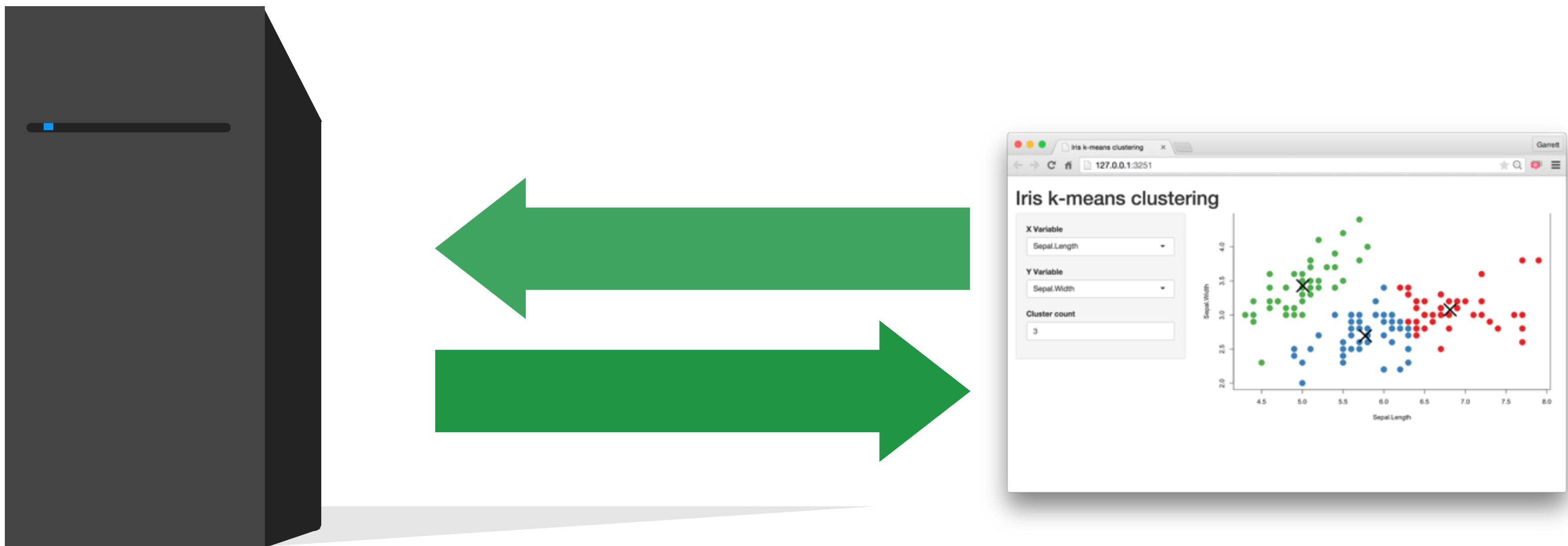
How to start with Shiny



1. How to build a Shiny app (www.rstudio.com/resources/webinars/)
2. How to customize reactions (Today)
3. How to customize appearance (June 17)

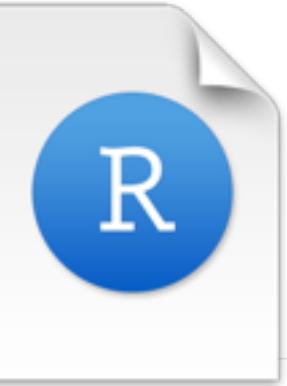
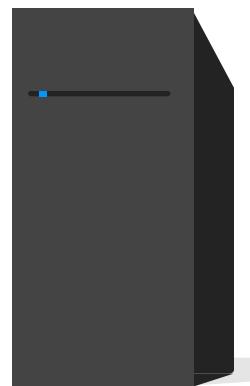
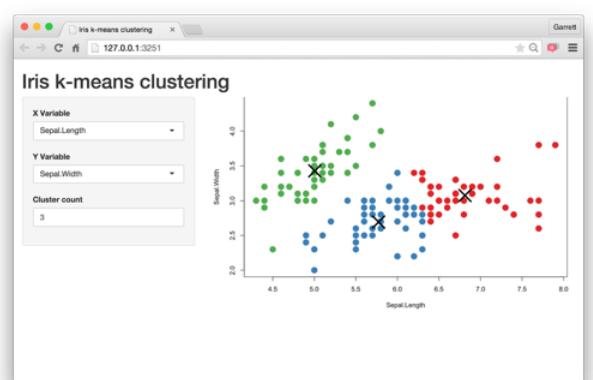
**The story
so far**

Every Shiny app is maintained by a computer running R



App template

The shortest viable shiny app



```
library(shiny)
```

```
ui <- fluidPage()
```

```
server <- function(input, output) {}
```

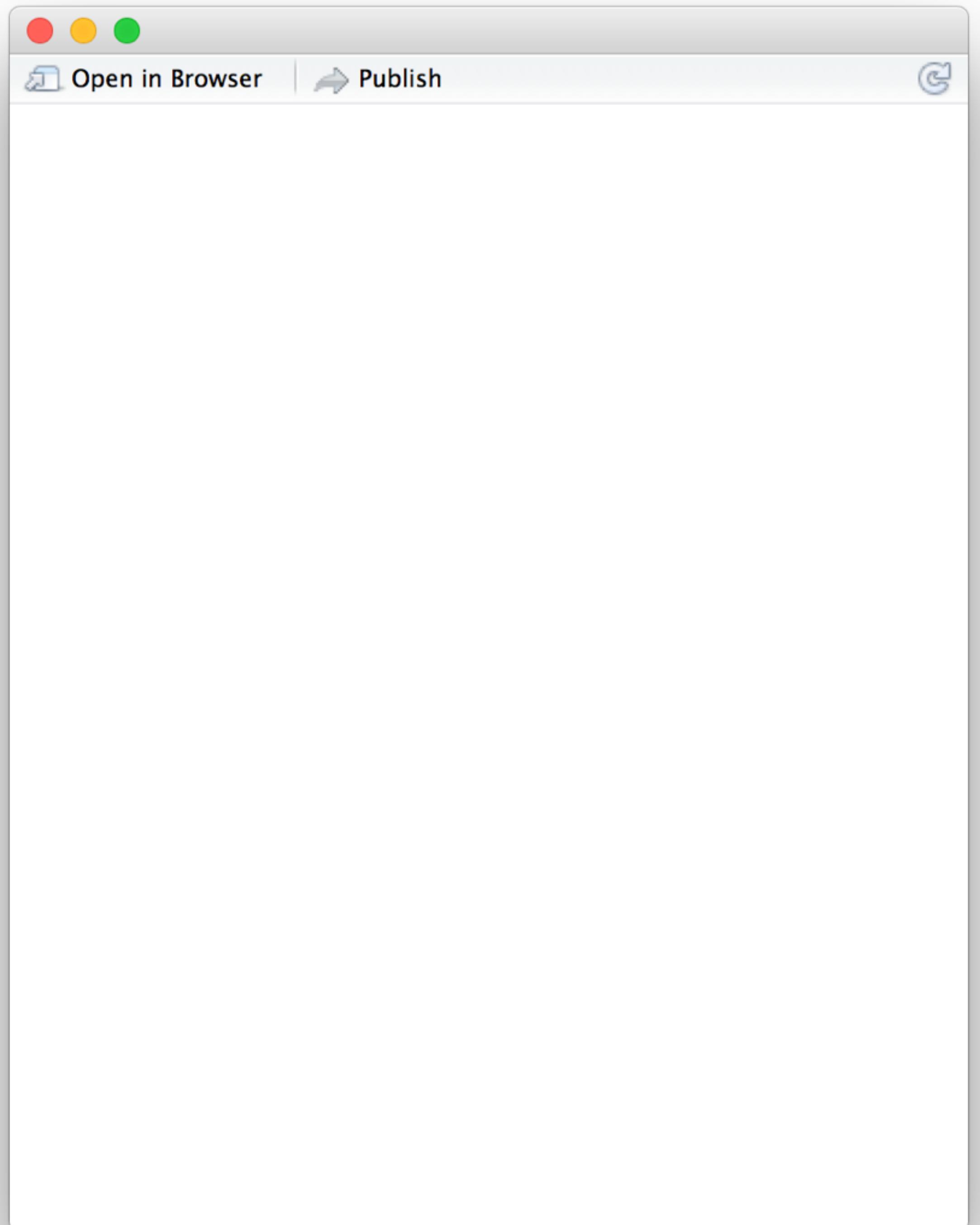
```
shinyApp(ui = ui, server = server)
```

```
library(shiny)

ui <- fluidPage(
  )

server <- function(input, output) {
  }

shinyApp(ui = ui, server = server)
```



```
library(shiny)

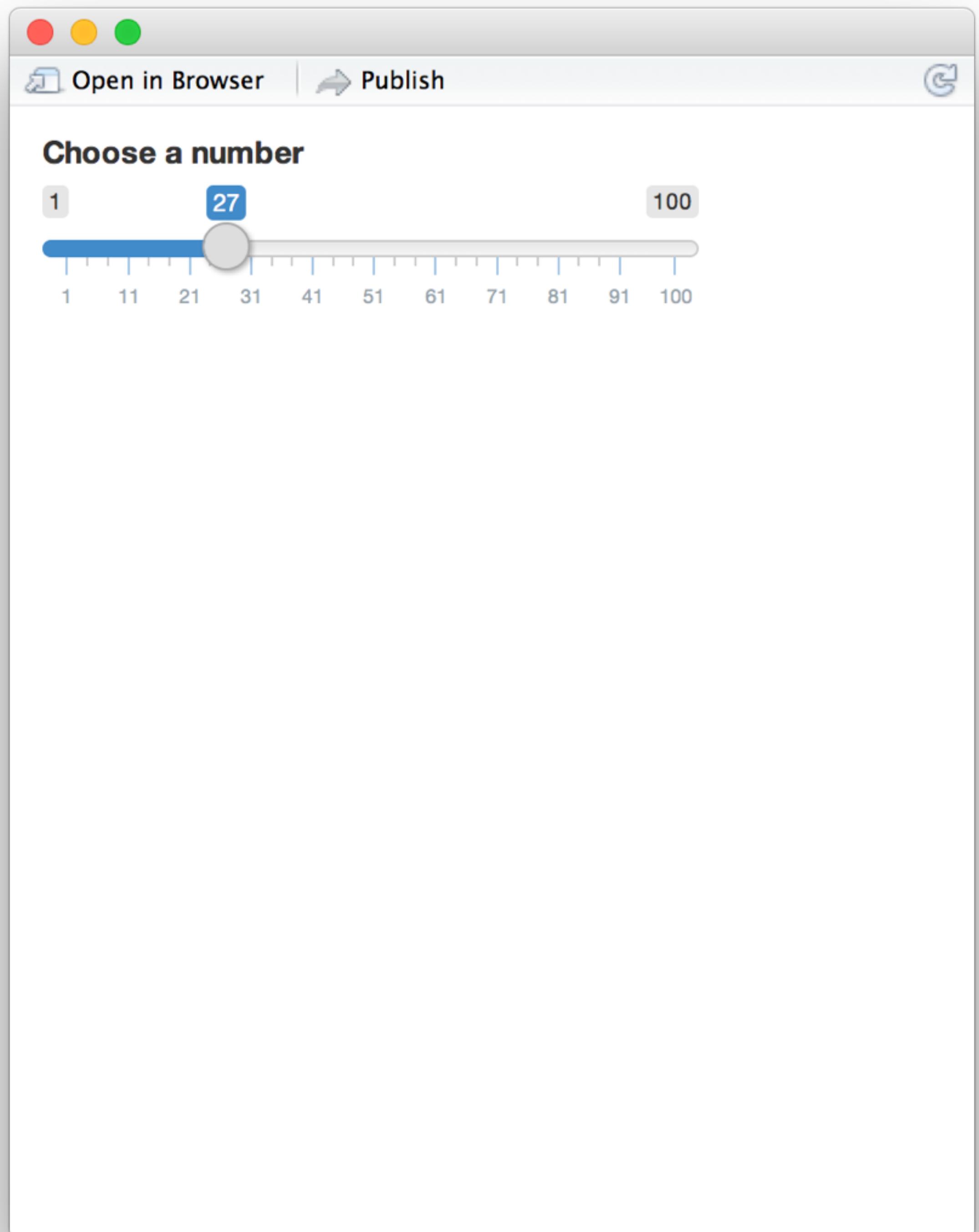
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100)

)

server <- function(input, output) {

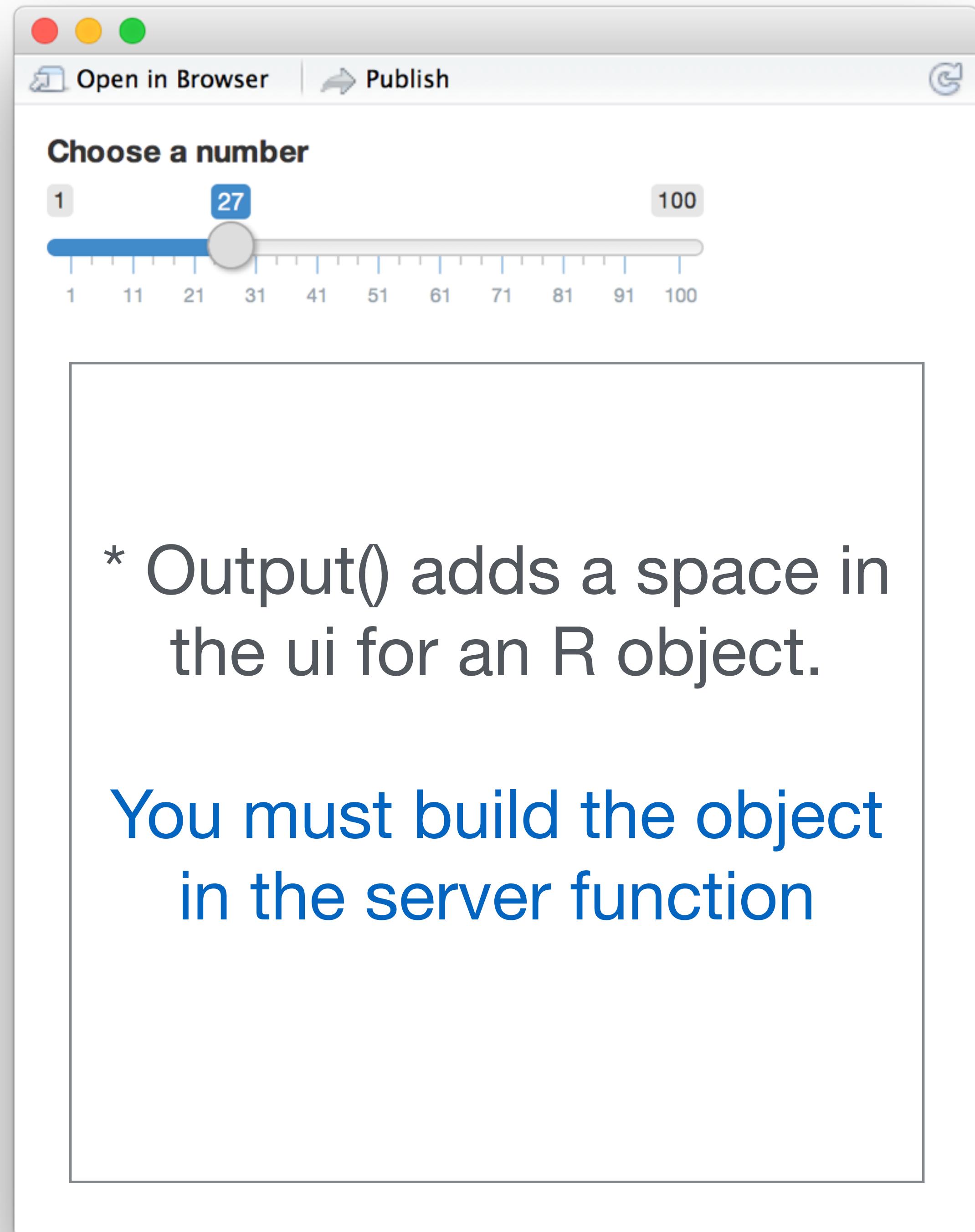
}

shinyApp(ui = ui, server = server)
```



```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
server <- function(input, output) {
}
shinyApp(ui = ui, server = server)
```



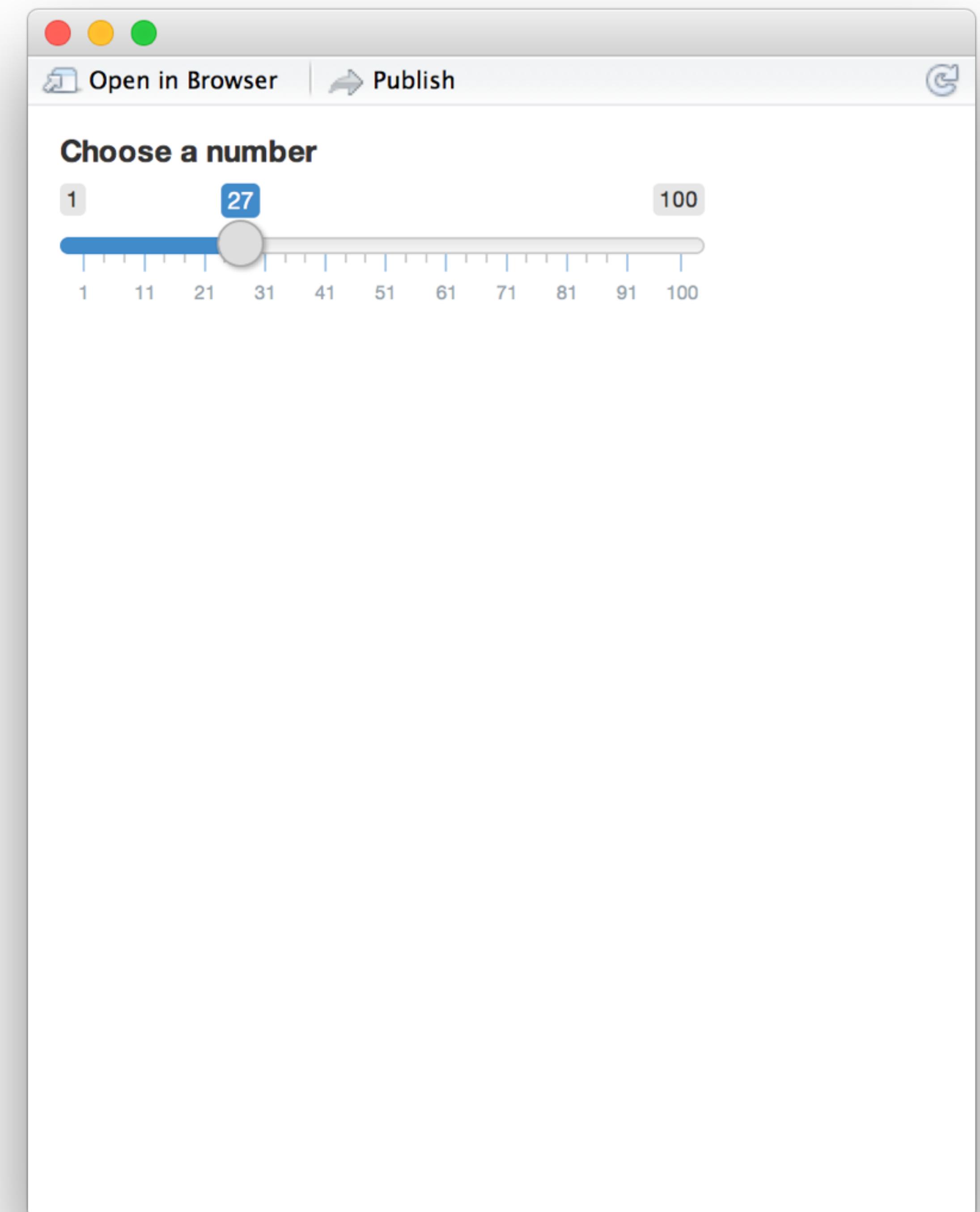
```
library(shiny)

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

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

shinyApp(ui = ui, server = server)
```

1



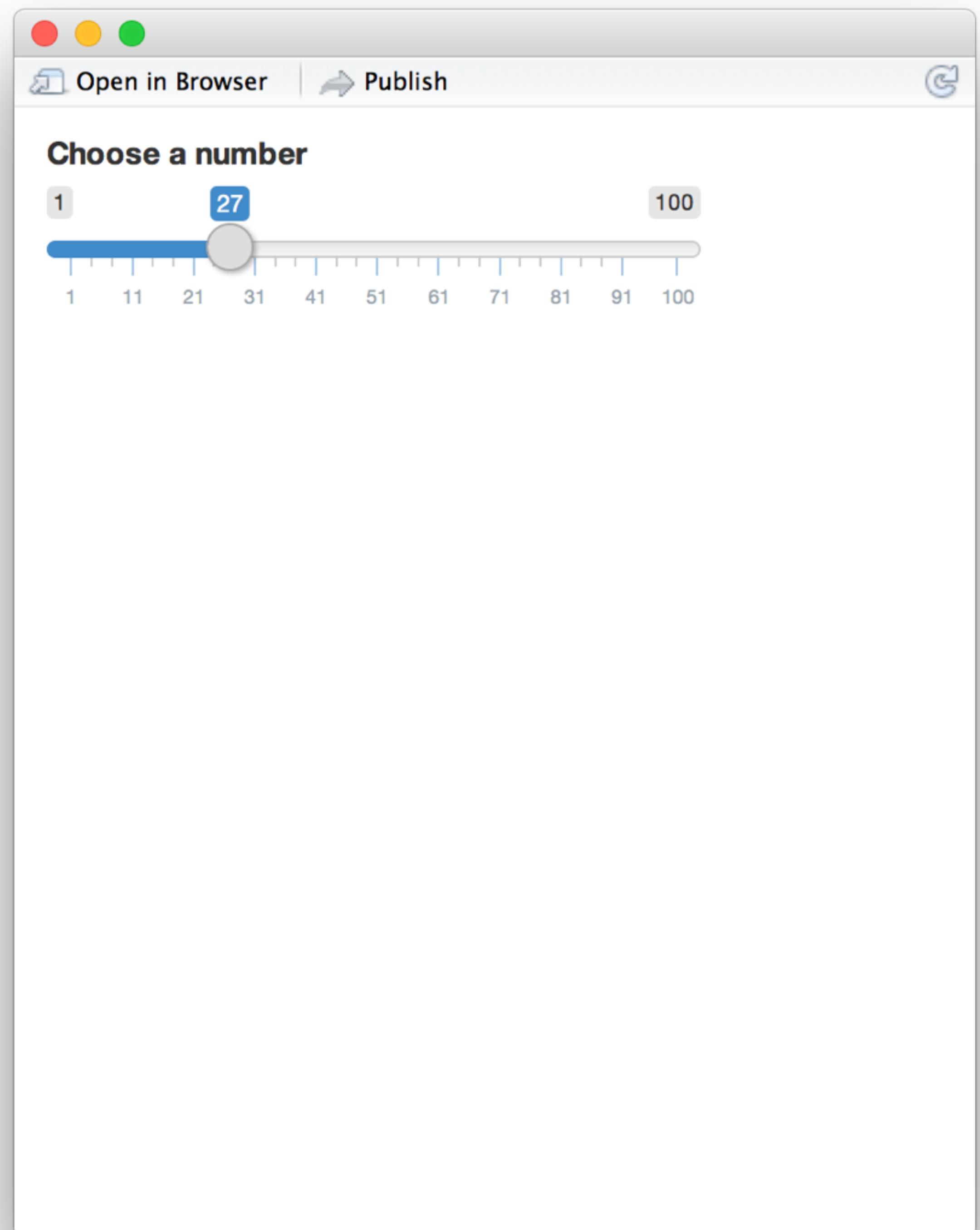
```
library(shiny)

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

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

shinyApp(ui = ui, server = server)
```

2



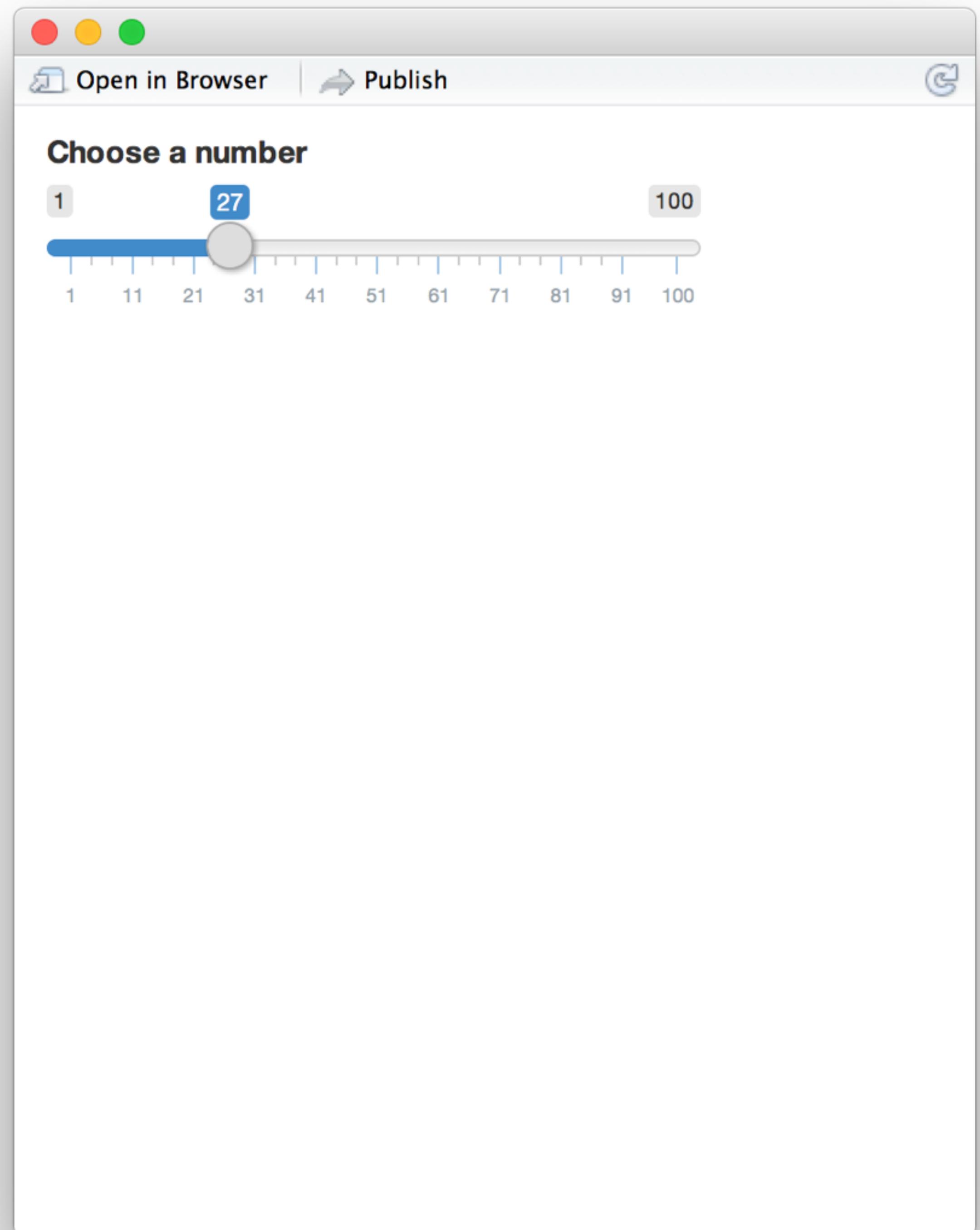
```
library(shiny)

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

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

shinyApp(ui = ui, server = server)
```

3

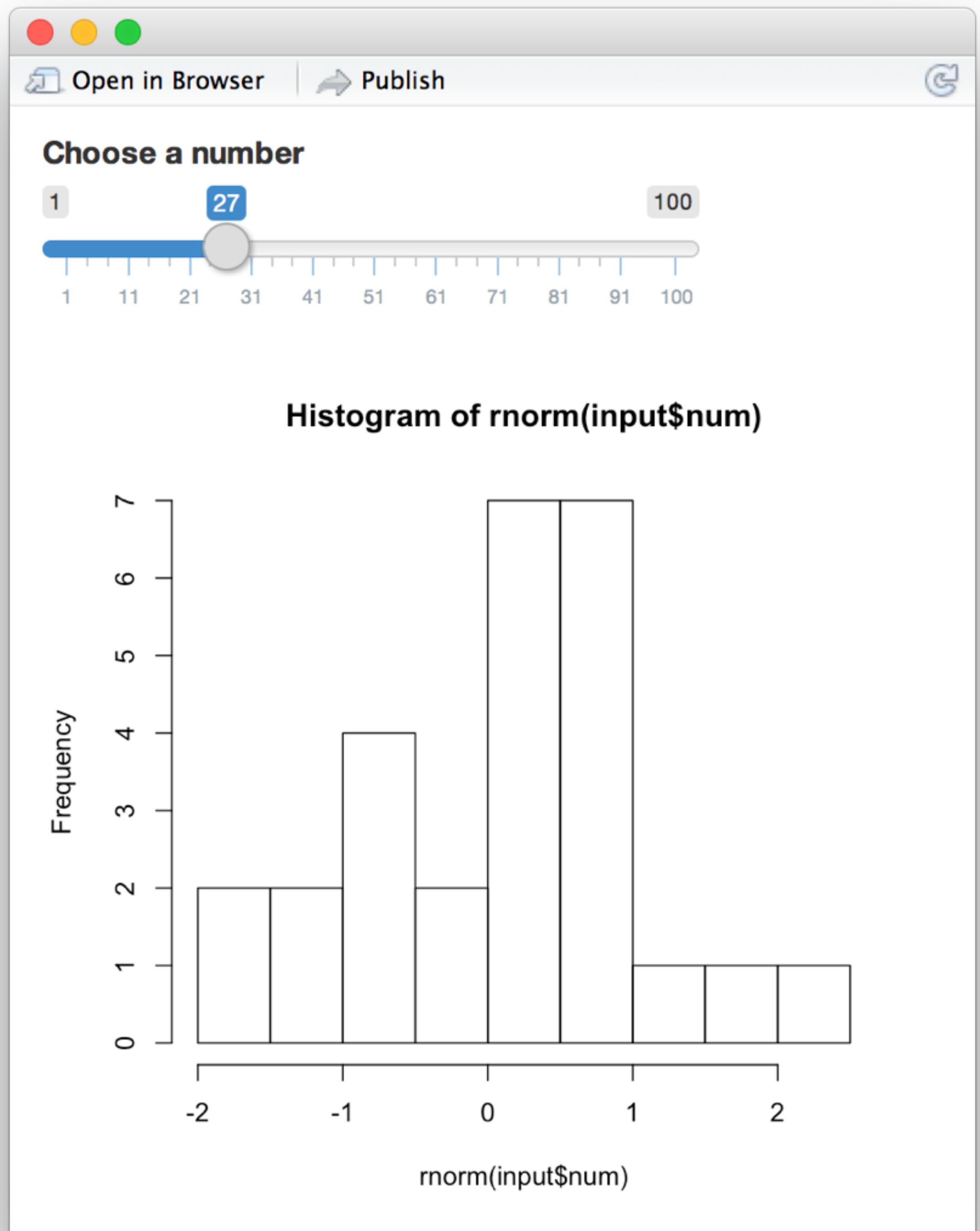


```
library(shiny)

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

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

shinyApp(ui = ui, server = server)
```



Sharing apps



Shiny Server (Pro)

<http://www.rstudio.com/resources/webinars/>

**what is
Reactivity?**

Think Excel.

The screenshot shows a Microsoft Excel application window titled "Workbook1". The ribbon menu is visible at the top, featuring tabs for Home, Layout, Tables, Charts, SmartArt, Formulas, Data, and Review. The Home tab is selected, displaying various tools for editing, font, alignment, number formats, and styles. A blank worksheet is open, with the active cell being A1. The column headers (A through Q) and row headers (1 through 25) are visible along the left and top edges respectively. The status bar at the bottom indicates "Normal View" and "Ready".

Workbook1

Tables Charts SmartArt Formulas Data Review

Font Alignment Number Format Cells

Body (Body) 12 A A abc Wrap Text General % , .00 .00 Conditional Formatting Styles Insert Delete

I U Merge

C D E F G H I J K L M N O

50				= F4 + 1
----	--	--	--	----------

© CC 2015 RStudio, Inc.

Workbook1

Tables Charts SmartArt Formulas Data Review

Font Alignment Number Format Cells

Body (Body) 12 A A abc Wrap Text General % , .00 .00 Conditional Formatting Styles Insert Delete

I U A Merge

C D E F G H I J K L M N O

50 51

© CC 2015 RStudio, Inc.

Workbook1

Tables Charts SmartArt Formulas Data Review

Font Alignment Number Format Cells

Body (Body) 12 A A abc Wrap Text General % , .00 .00 Conditional Formatting Styles Insert Delete

I U Merge

C D E F G H I J K L M N O

100 101

Workbook1

Tables Charts SmartArt Formulas Data Review

Font Alignment Number Format Cells

Body (Body) 12 A A abc Wrap Text General % , .00 .00 Conditional Formatting Styles Insert Delete

I U Merge

C D E F G H I J K L M N O

999 1000

Workbook1

Tables Charts SmartArt Formulas Data Review

Font Alignment Number Format Cells

1000 → 1001

C D E F G H I J K L M N O

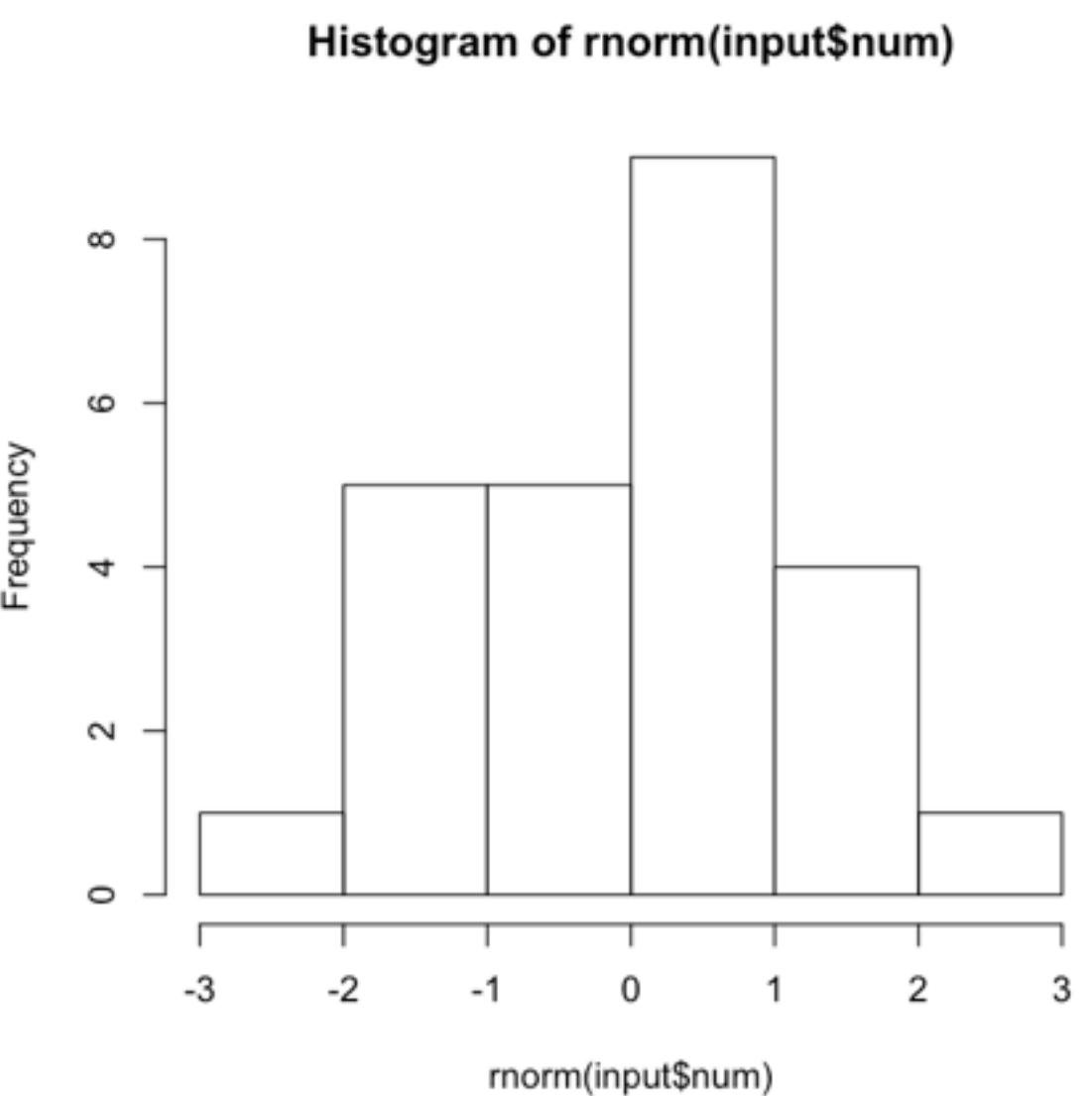
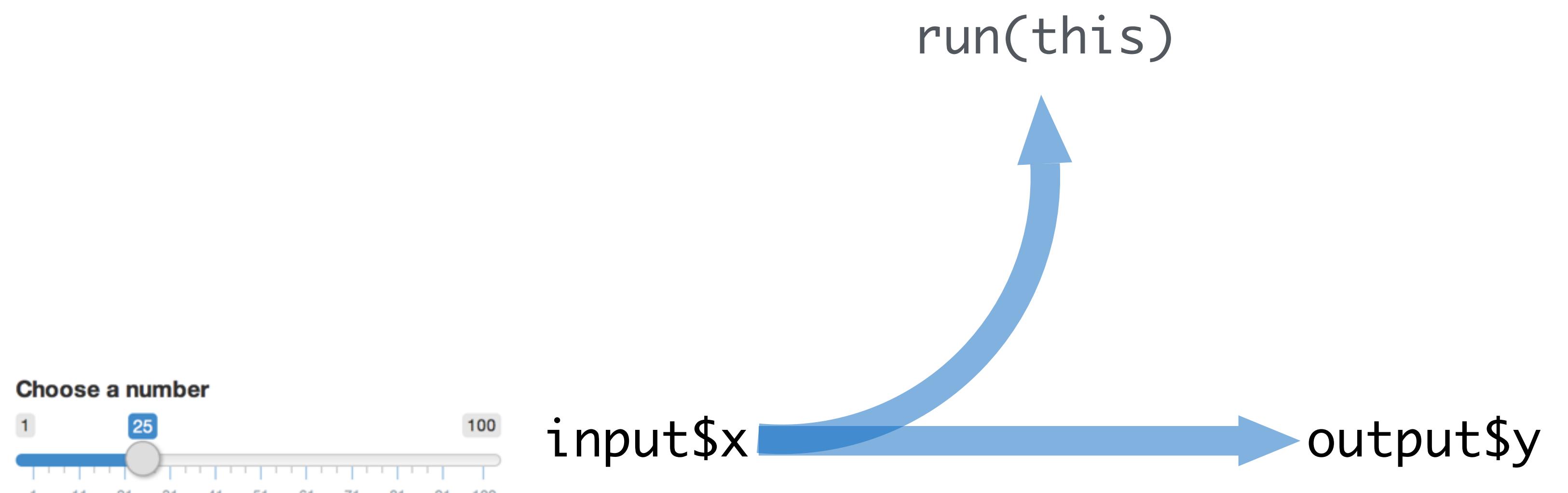
The screenshot shows a Microsoft Excel spreadsheet titled "Workbook1". The ribbon menu is visible with tabs for Tables, Charts, SmartArt, Formulas, Data, and Review. The "Formulas" tab is selected. The formula bar shows the formula `=1000` with a dropdown arrow. The main area displays a single cell with the value "1000". A large blue arrow points from this cell to the adjacent cell, which contains the value "1001". The ribbon also includes sections for Font, Alignment, Number, Format, and Cells, with various tools like Conditional Formatting, Styles, Insert, and Delete. The column headers C through O are visible at the top.

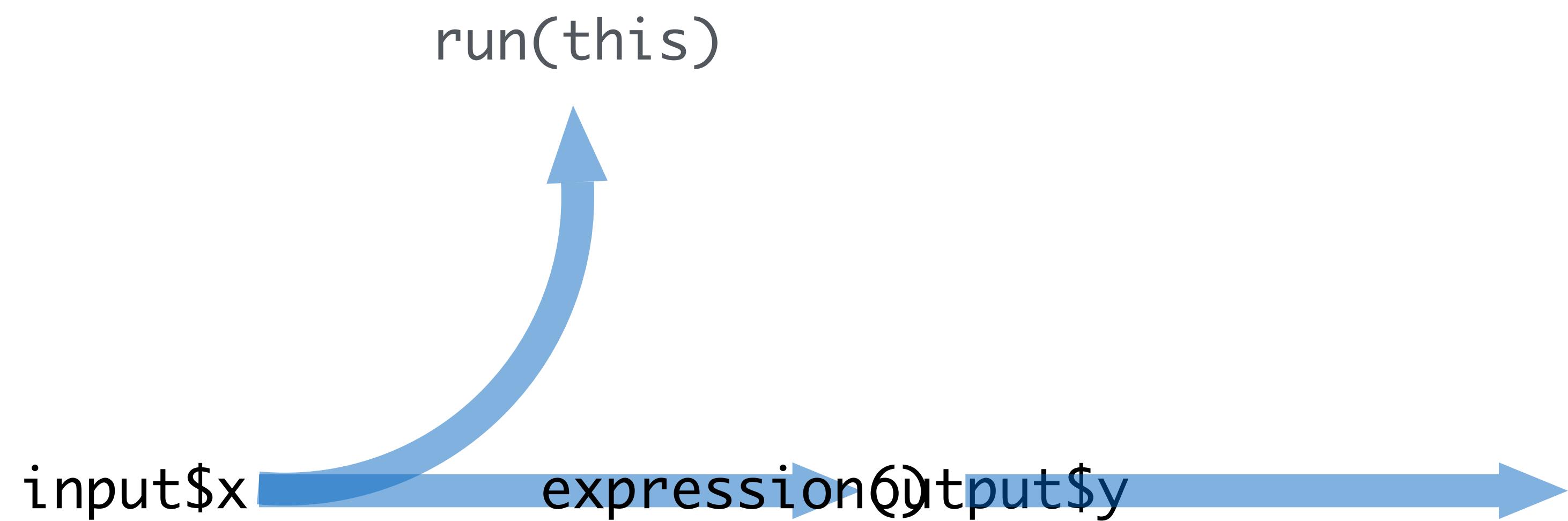
Workbook1

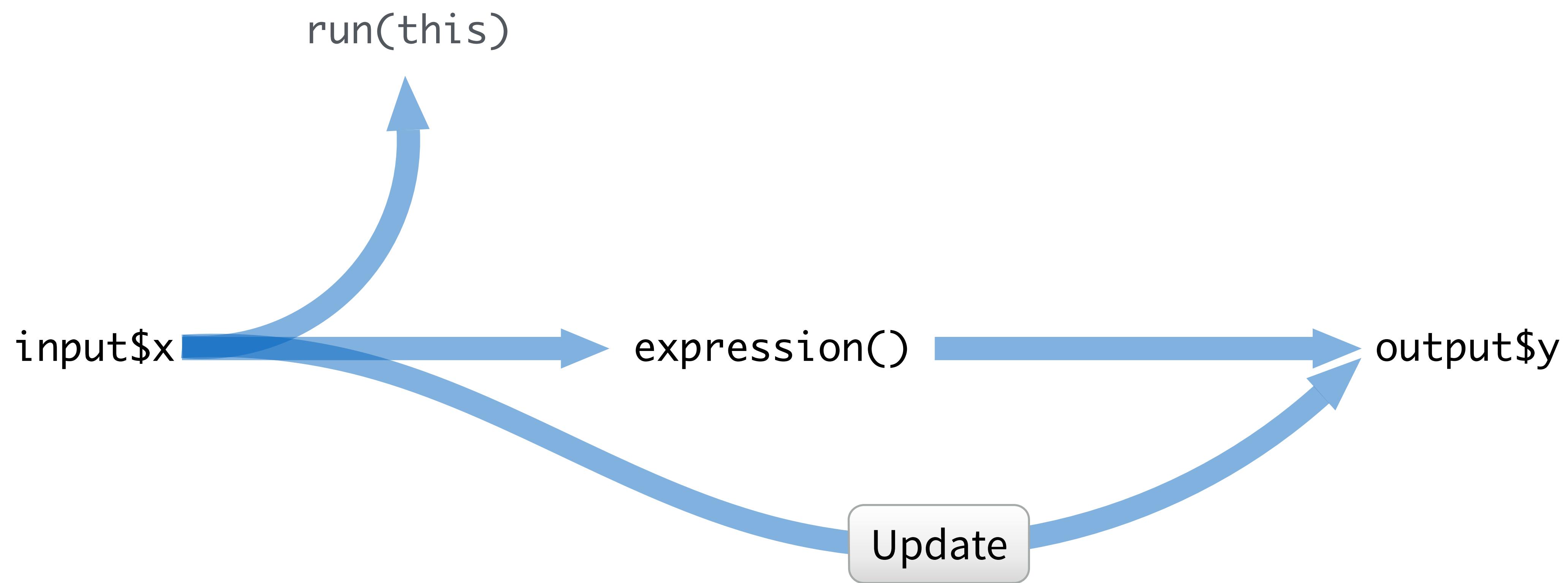
Tables Charts SmartArt Formulas Data Review

Font Alignment Number Format Cells

1000 → 1001
input\$x → output\$y



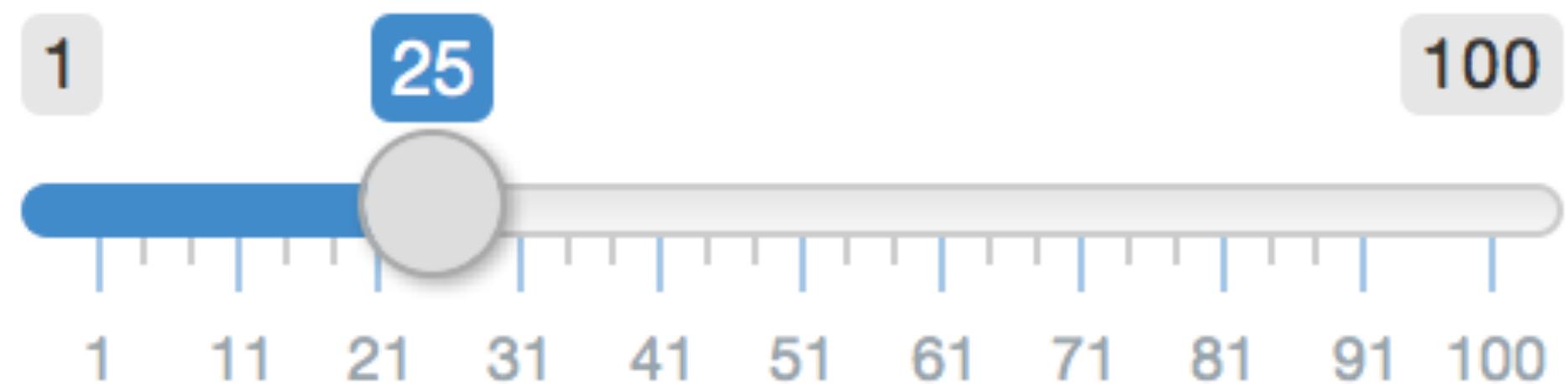




**Begin with
reactive values**

Syntax

Choose a number

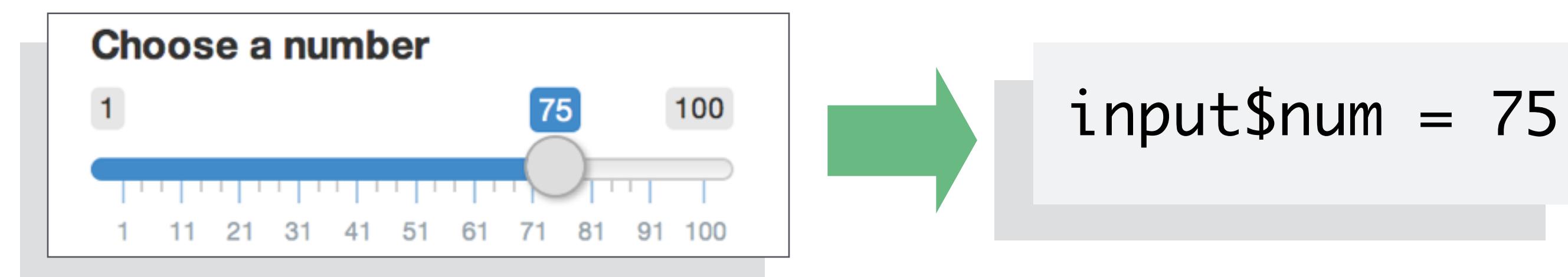
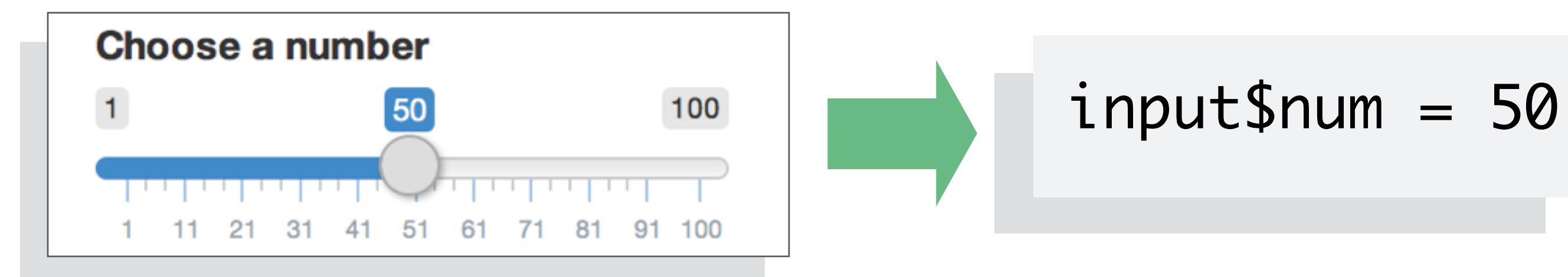
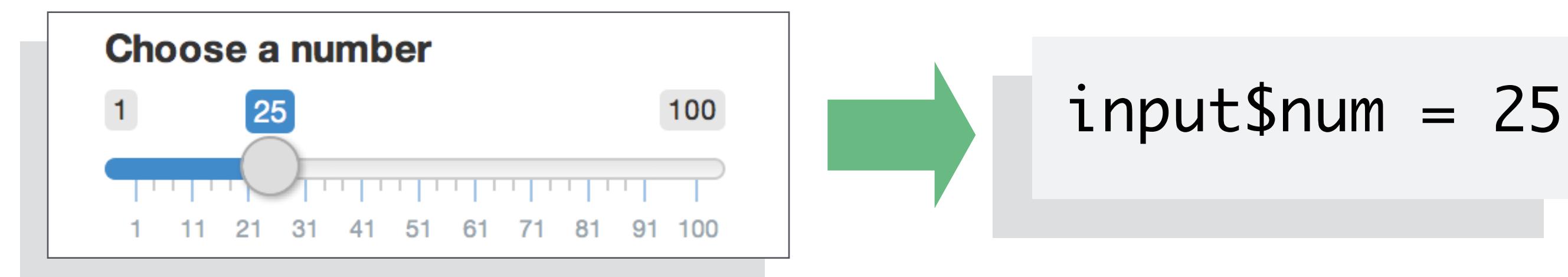


```
sliderInput(inputId = "num", label = "Choose a number", ...)
```

this input will provide a value
saved as `input$num`

Input values

The input value changes whenever a user changes the input.



Reactive values work together with reactive functions.
You cannot call a reactive value from outside of one.



```
renderPlot({ hist(rnorm(100, input$num)) })
```



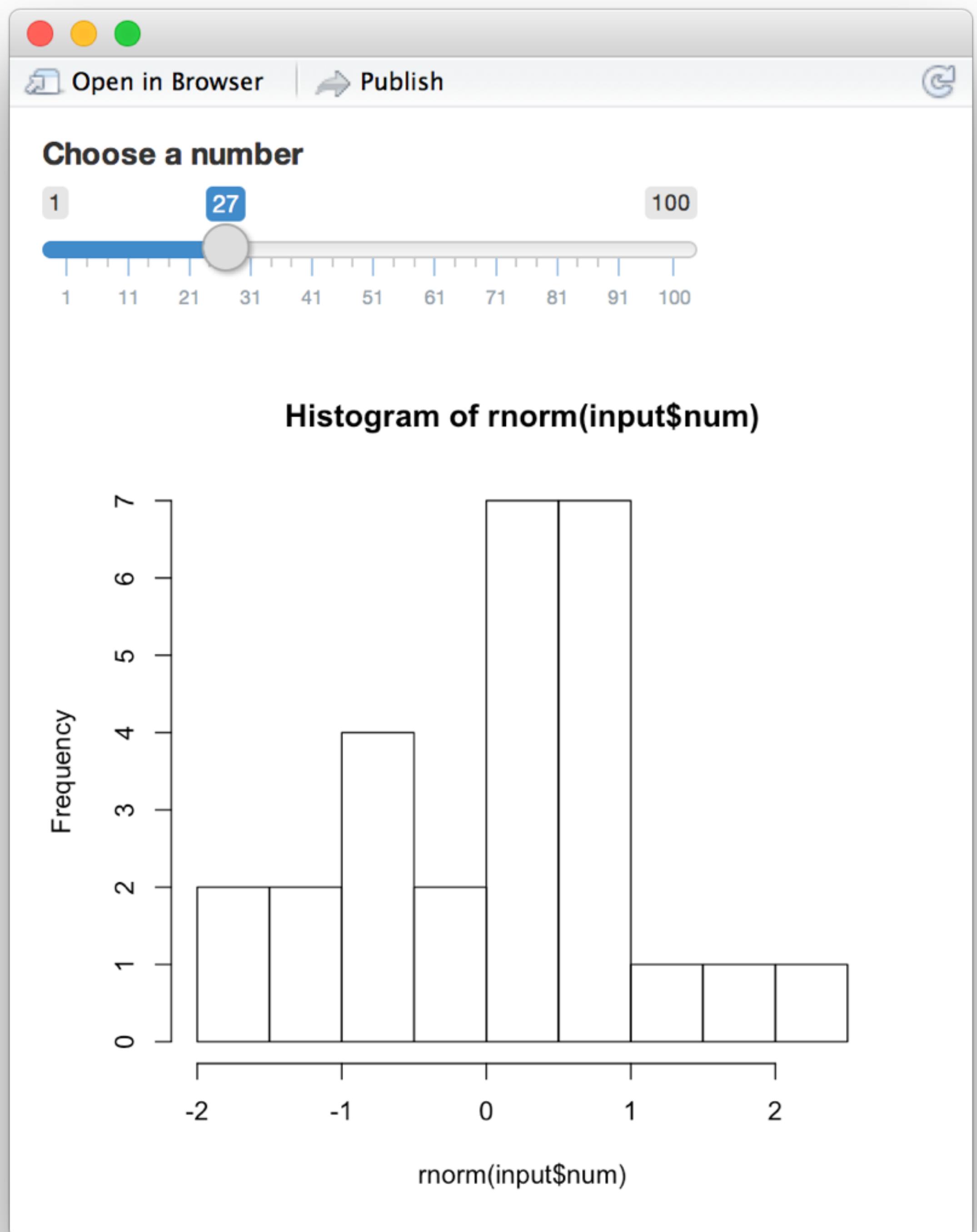
```
hist(rnorm(100, input$num))
```

```
library(shiny)

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

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

shinyApp(ui = ui, server = server)
```

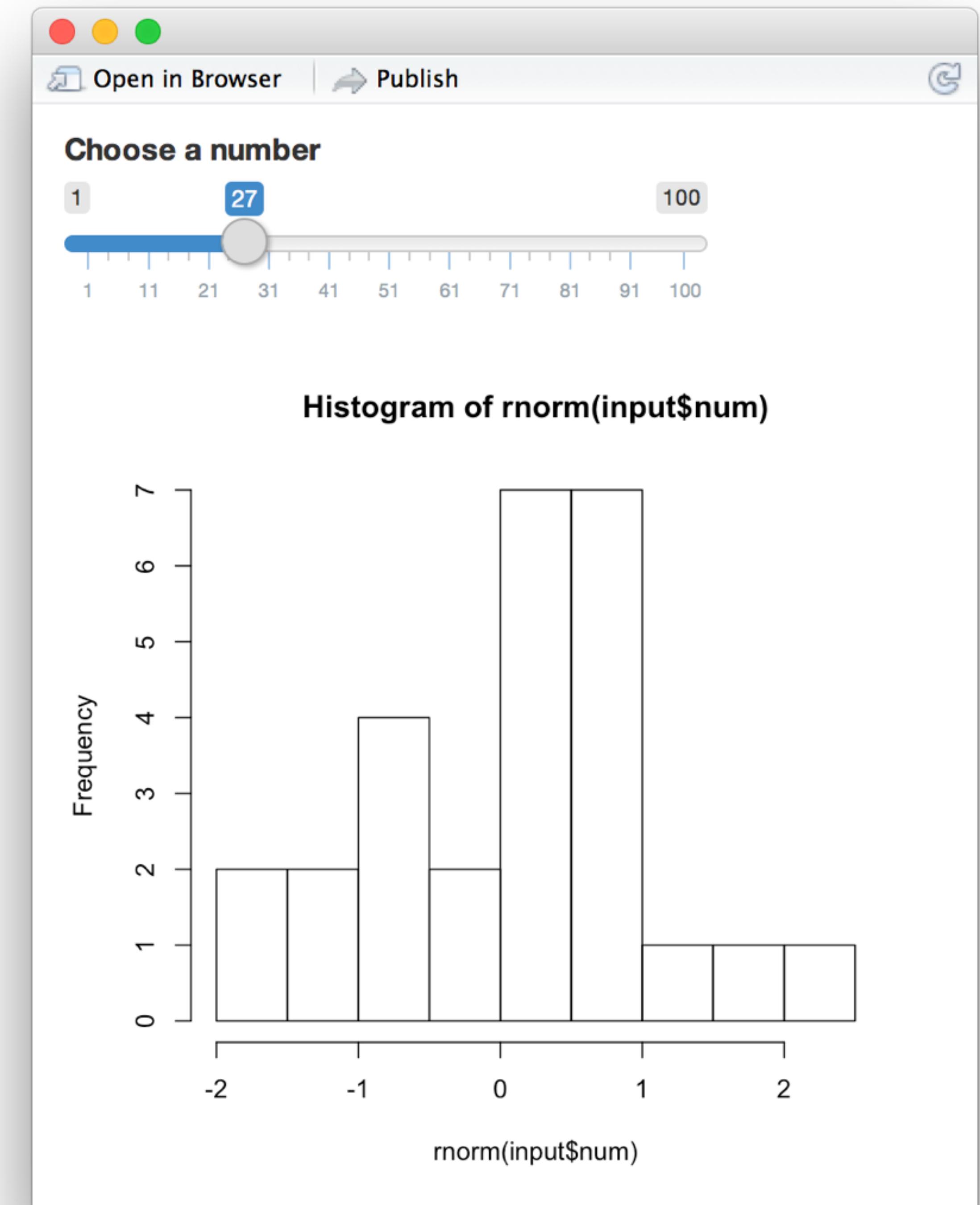


```
library(shiny)

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

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

shinyApp(ui = ui, server = server)
```



```
library(shiny)

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

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

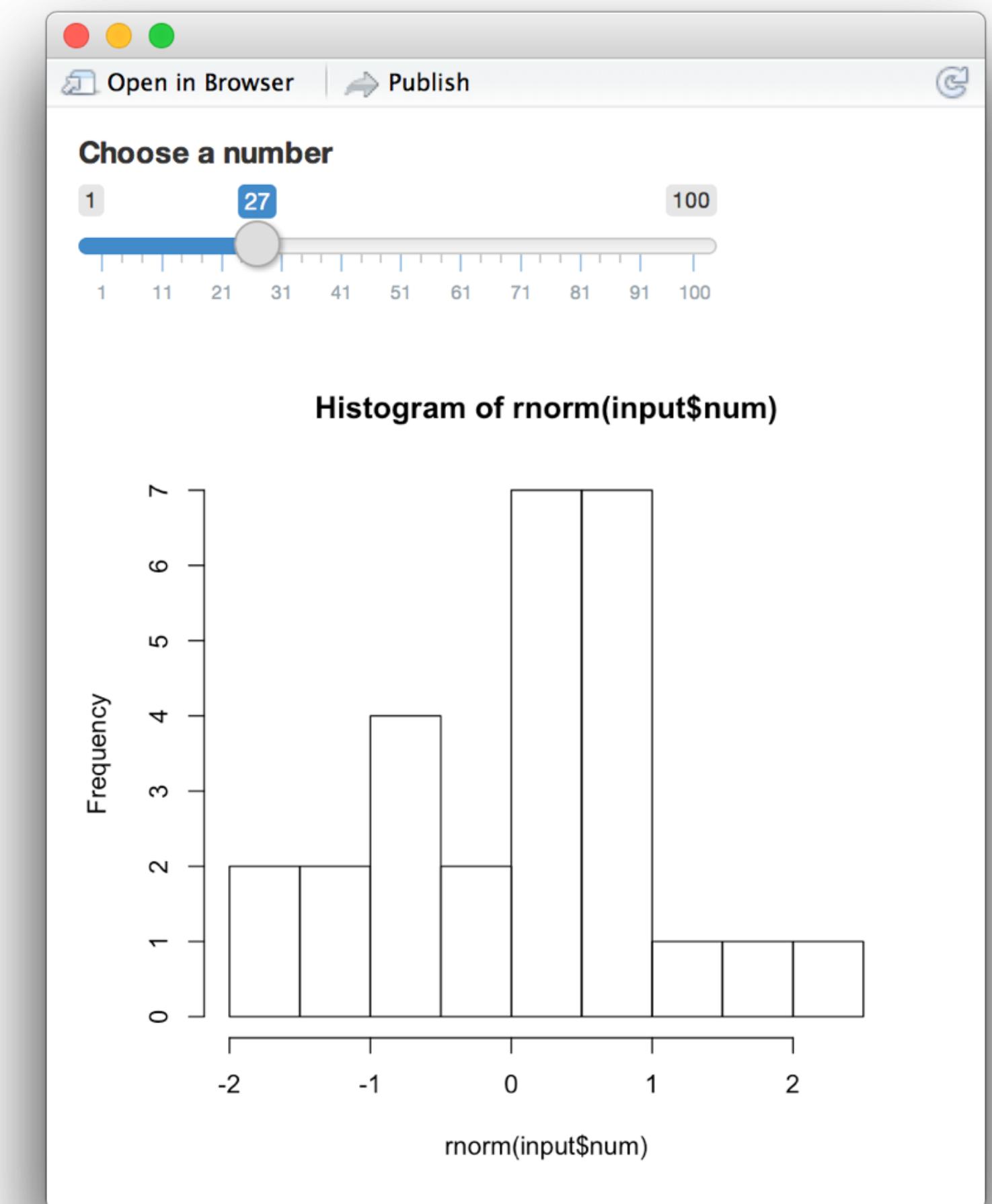
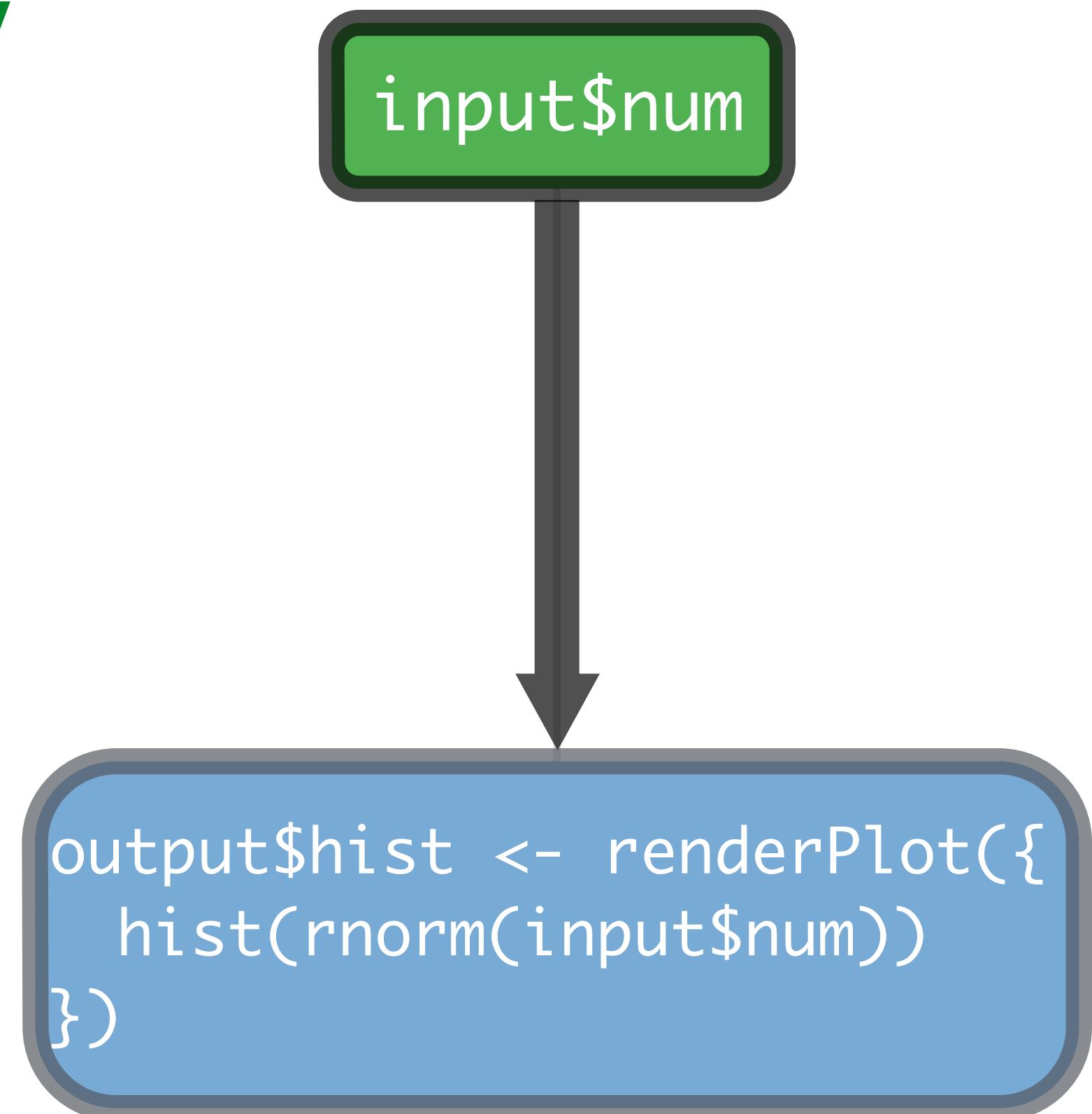
shinyApp(ui = ui, server = server)
```

Error in .getReactiveEnvironment()
\$currentContext() :
Operation not allowed without an
active reactive context. (You tried
to do something that can only be done
from inside a reactive expression or
observer.)

Think of reactivity in R as a two step process

1 Reactive values notify

the functions that use them
when they become invalid

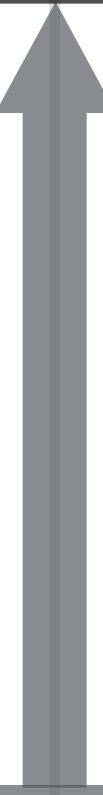


Think of reactivity in R as a two step process

1 Reactive values notify

the functions that use them
when they become invalid

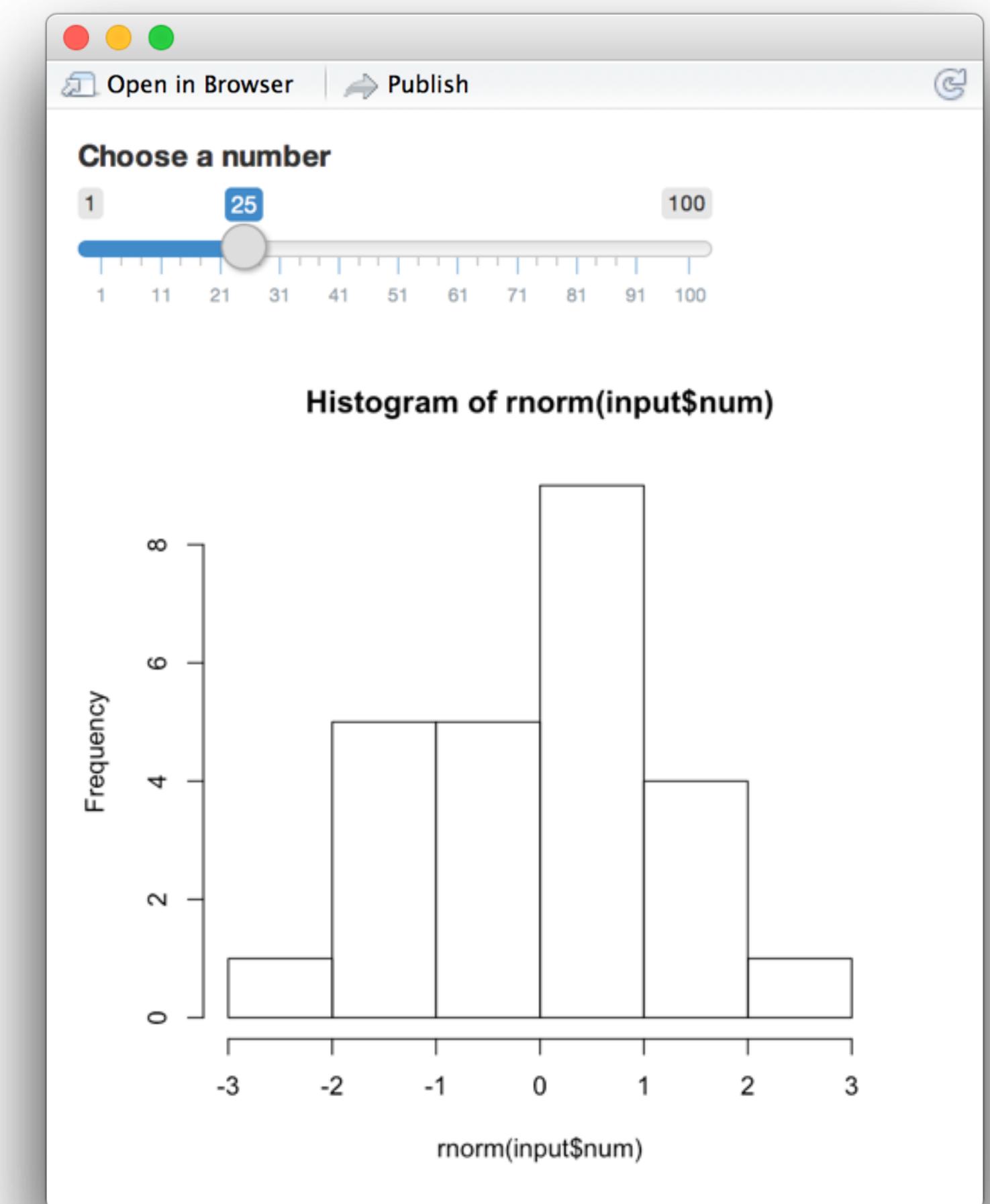
input\$num



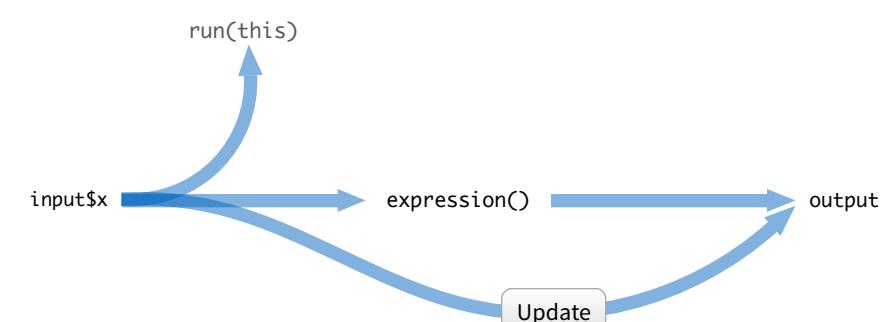
2 The objects created by reactive functions respond

(different objects respond differently)

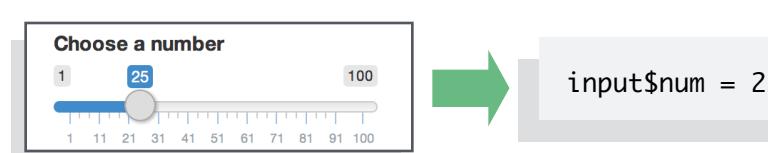
```
output$hist <- renderPlot({  
  hist(rnorm(input$num))  
})
```



Recap: Reactive values



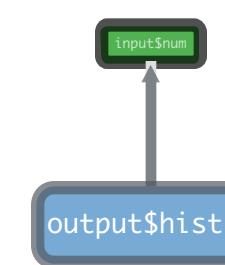
Reactive values act as the data streams that flow through your app.



The **input** list is a list of reactive values. The values show the current state of the inputs.



You can only call a reactive value from a function that is designed to work with one



Reactive values notify. The objects created by **reactive functions respond.**

Reactive toolkit

(7 indispensable functions)

Reactive functions

- 1** Use a code chunk to build (and rebuild) an object
 - **What code** will the function use?

- 2** The object will respond to changes in a set of reactive values
 - **Which reactive values** will the object respond to?

**Display output
with render*()**

Render functions build output to display in the app

function	creates
renderDataTable()	An interactive table <small>(from a data frame, matrix, or other table-like structure)</small>
renderImage()	An image (saved as a link to a source file)
renderPlot()	A plot
renderPrint()	A code block of printed output
renderTable()	A table <small>(from a data frame, matrix, or other table-like structure)</small>
renderText()	A character string
renderUI()	a Shiny UI element

render*()

Builds reactive output to display in UI

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

object will respond to *every reactive value in the code*

code used to build (and rebuild) object

render*()

Builds reactive output to display in UI

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

When notified that it is invalid, the object created by a render*() function **will rerun the entire block of code** associated with it

```

# 01-two-inputs

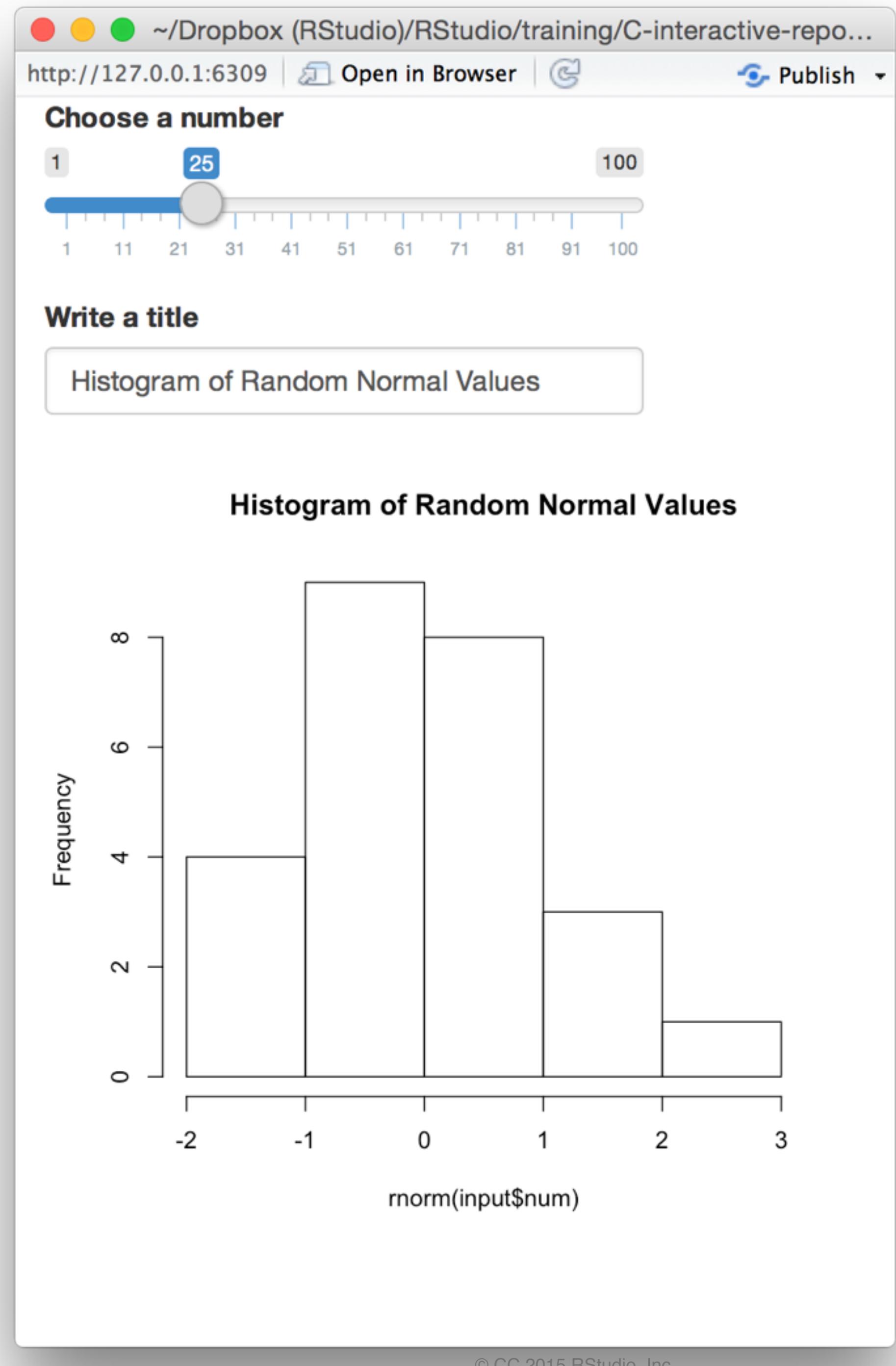
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num), main = input$title)
  })
}

shinyApp(ui = ui, server = server)

```



```

# 01-two-inputs

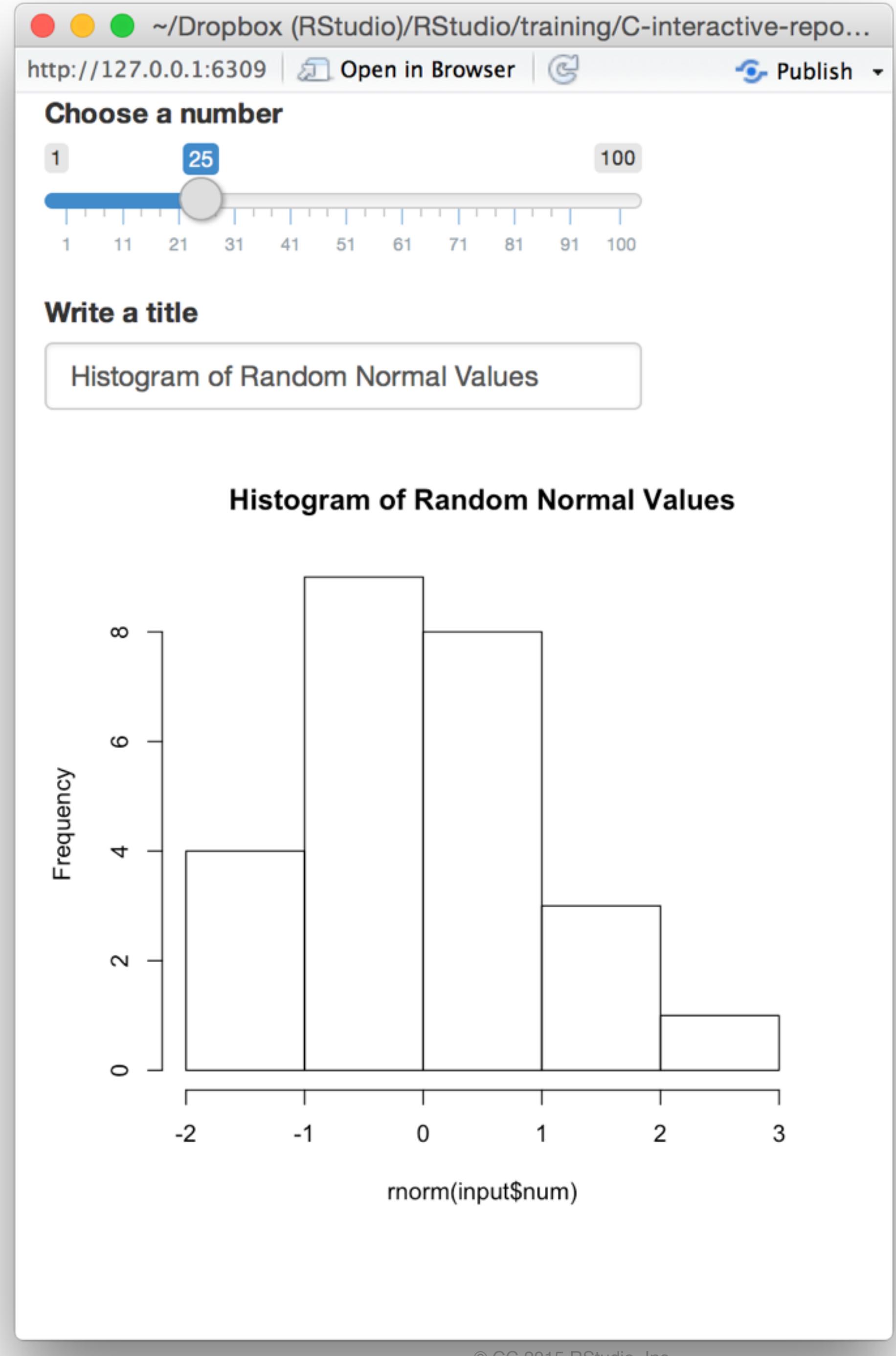
library(shiny)

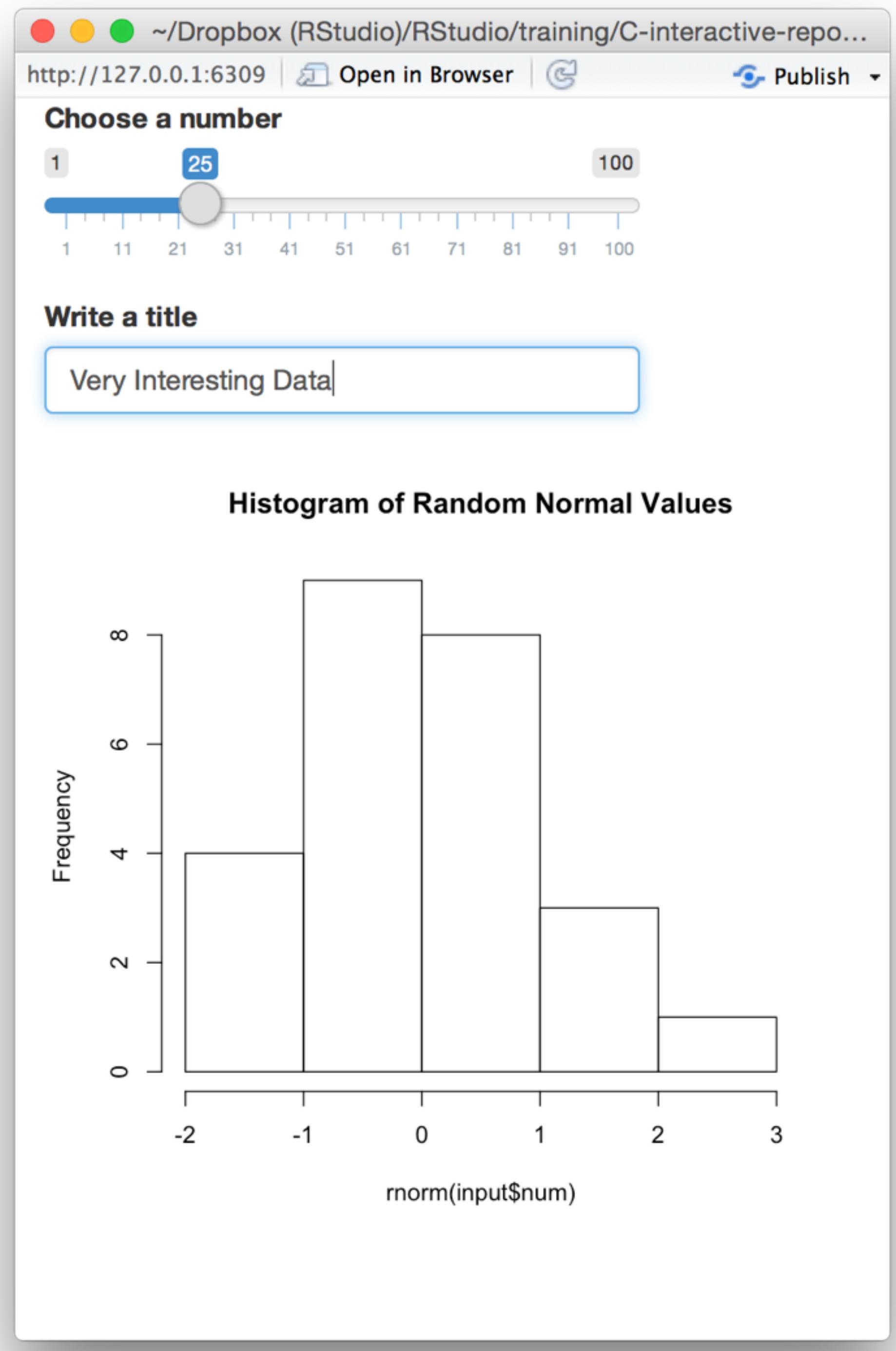
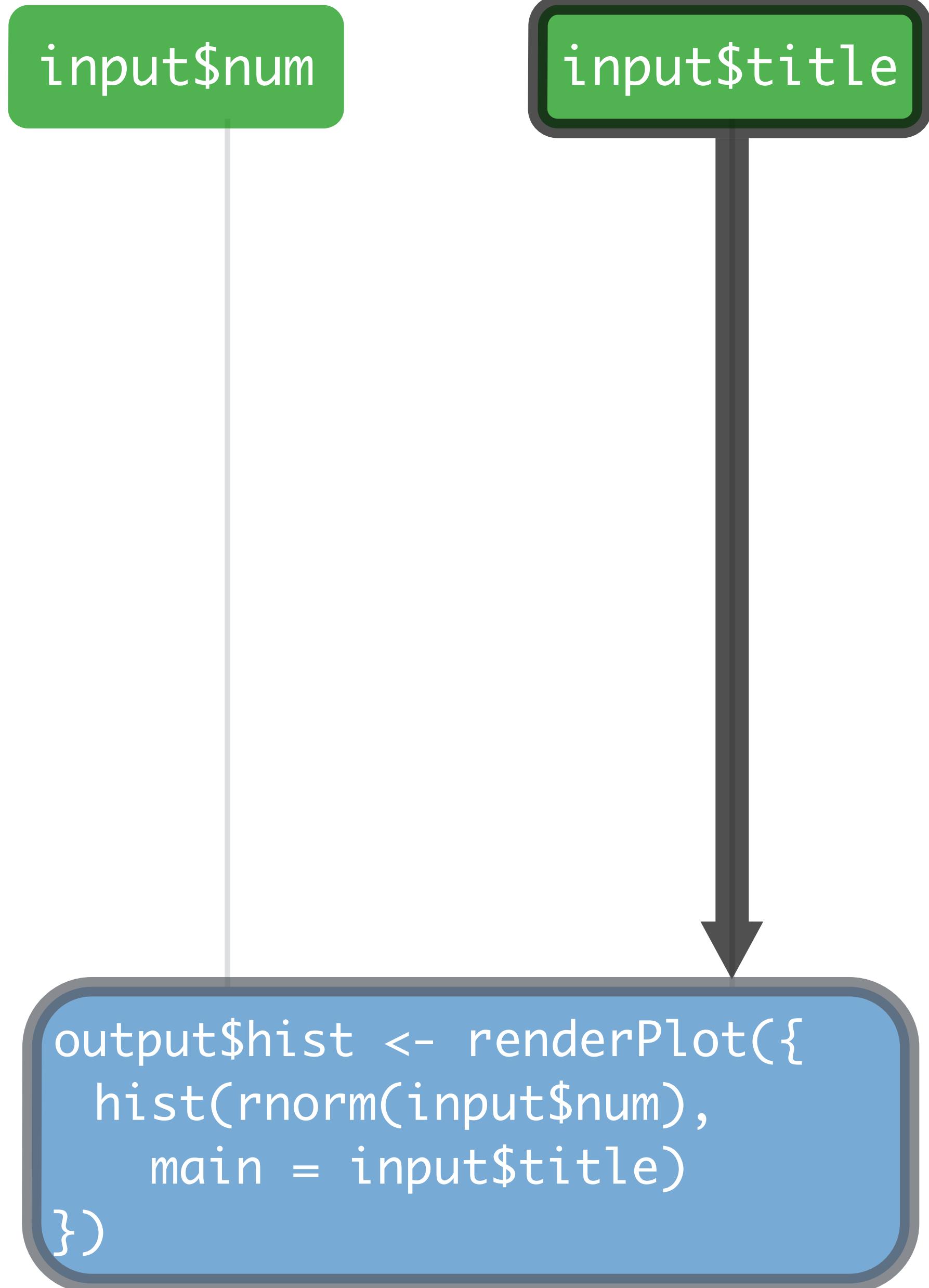
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
 textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num), main = input$title)
  })
}

shinyApp(ui = ui, server = server)

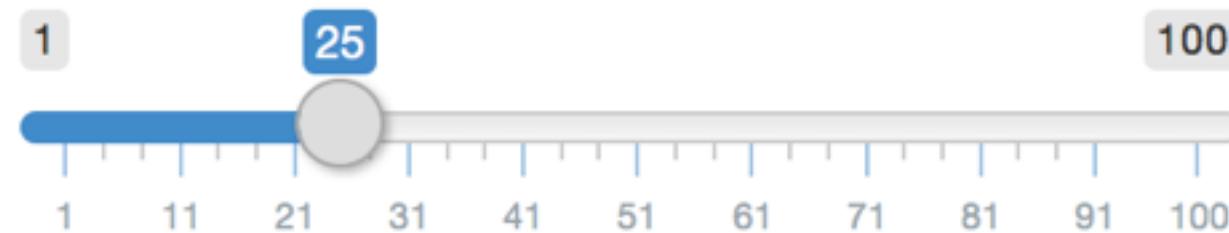
```





Choose a number

1 25 100

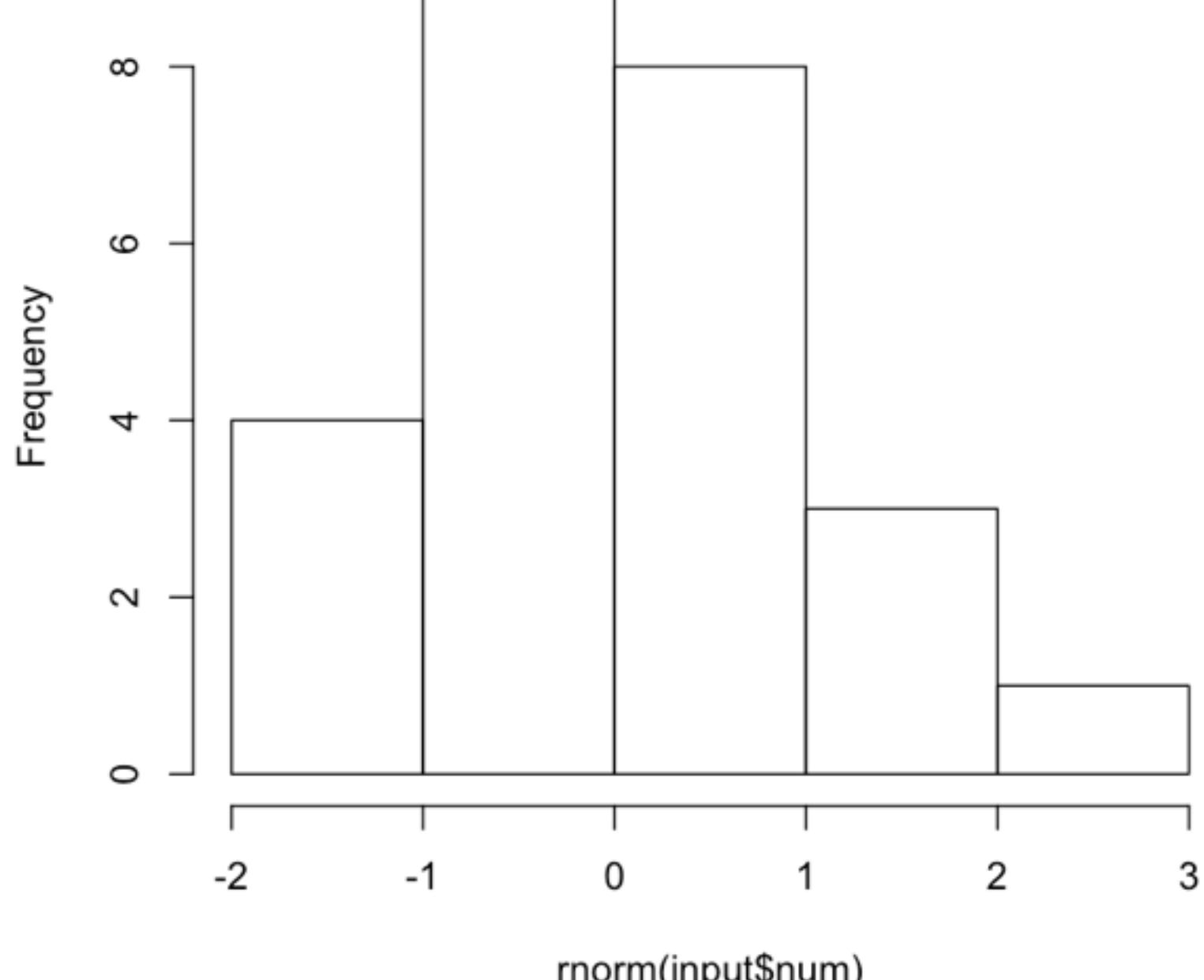


1 11 21 31 41 51 61 71 81 91 100

Write a title

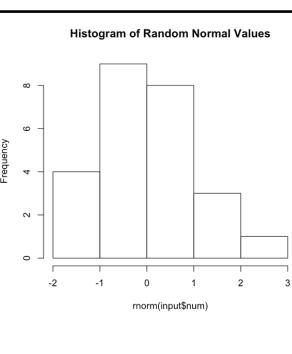
Very Interesting Data

Histogram of Random Normal Values



```
output$hist <- renderPlot({  
  hist(rnorm(input$num),  
        main = input$title)  
})
```

Recap: render*



render*() functions make **objects to display**

output\$

Always save the result to **output\$**

```
output$hist <- renderPlot({  
  hist(rnorm(input$num),  
    main = input$title)  
})
```

render*() makes an observer object that has a
block of code associated with it

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

The object will **rerun the entire code block**
to update itself whenever it is invalidated

**Modularize code
with reactive()**

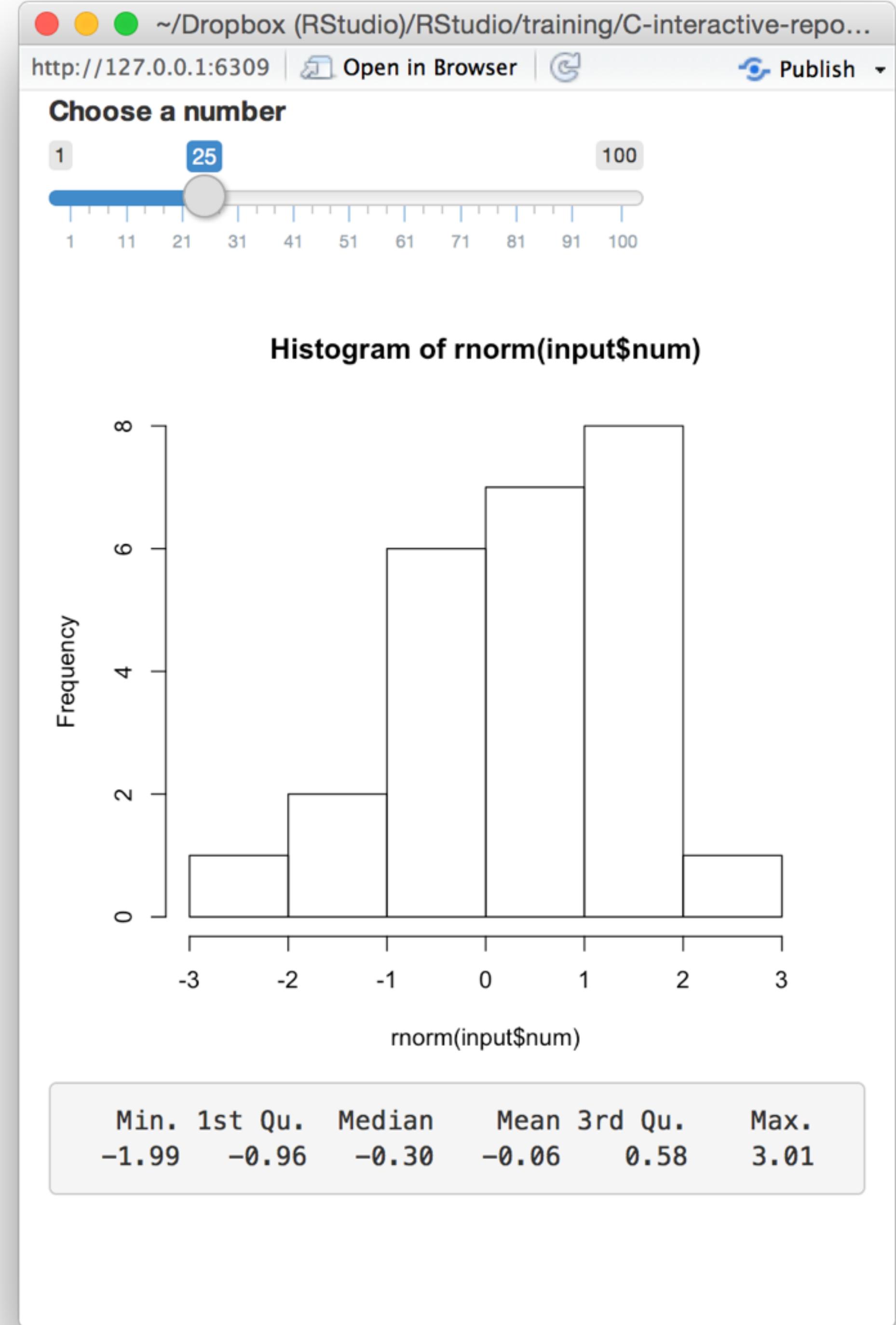
```
# 02-two-outputs

library(shiny)

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

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
  output$stats <- renderPrint({
    summary(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



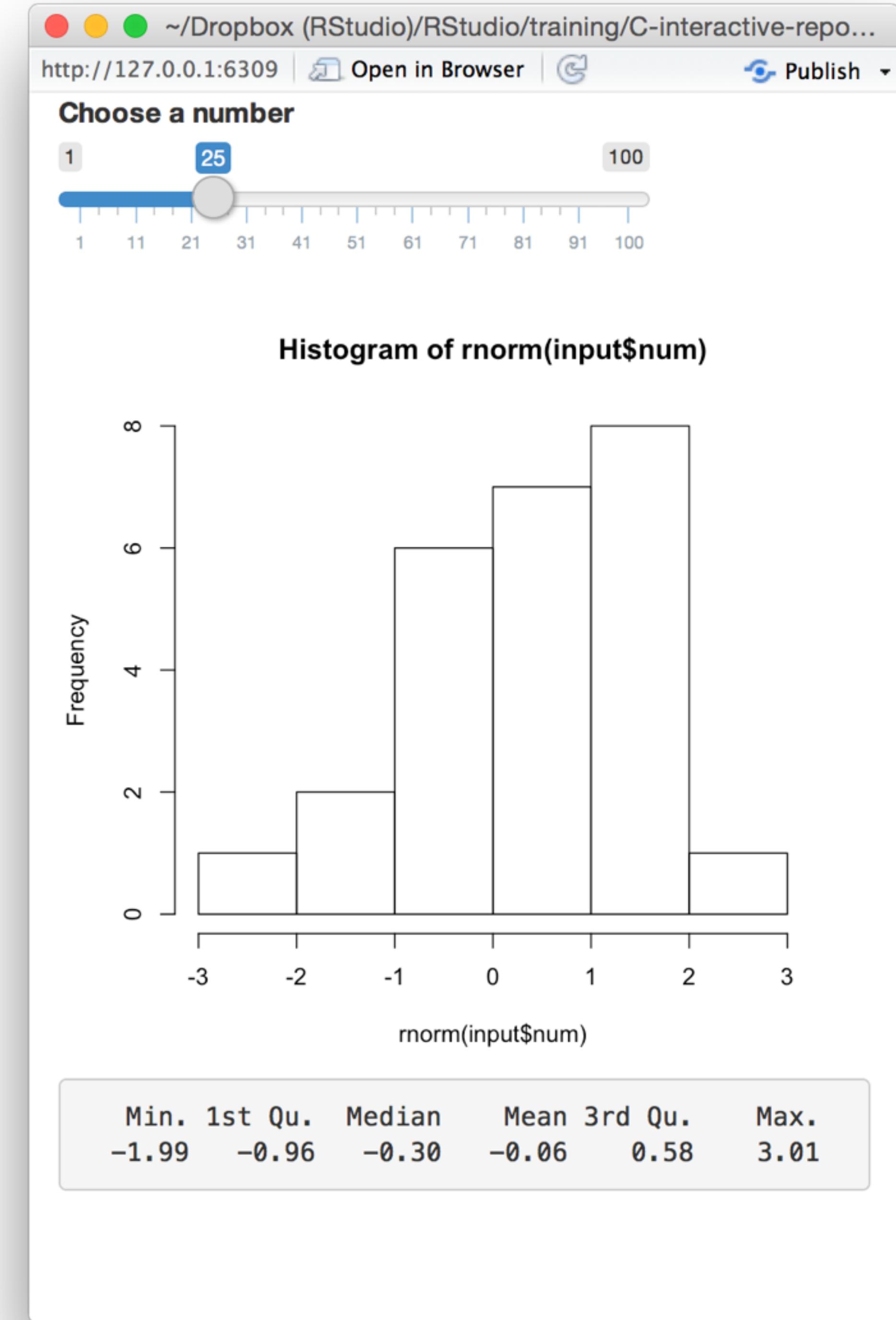
```
# 02-two-outputs
```

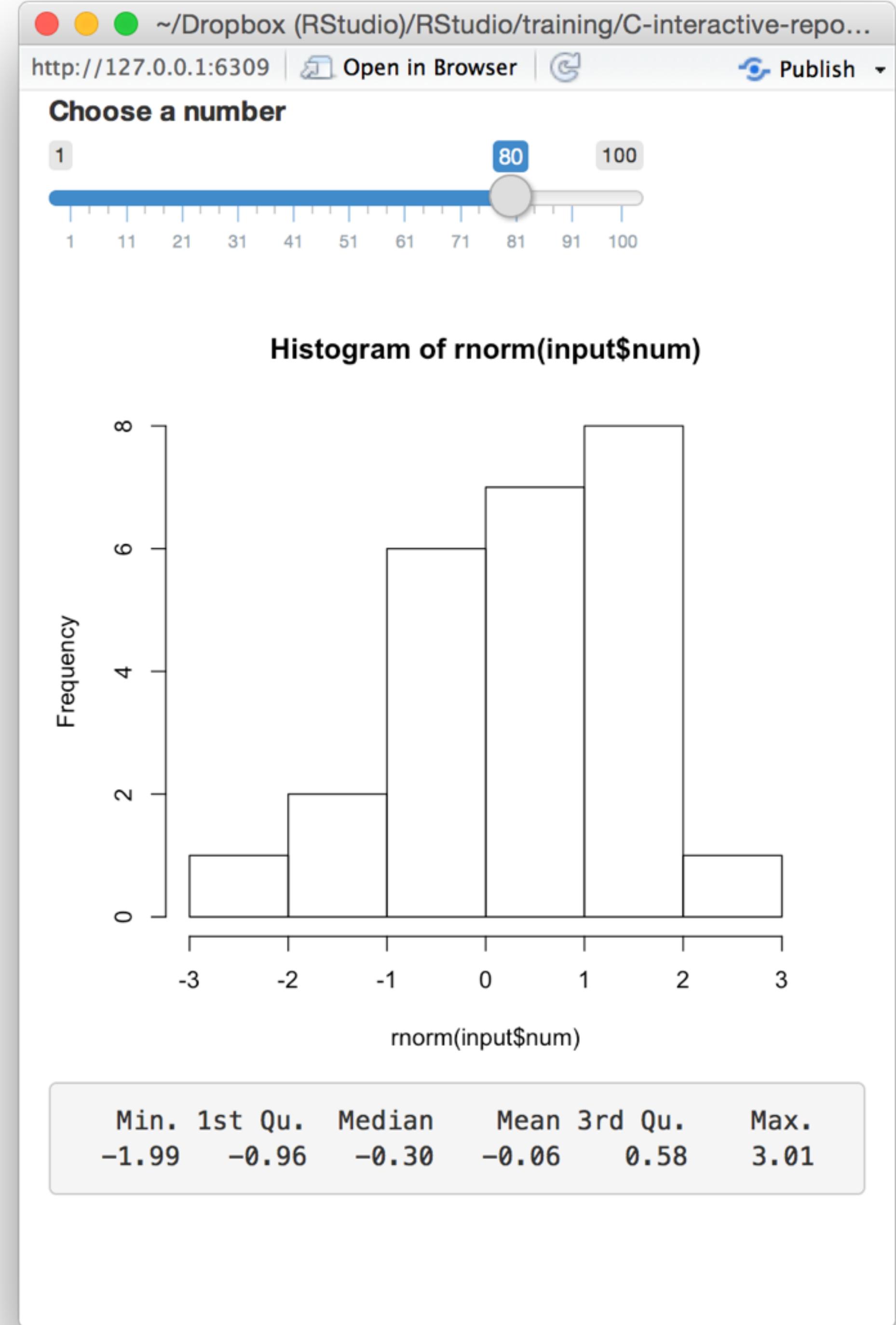
```
library(shiny)
```

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

```
server <- function(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(input$num))  
  })  
  output$stats <- renderPrint({  
    summary(rnorm(input$num))  
  })  
}
```

```
shinyApp(ui = ui, server = server)
```

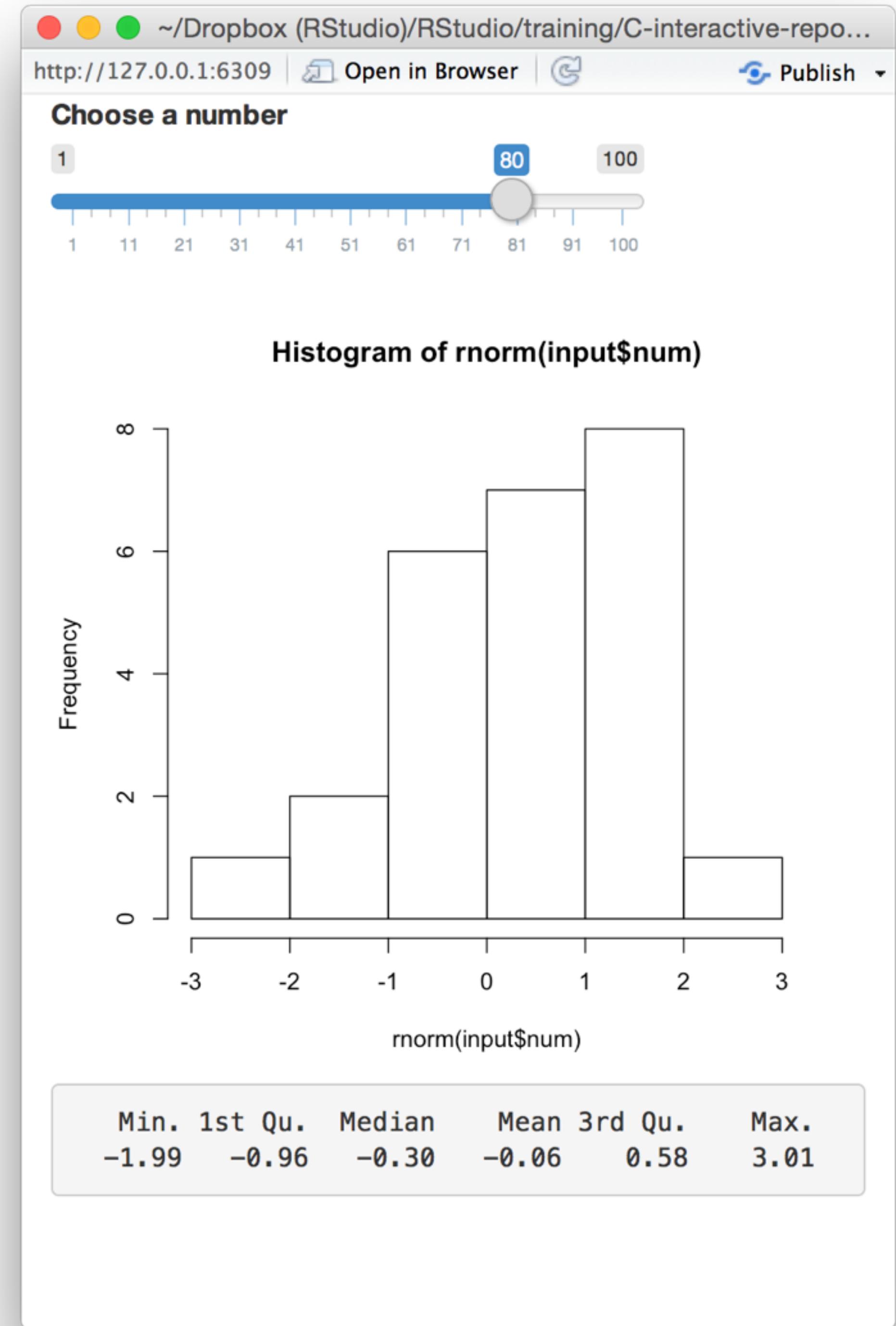




input\$num

```
output$hist <-  
  renderPlot({  
    hist(rnorm(input$num))  
  })
```

```
output$stats <-  
  renderPrint({  
    summary(rnorm(input$num))  
  })
```



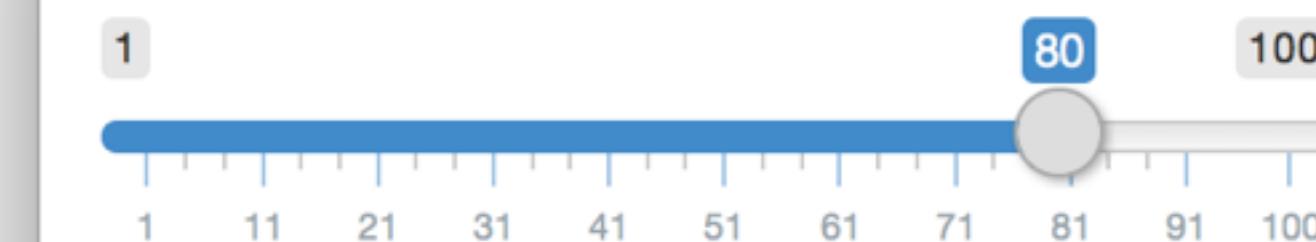
`input$num`

```
output$hist <-  
  renderPlot({  
    hist(rnorm(input$num))  
  })
```

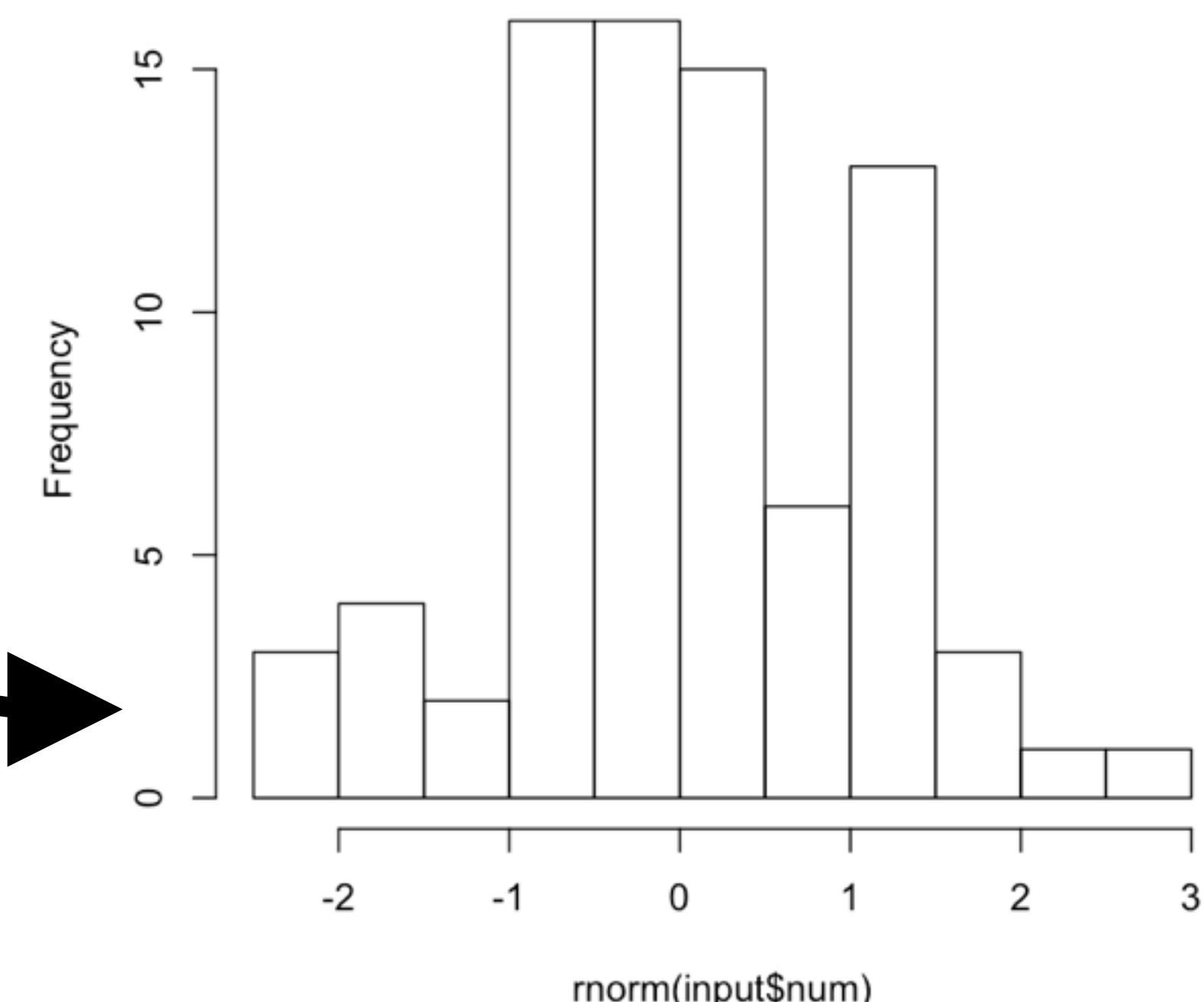
```
output$stats <-  
  renderPrint({  
    summary(rnorm(input$num))  
  })
```

input\$num

Choose a number



Histogram of rnorm(input\$num)



**Can these describe
the same data?**

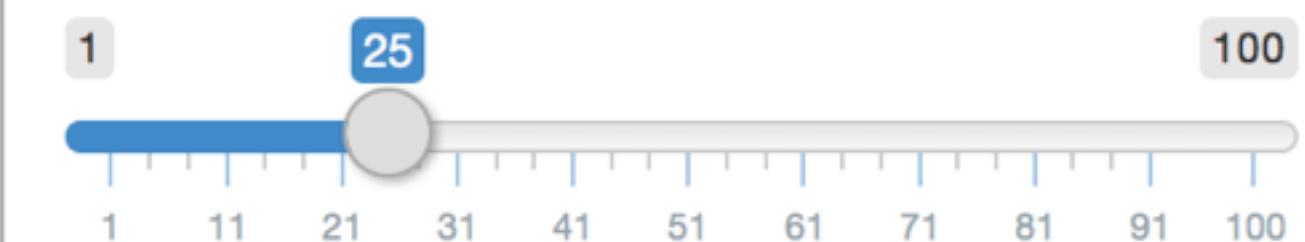
output\$hist <-
renderPlot({
 hist(rnorm(input\$num))
})

output\$stats <-
renderPrint({
 summary(rnorm(input\$num))
})

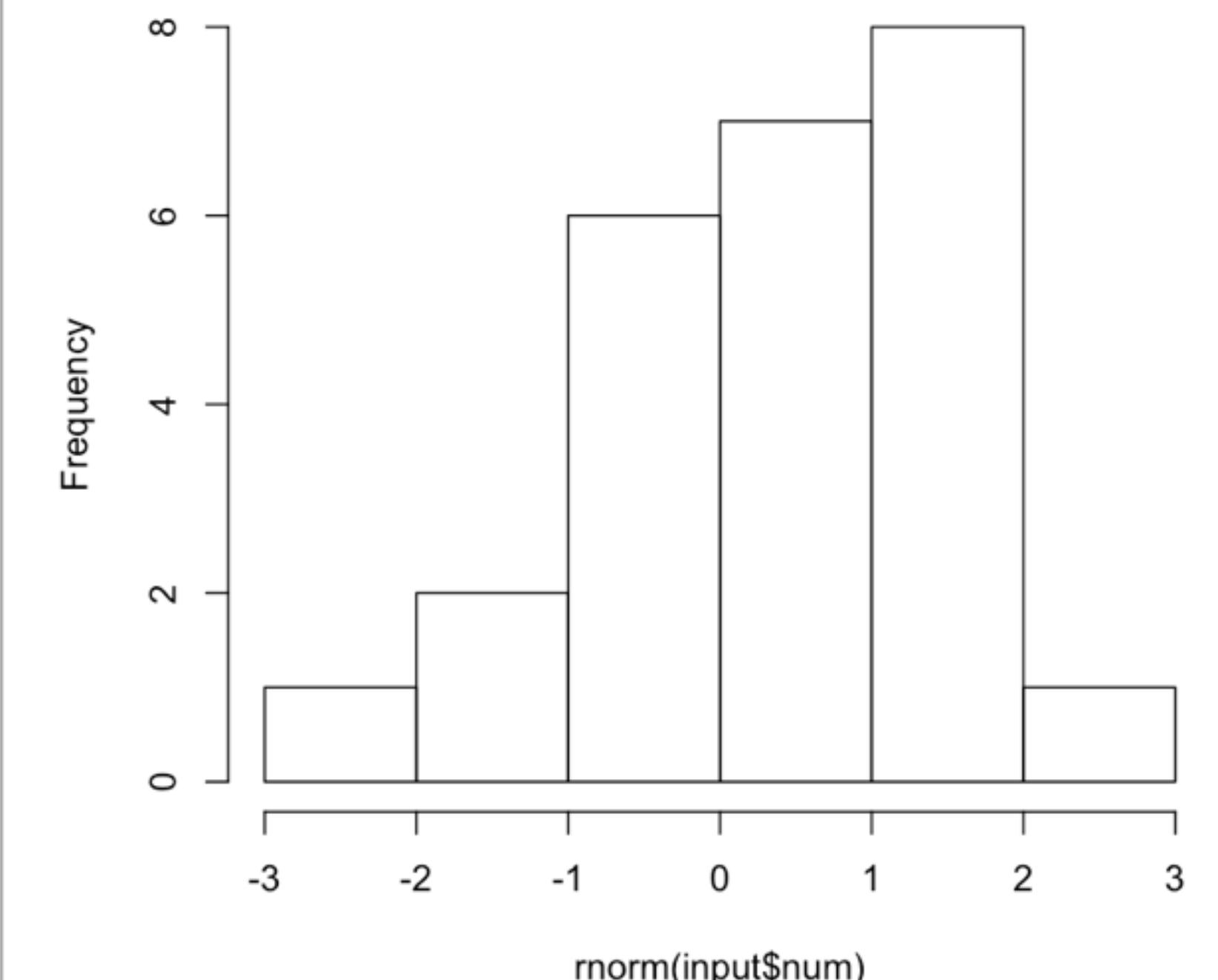
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-2.23	-0.66	0.11	0.11	0.72	2.14

Choose a number

1 25 100



Histogram of rnorm(input\$num)



input\$num

data <-? rnorm(input\$num)

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

output\$stats <-
renderPrint({
 summary(data)
})

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-1.99	-0.96	-0.30	-0.06	0.58	3.01

reactive()

Builds a reactive object (reactive expression)

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

object will respond to *every reactive value in the code*

code used to build (and rebuild) object

A reactive expression is special in two ways

```
data()
```

- 1 You call a reactive expression like a function

```
# 02-two-outputs

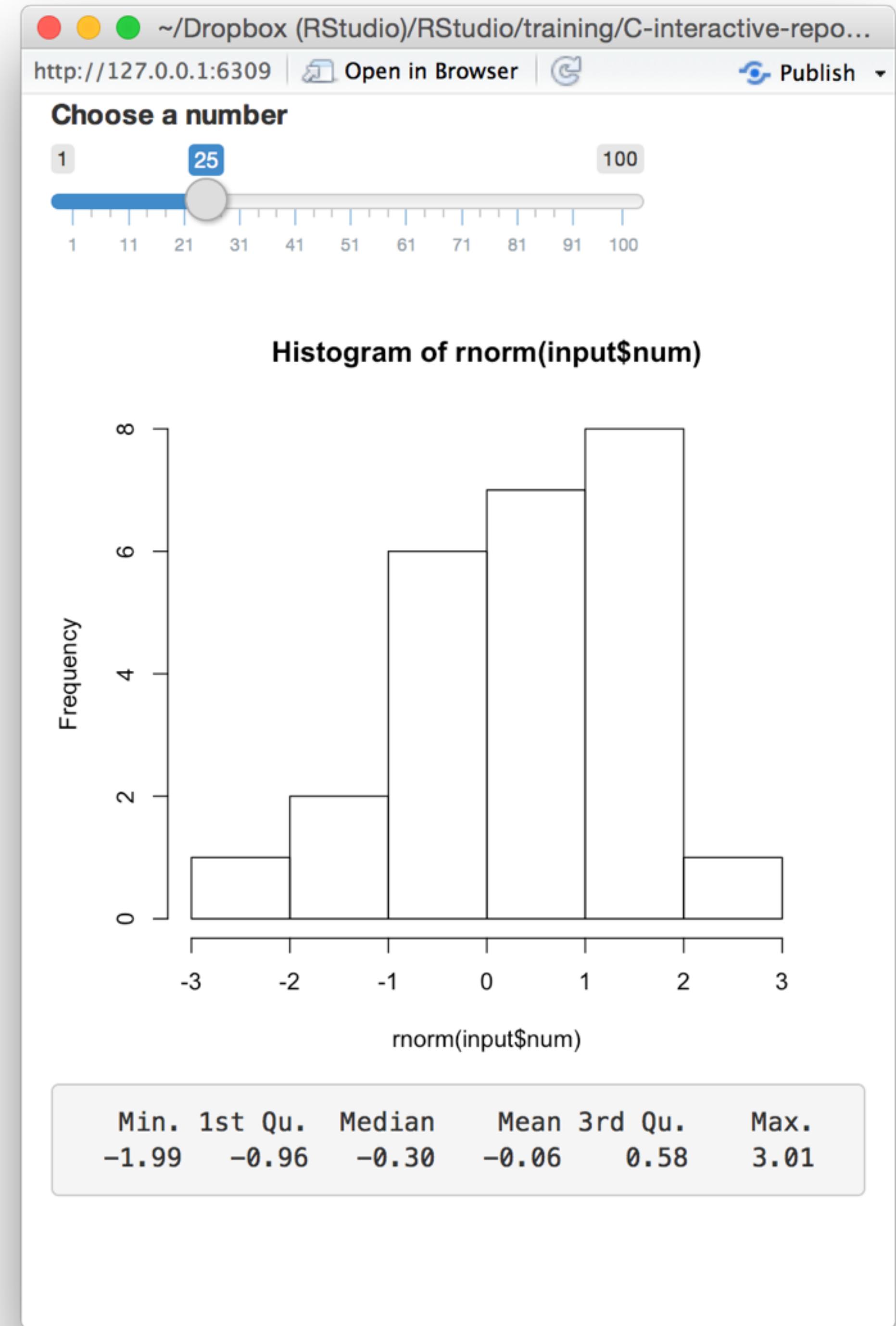
library(shiny)

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

server <- function(input, output) {

  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
  output$stats <- renderPrint({
    summary(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```

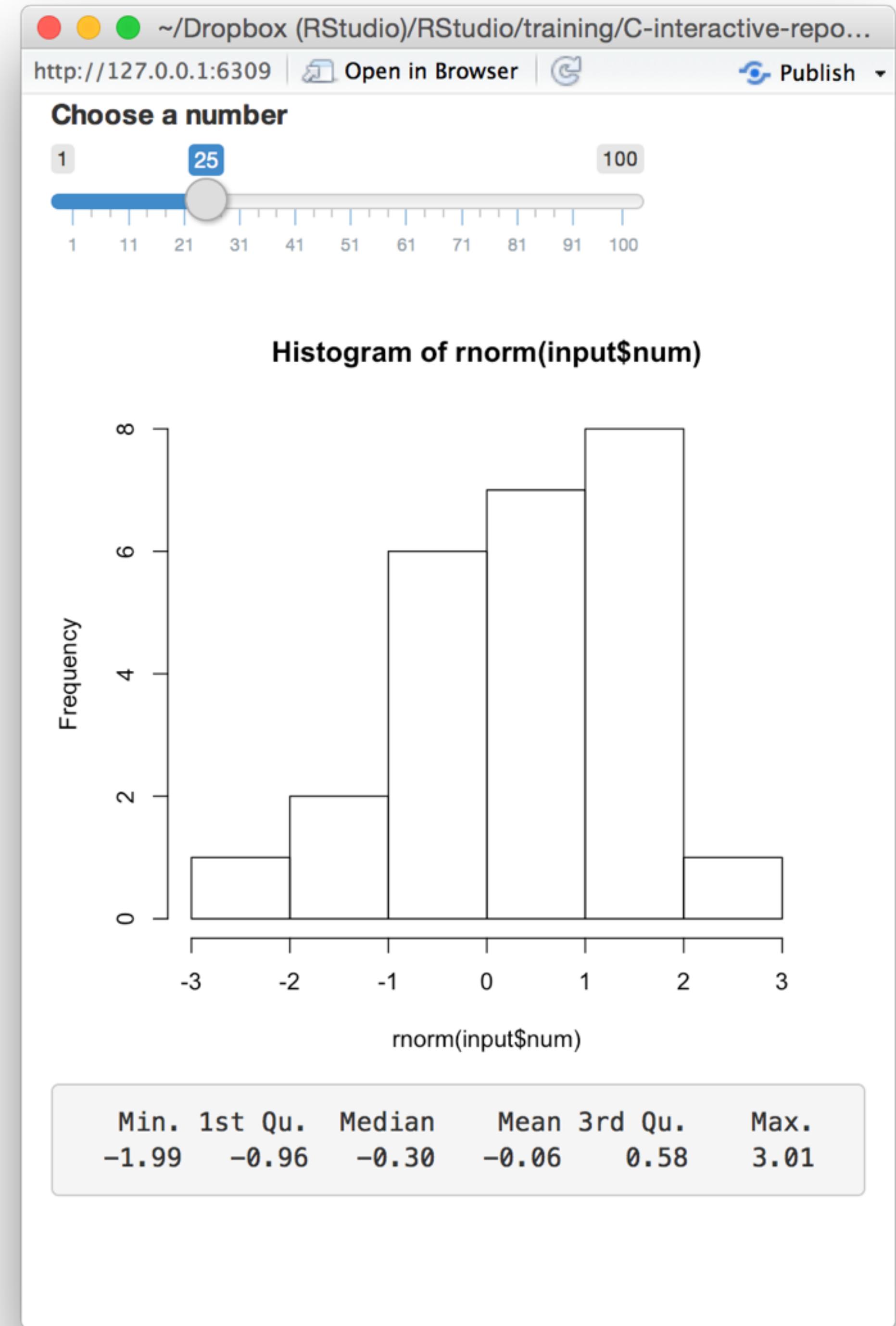


```
# 02-two-outputs

library(shiny)

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

server <- function(input, output) {
  data <- reactive({
    rnorm(input$num)
  })
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
  output$stats <- renderPrint({
    summary(rnorm(input$num))
  })
}
shinyApp(ui = ui, server = server)
```

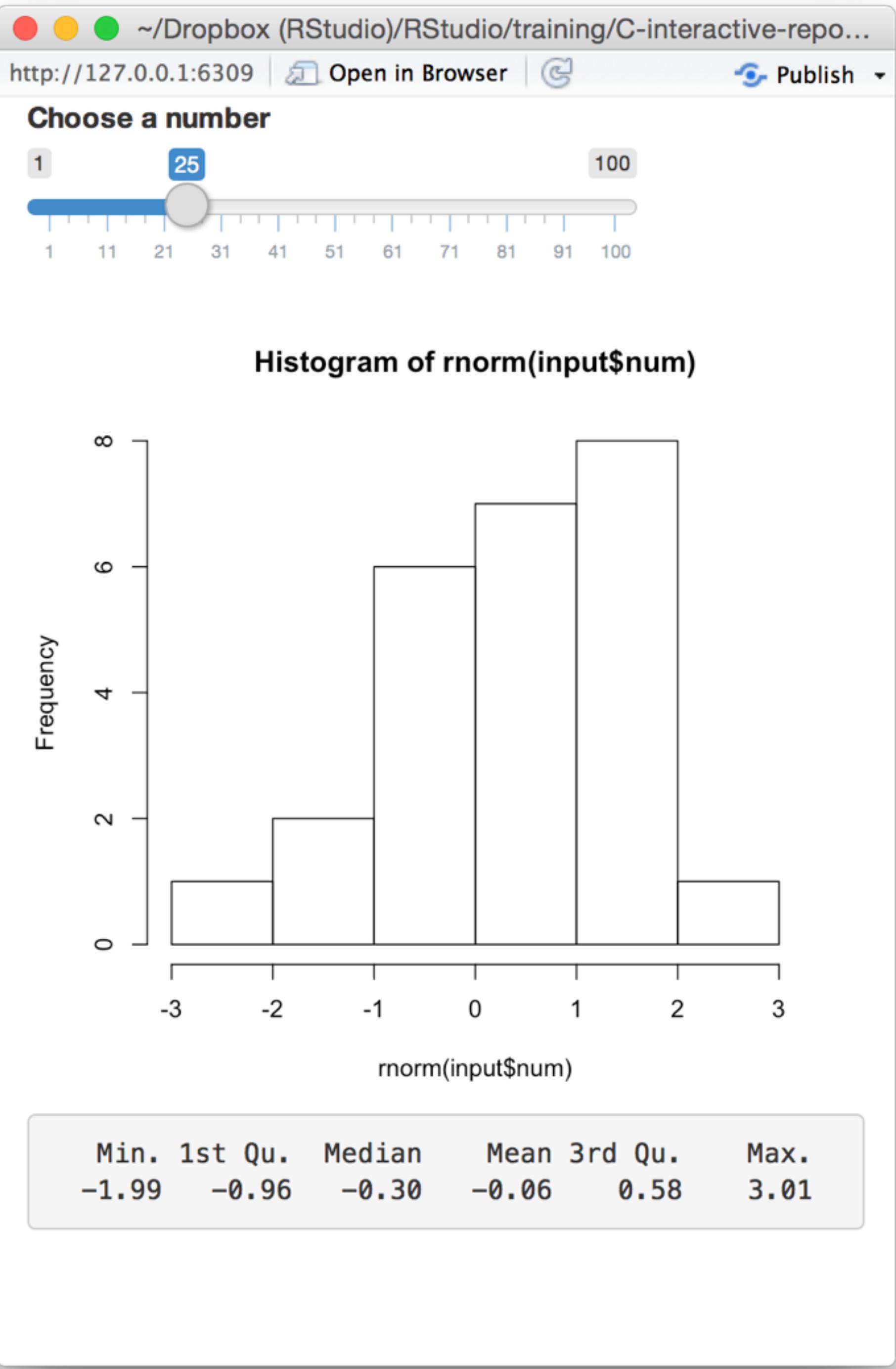


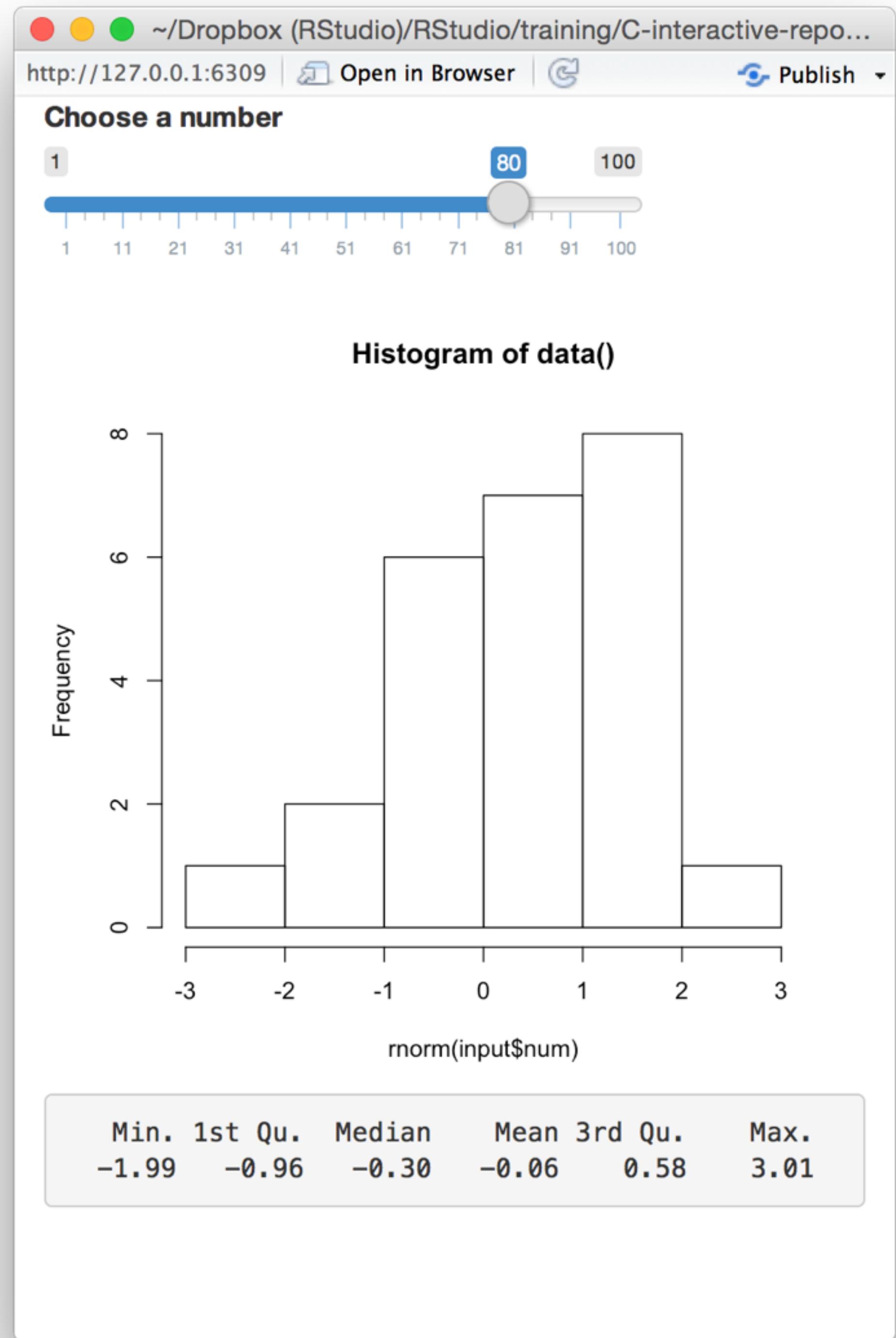
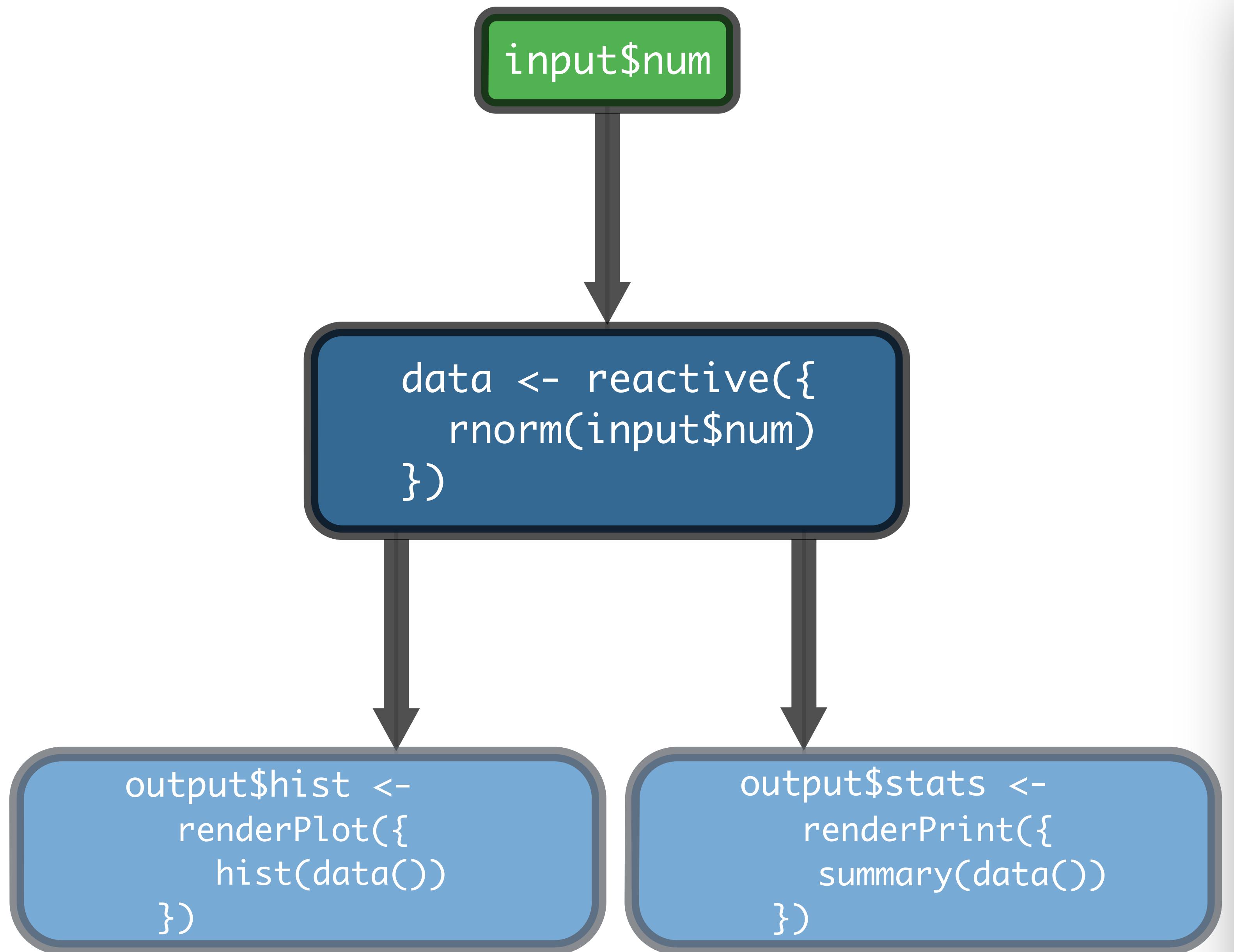
```
# 03-reactive

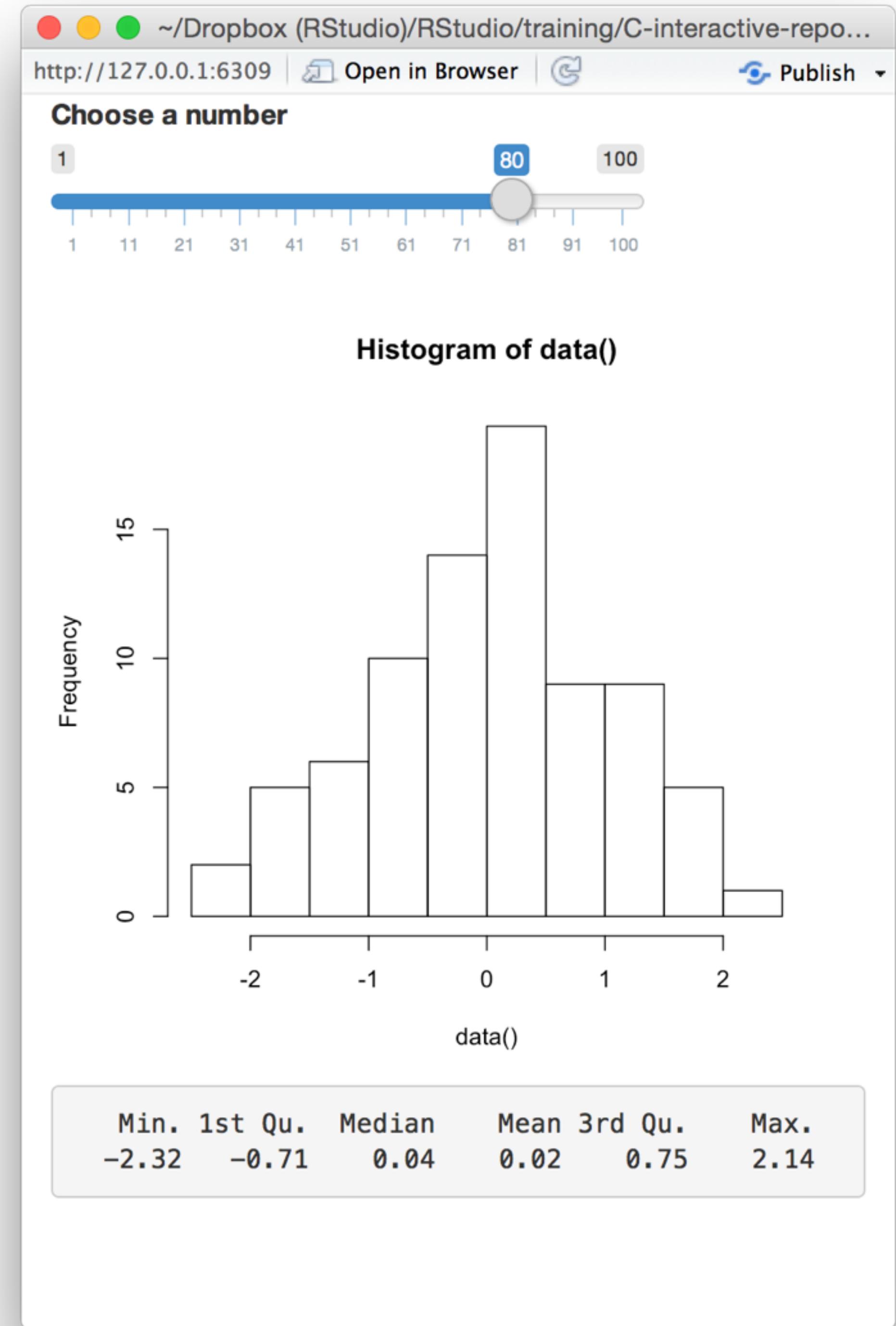
library(shiny)

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

server <- function(input, output) {
  data <- reactive({
    rnorm(input$num)
  })
  output$hist <- renderPlot({
    hist(data())
  })
  output$stats <- renderPrint({
    summary(data())
  })
}
shinyApp(ui = ui, server = server)
```







input\$num

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

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

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

A reactive expression is special in two ways

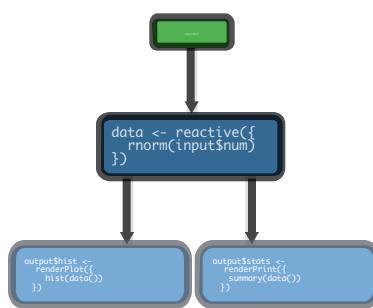
```
data()
```

- 1** You call a reactive expression like a function
- 2** Reactive expressions **cache** their values
(the expression will return its most recent value,
unless it has become invalidated)

Recap: reactive()

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

reactive() makes an **object to use** (in downstream code)



Reactive expressions are themselves **reactive**. Use them to modularize your apps.

data()

Call a reactive expression like a **function**

2

Reactive expressions **cache** their values to avoid unnecessary computation

**Prevent reactions
with isolate()**

```
# 01-two-inputs

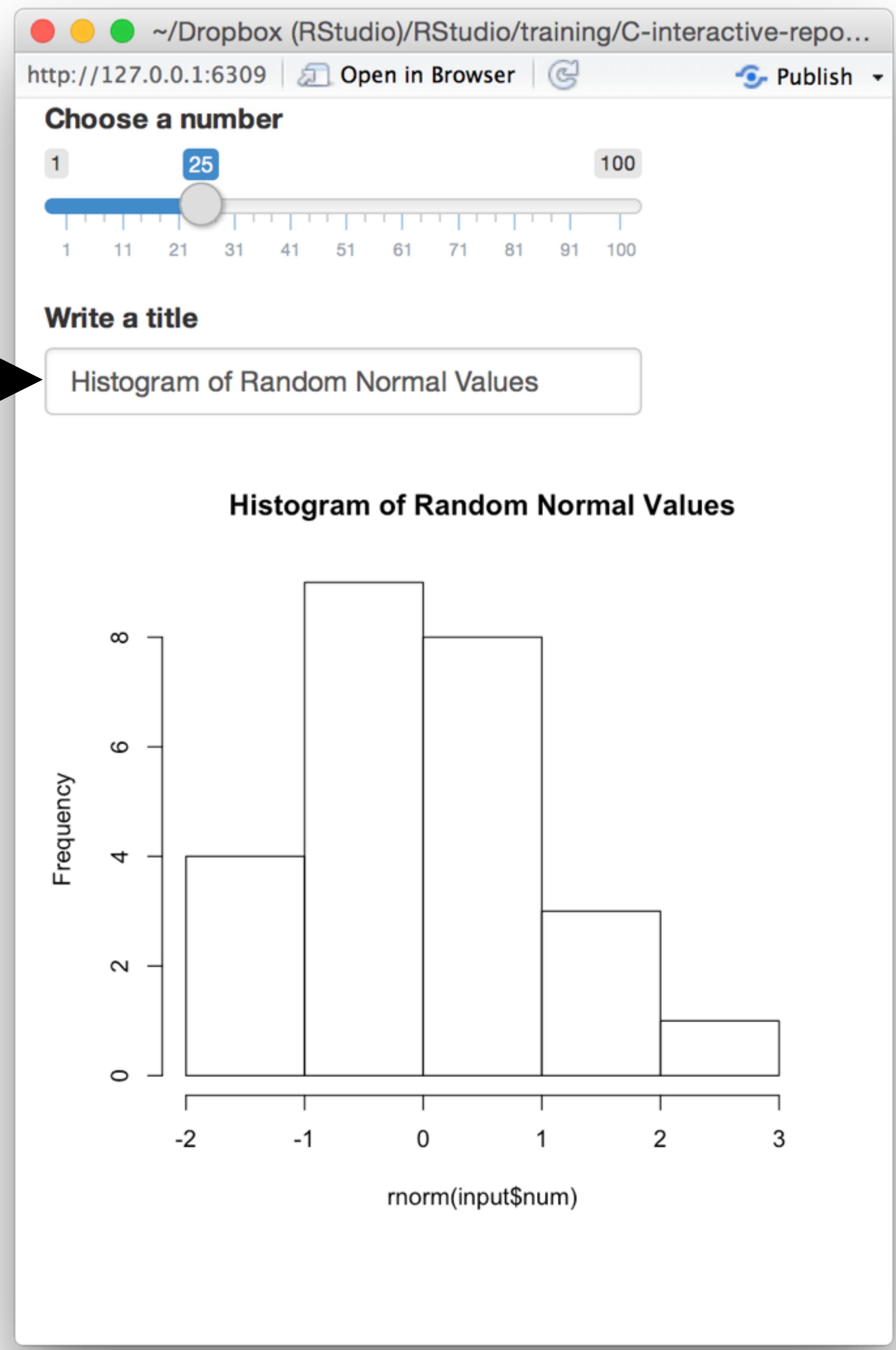
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textFieldInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num),
      main = input$title)
  })
}

shinyApp(ui = ui, server = server)
```

**Can we prevent
the title field from
updating the plot?**



isolate()

Returns the result as a non-reactive value

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

object will NOT respond to
*any reactive value in the
code*

code used to build
object

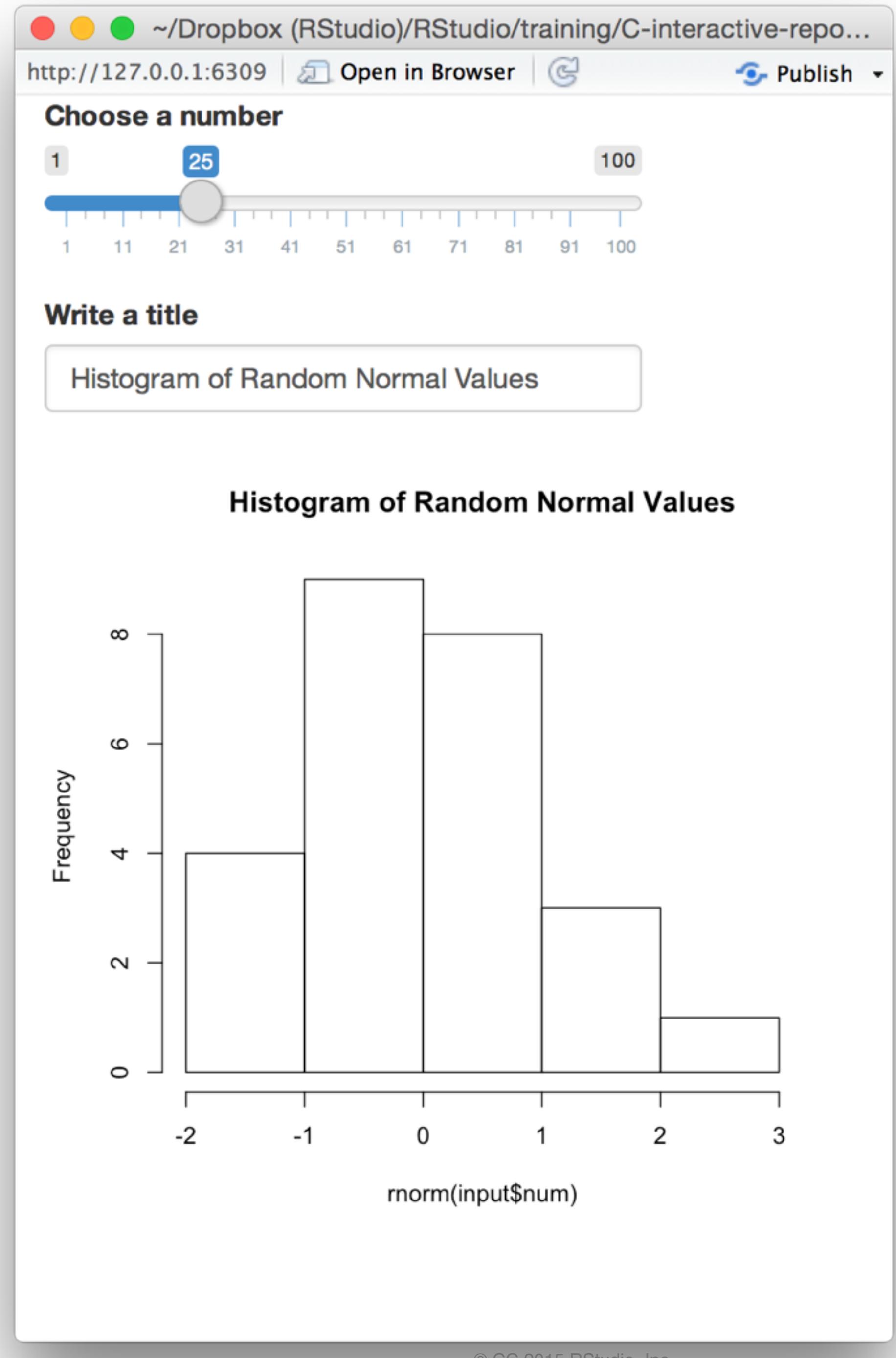
```
# 01-two-inputs

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num),
      main = input$title)
  })
}

shinyApp(ui = ui, server = server)
```



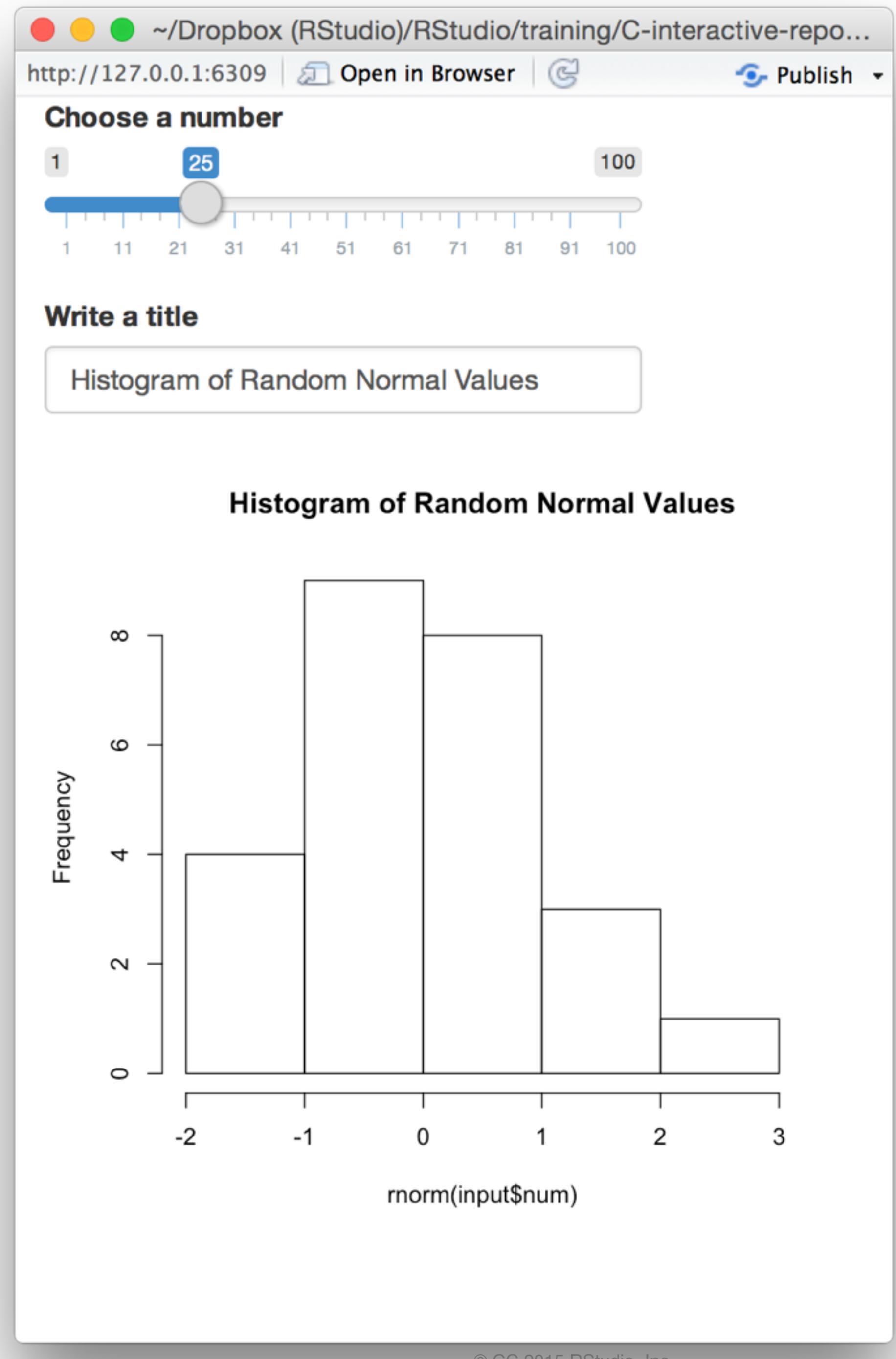
```
# 04-isolate

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num),
      main = isolate({input$title}))})
}

shinyApp(ui = ui, server = server)
```



input\$num

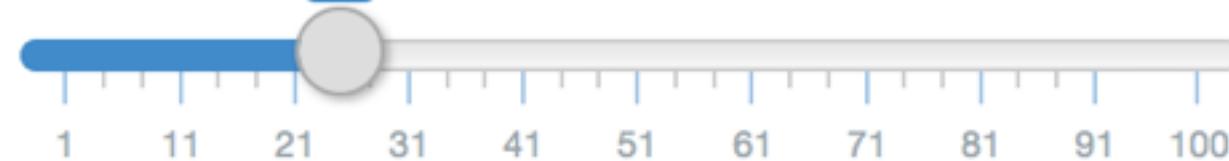
input\$title



```
output$hist <- renderPlot({  
  hist(rnorm(input$num),  
    main = isolate(input$title))  
})
```

Choose a number

1 25 100

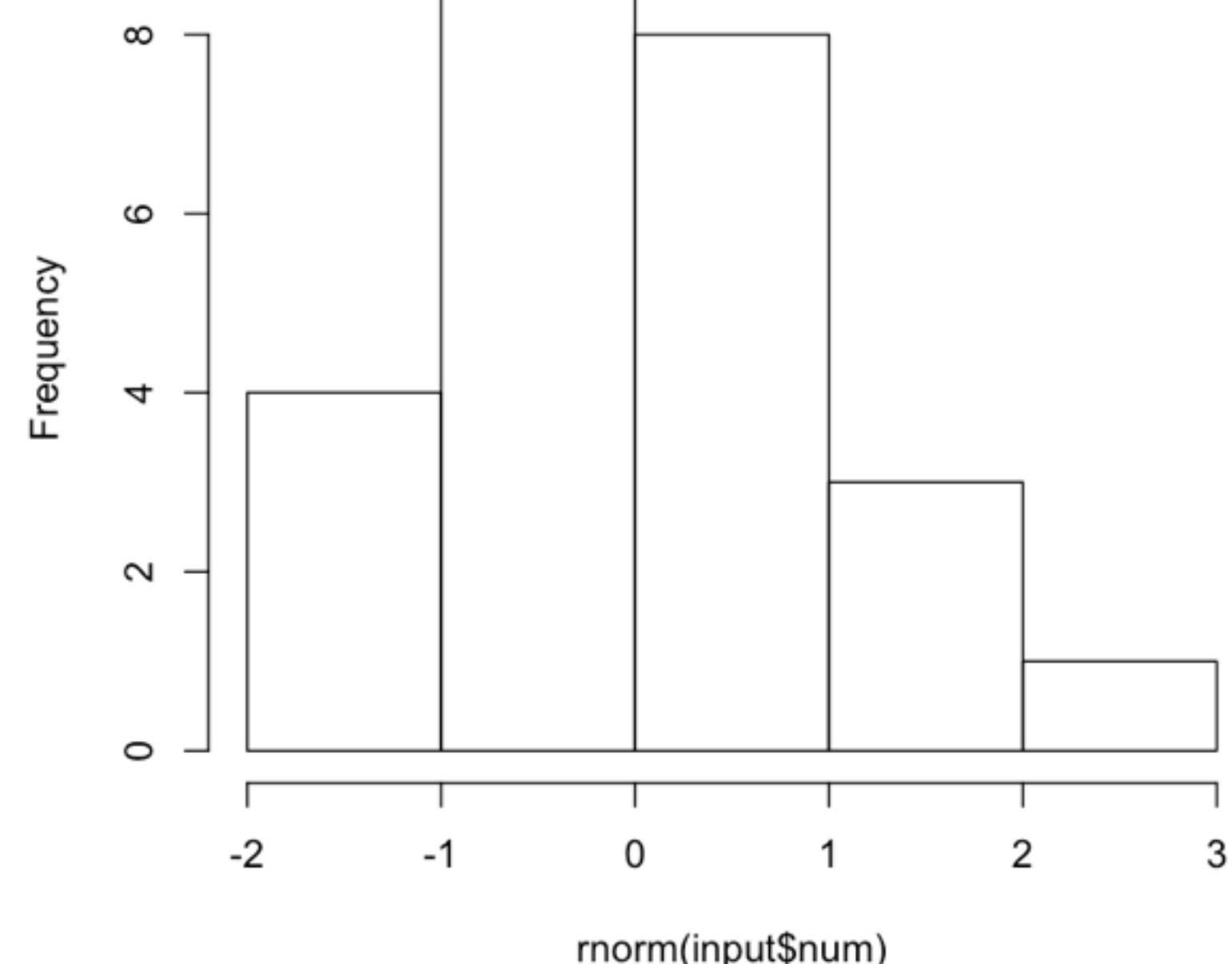


1 11 21 31 41 51 61 71 81 91 100

Write a title

Histogram of Random Normal Values

Histogram of Random Normal Values

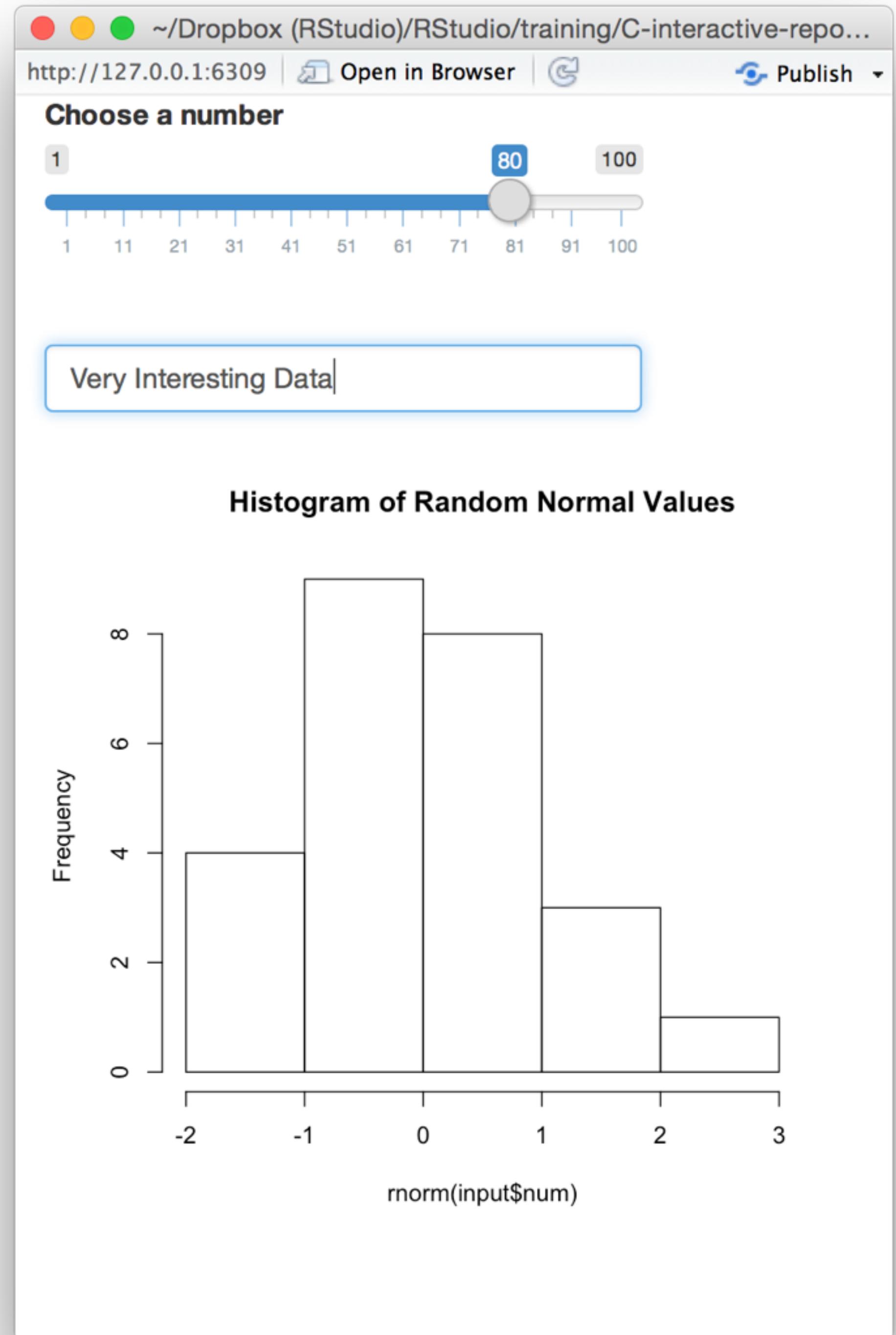


input\$num

input\$title

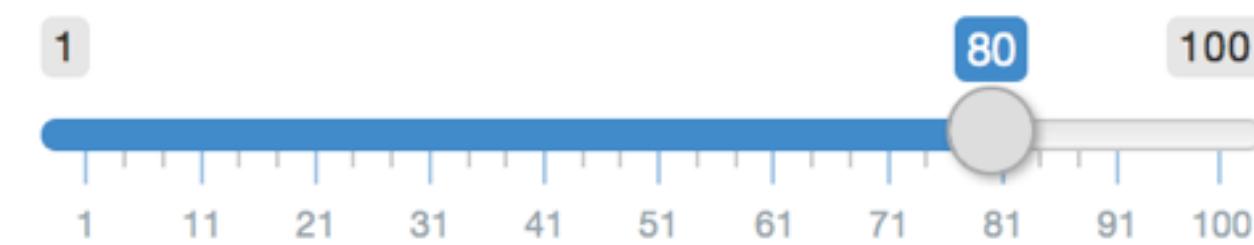


```
output$hist <- renderPlot({  
  hist(rnorm(input$num),  
    main = isolate(input$title))  
})
```



Choose a number

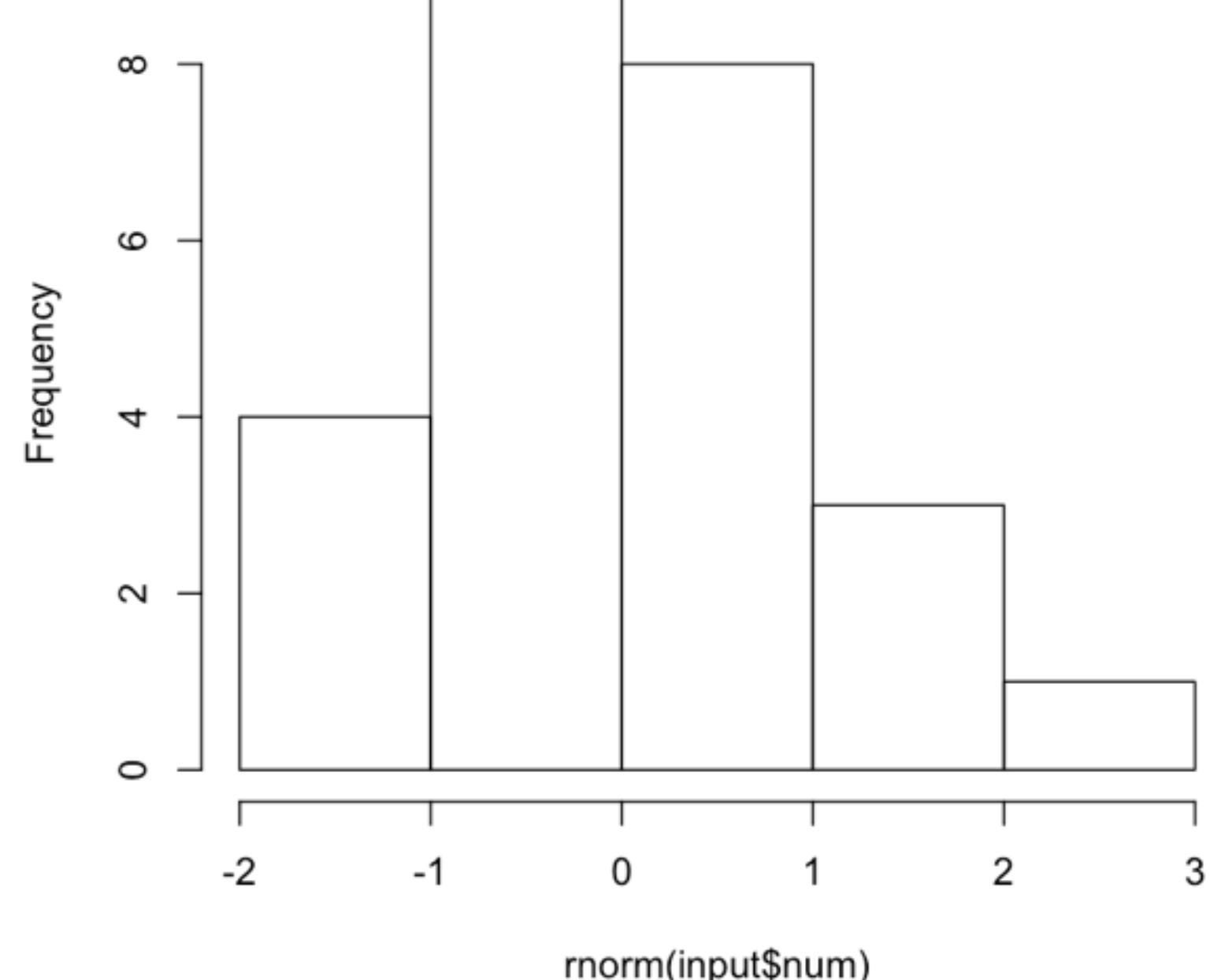
1 80 100



1 11 21 31 41 51 61 71 81 91 100

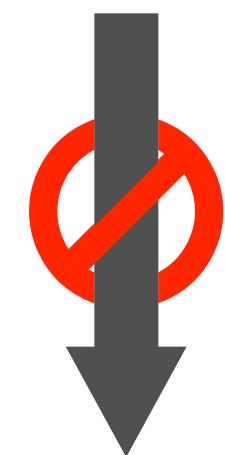
Very Interesting Data

Histogram of Random Normal Values



```
output$hist <- renderPlot({  
  hist(rnorm(input$num),  
    main = isolate(input$title))  
})
```

Recap: isolate()



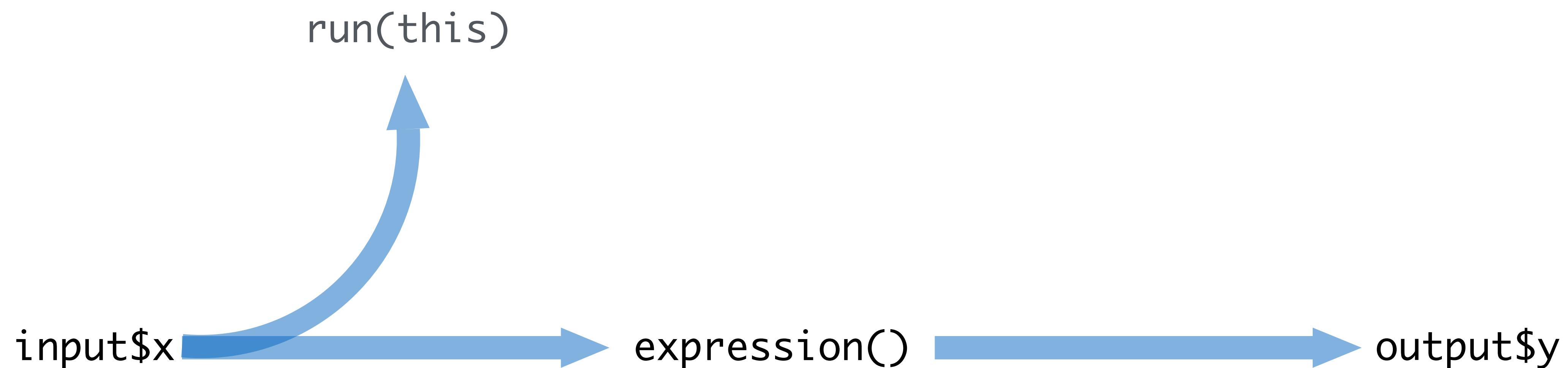
isolate() makes an **non-reactive object**



Use isolate() to treat reactive values like
normal R values

**Trigger code
with observeEvent()**

input\$x  expression  output\$y



Action buttons

An Action Button

Click Me!

input
function

input name
(for internal use)

label to
display

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

Notice:
Id not ID

```
# 05ActionButton
```

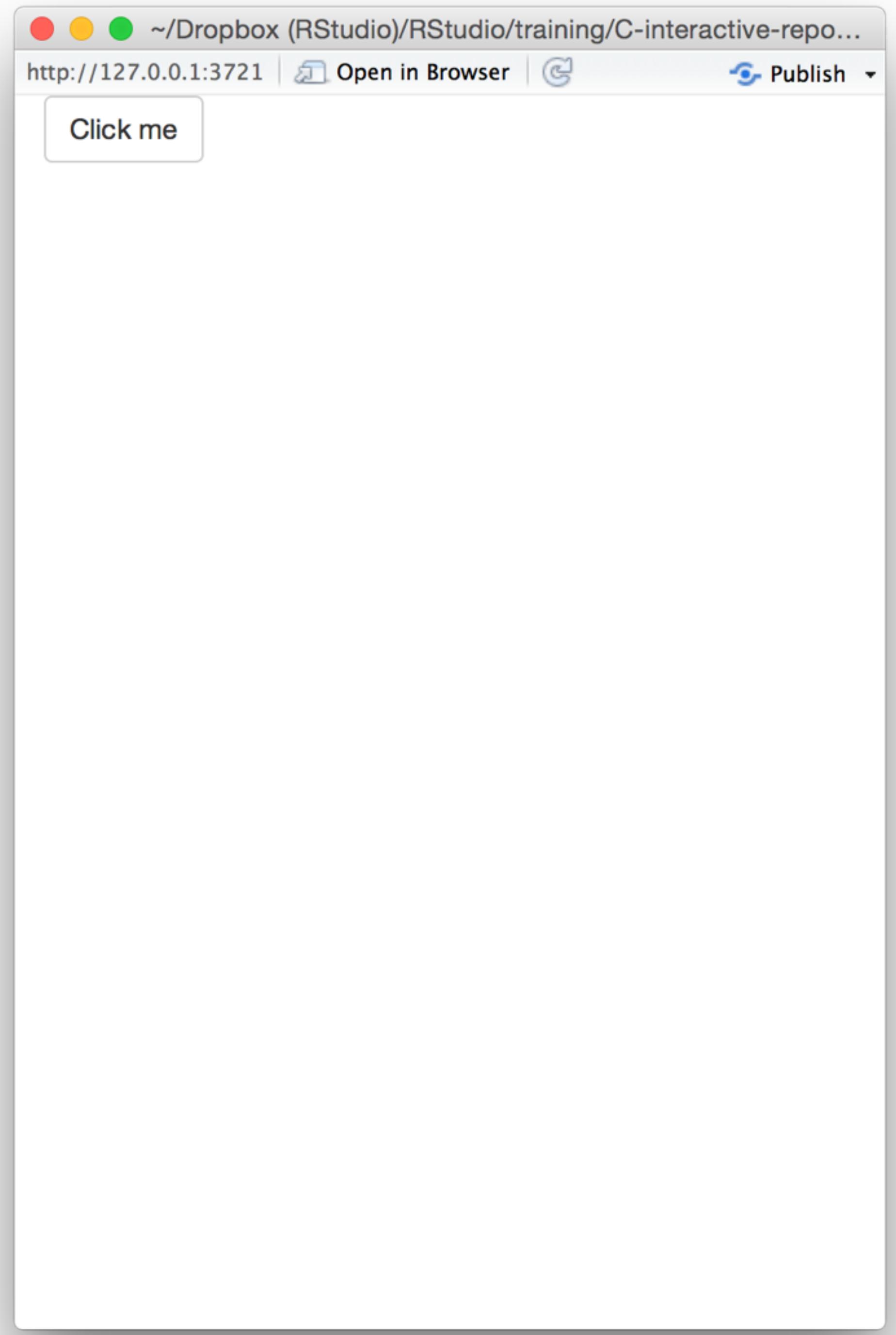
```
library(shiny)
```

```
ui <- fluidPage(  
  actionButton(inputId = "clicks",  
               label = "Click me")  
)
```

```
server <- function(input, output) {
```

```
}
```

```
shinyApp(ui = ui, server = server)
```



observeEvent()

Triggers code to run on server

```
observeEvent(input$clicks, { print(input$clicks) })
```

reactive value(s) to respond to

(observer invalidates ONLY when this value changes)

code block to run whenever observer is invalidated

note: observer treats this code as if it has been isolated with isolate()

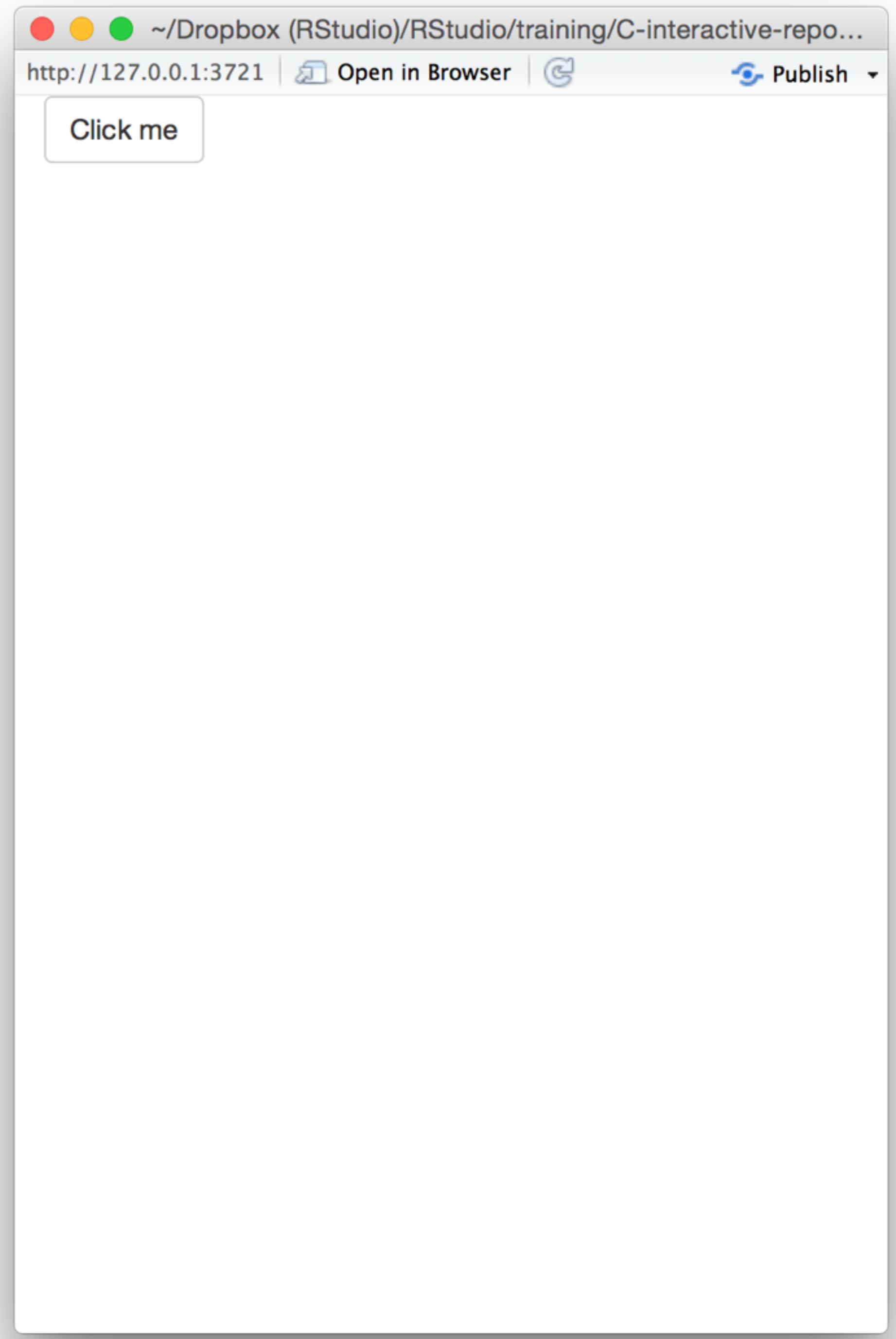
```
# 05ActionButton
```

```
library(shiny)
```

```
ui <- fluidPage(  
  actionButton(inputId = "clicks",  
    label = "Click me")  
)
```

```
server <- function(input, output) {  
  observeEvent(input$clicks, {  
    print(as.numeric(input$clicks))  
  })  
}
```

```
shinyApp(ui = ui, server = server)
```



Action buttons article

<http://shiny.rstudio.com/articles/action-buttons.html>

The screenshot shows a web browser window with the title bar "Shiny - Using Action Button X". The address bar contains the URL "shiny.rstudio.com/articles/action-buttons.html". The page itself has a blue header with the "Shiny by RStudio" logo and a search bar. On the left, there's a sidebar with links: OVERVIEW, TUTORIAL, ARTICLES (which is selected and highlighted in blue), GALLERY, REFERENCE, DEPLOY, and HELP. The main content area has a large heading "Using Action Buttons" and a sub-heading "ADDED: 26 MAR 2015". Below this, a paragraph explains that the article describes five patterns for using Shiny's action buttons and action links, noting they are intended for exclusive use with `observeEvent()` or `eventReactive()`. A section titled "How action buttons work" follows, with a note about creating them with `actionButton()` and `actionLink()`, each taking two arguments: `inputId` and `label`. It includes a code block:

```
actionButton("button", "An action button")
actionLink("button", "An action link")
```

Below the code, a note states: "An action button appears as a button in your app." A button labeled "An action button" is shown. At the bottom, it says: "An action link appears as a hyperlink, but behaves in the same way as an action button."

observe()

Also triggers code to run on server.

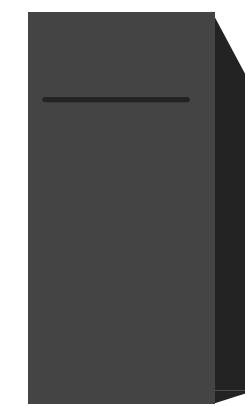
Uses same syntax as render*, reactive(), and isolate()

```
observe({ print(input$clicks) })
```

observer will respond to
*every reactive value in the
code*

code block to run
whenever observer is
invalidated

Recap: observeEvent()



observeEvent() **triggers code to run on the server**

```
observeEvent(input$clicks, { print(input$clicks) })
```

reactive value(s)
to respond to

A snippet of R code showing an observeEvent call. A callout box points from the text "reactive value(s) to respond to" to the "input\$clicks" argument in the code.

Specify precisely which reactive values should invalidate the observer

observe()

Use observe() for a more implicit syntax

Delay reactions with eventReactive()

```
# 07-eventReactive

library(shiny)

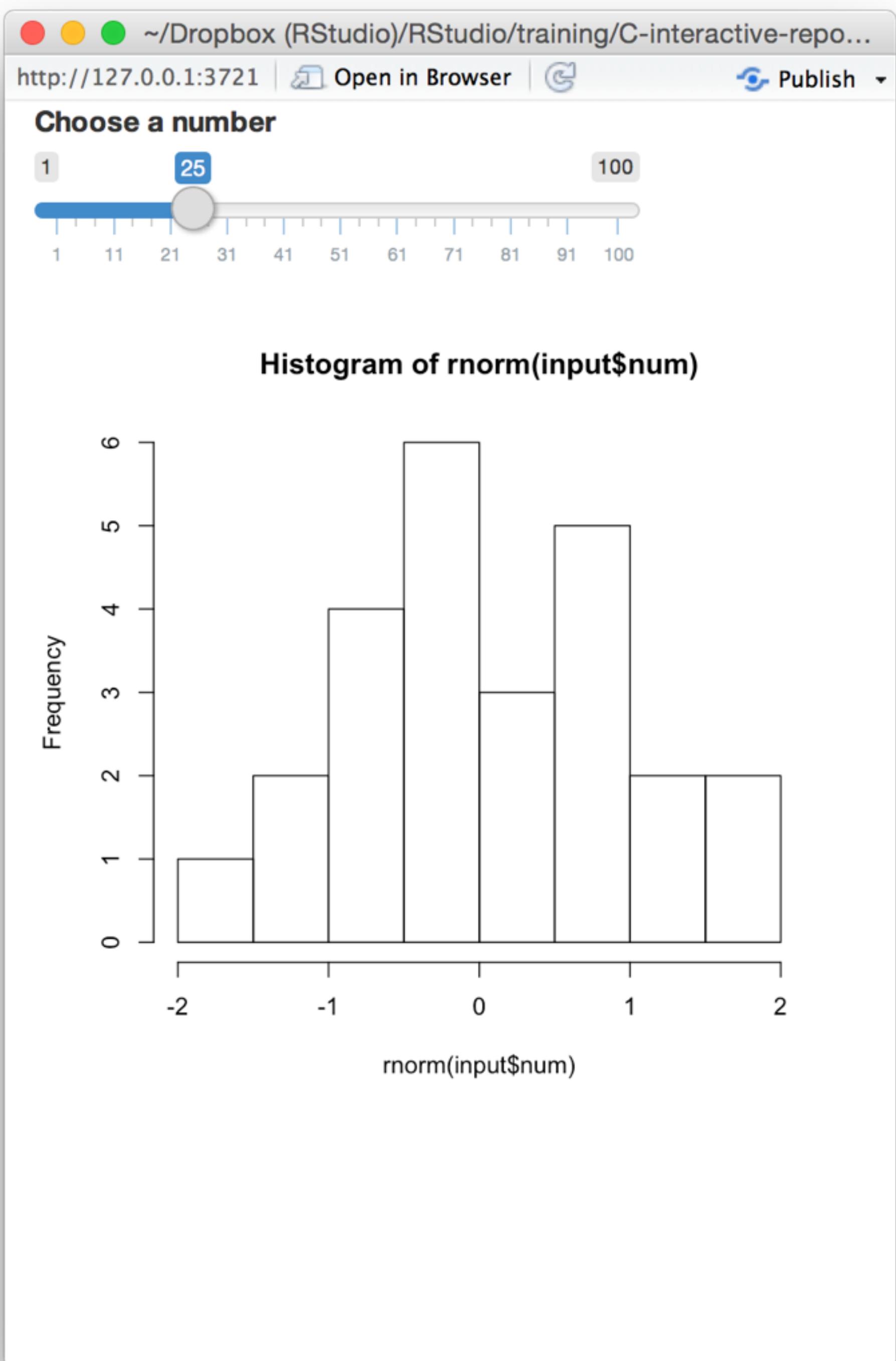
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),

  plotOutput("hist")
)

server <- function(input, output) {

  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



```
# 07-eventReactive

library(shiny)

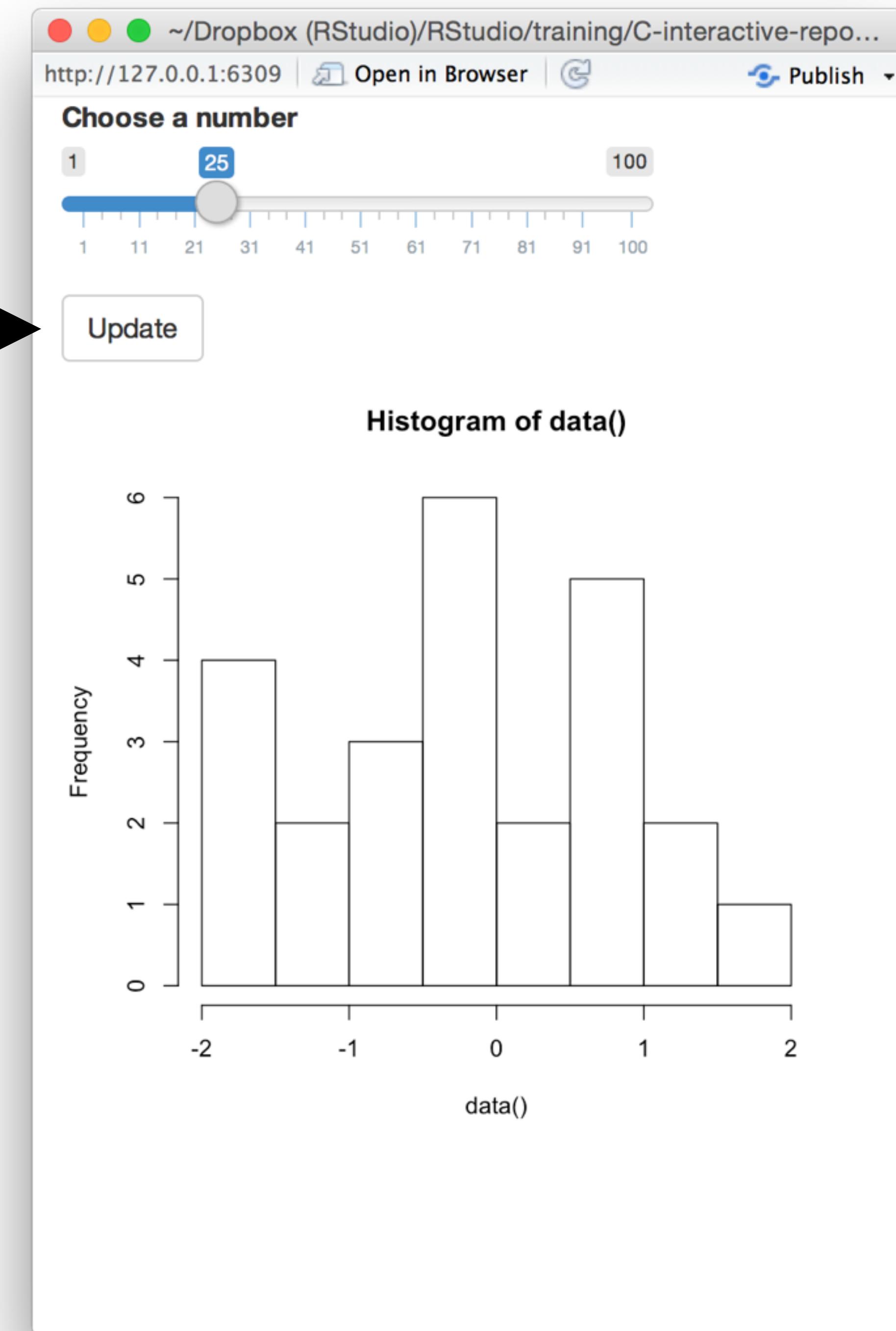
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  actionButton(inputId = "go",
    label = "Update"),
  plotOutput("hist")
)

server <- function(input, output) {

  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```

Can we prevent the graph from updating until we hit the button?



eventReactive()

A reactive expression that only responds to specific values

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

reactive value(s) to respond to

code used to build (and rebuild) object

note: expression treats this code as if it has been isolated with isolate()

(expression invalidates ONLY when this value changes)

```
# 07-eventReactive

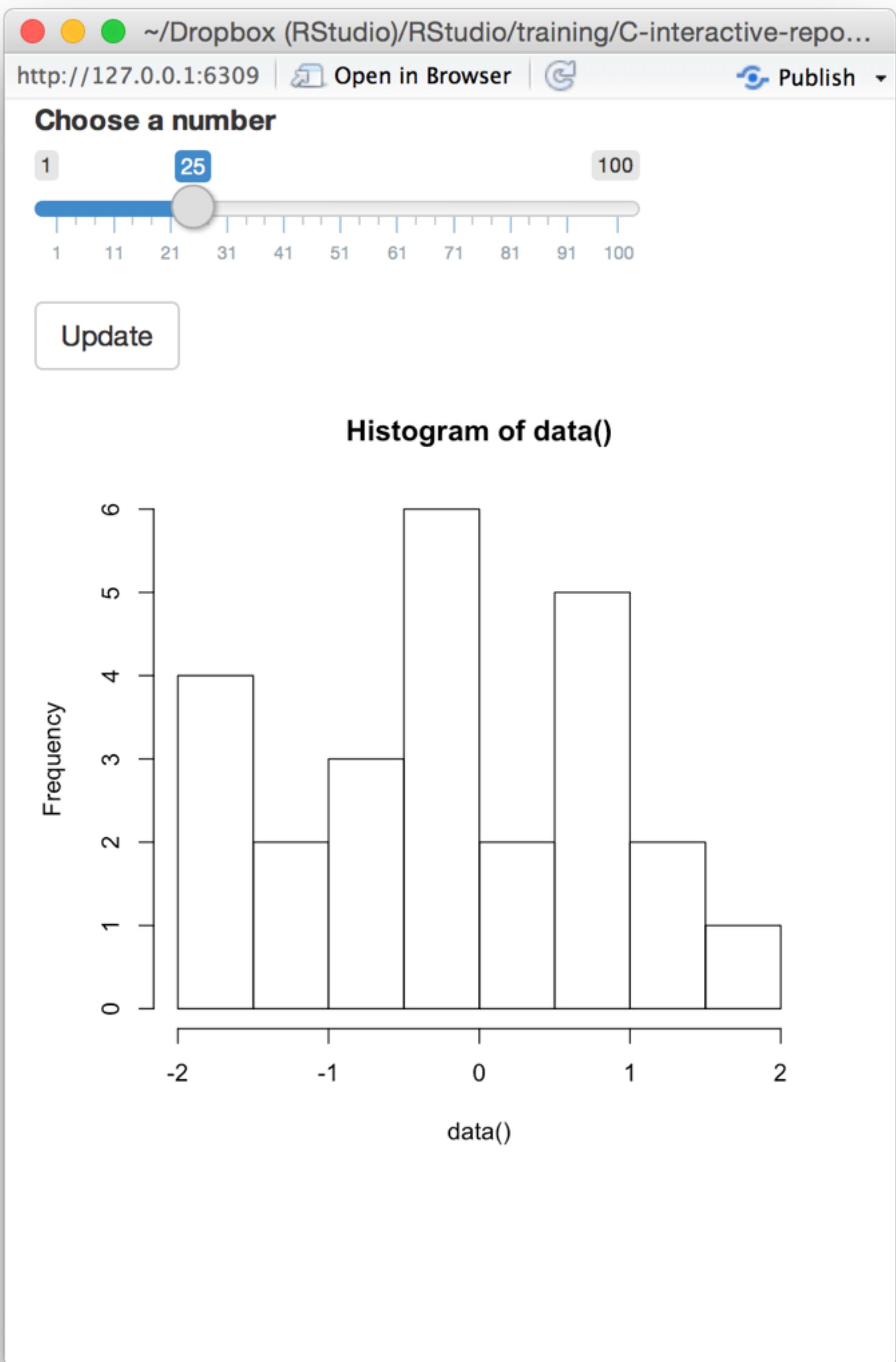
library(shiny)

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

server <- function(input, output) {

  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



```
# 07-eventReactive

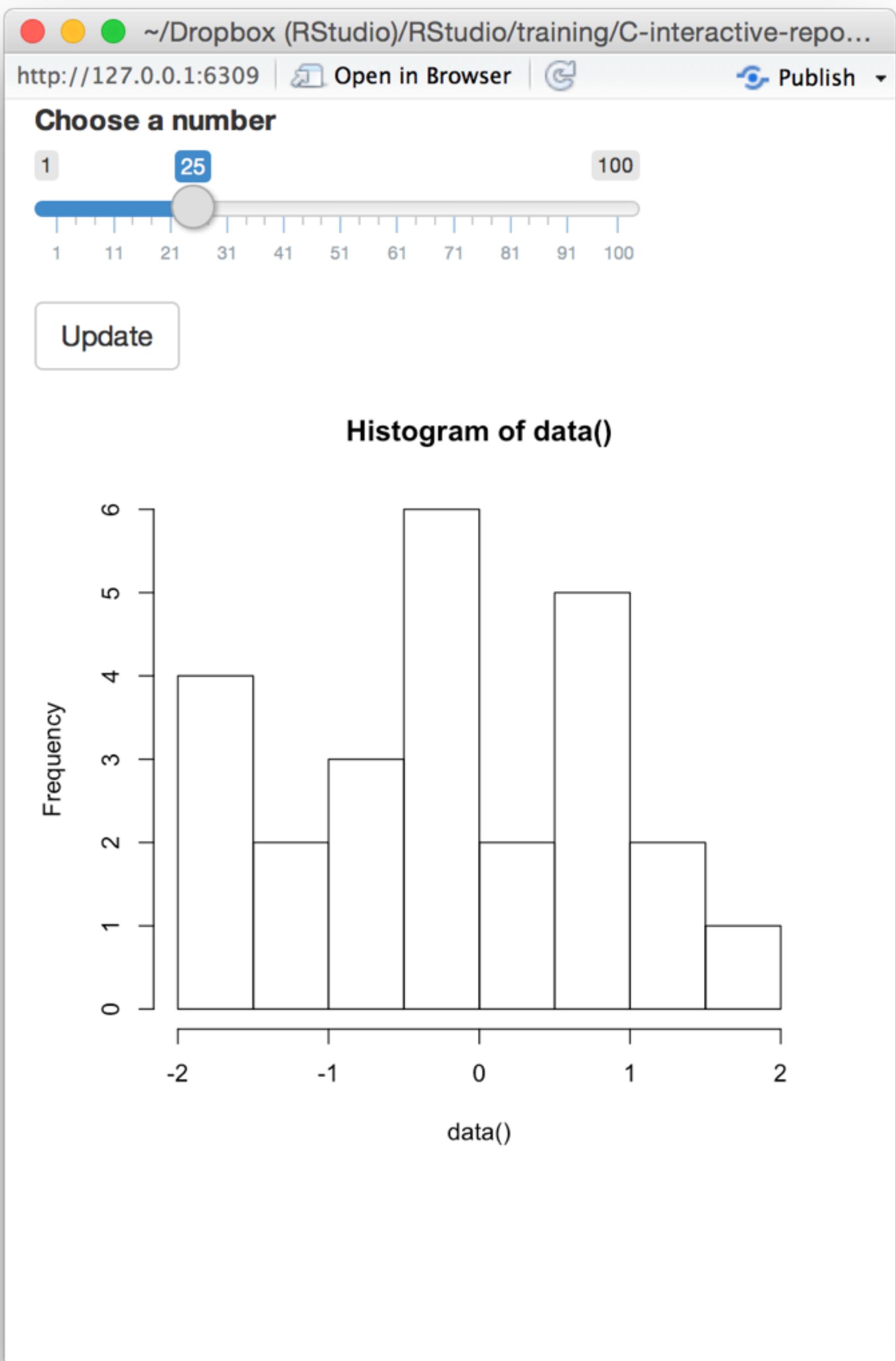
library(shiny)

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

server <- function(input, output) {
  data <- eventReactive(input$go, {
    })

  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



```
# 07-eventReactive

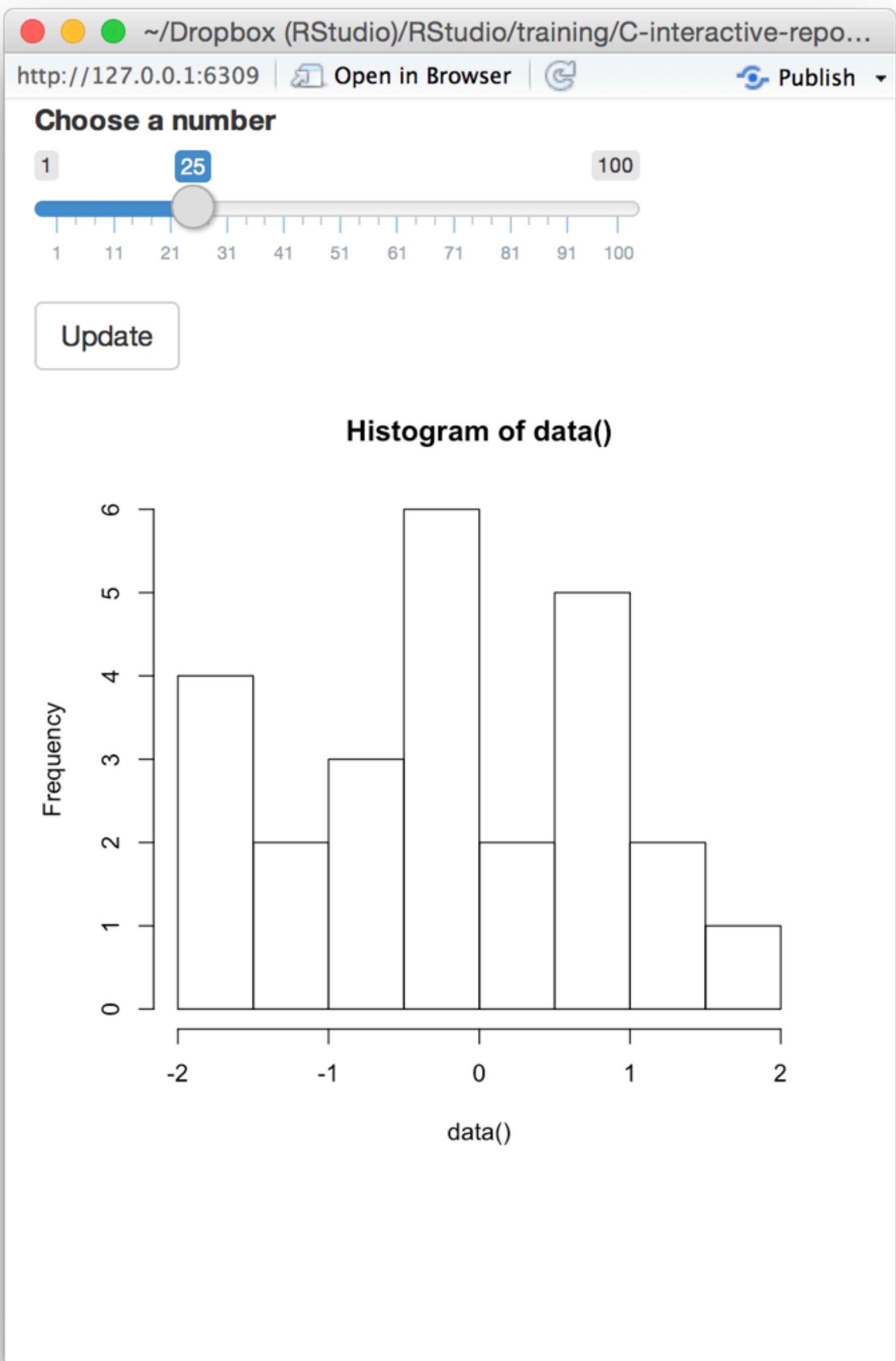
library(shiny)

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

server <- function(input, output) {
  data <- eventReactive(input$go, {
    })

  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



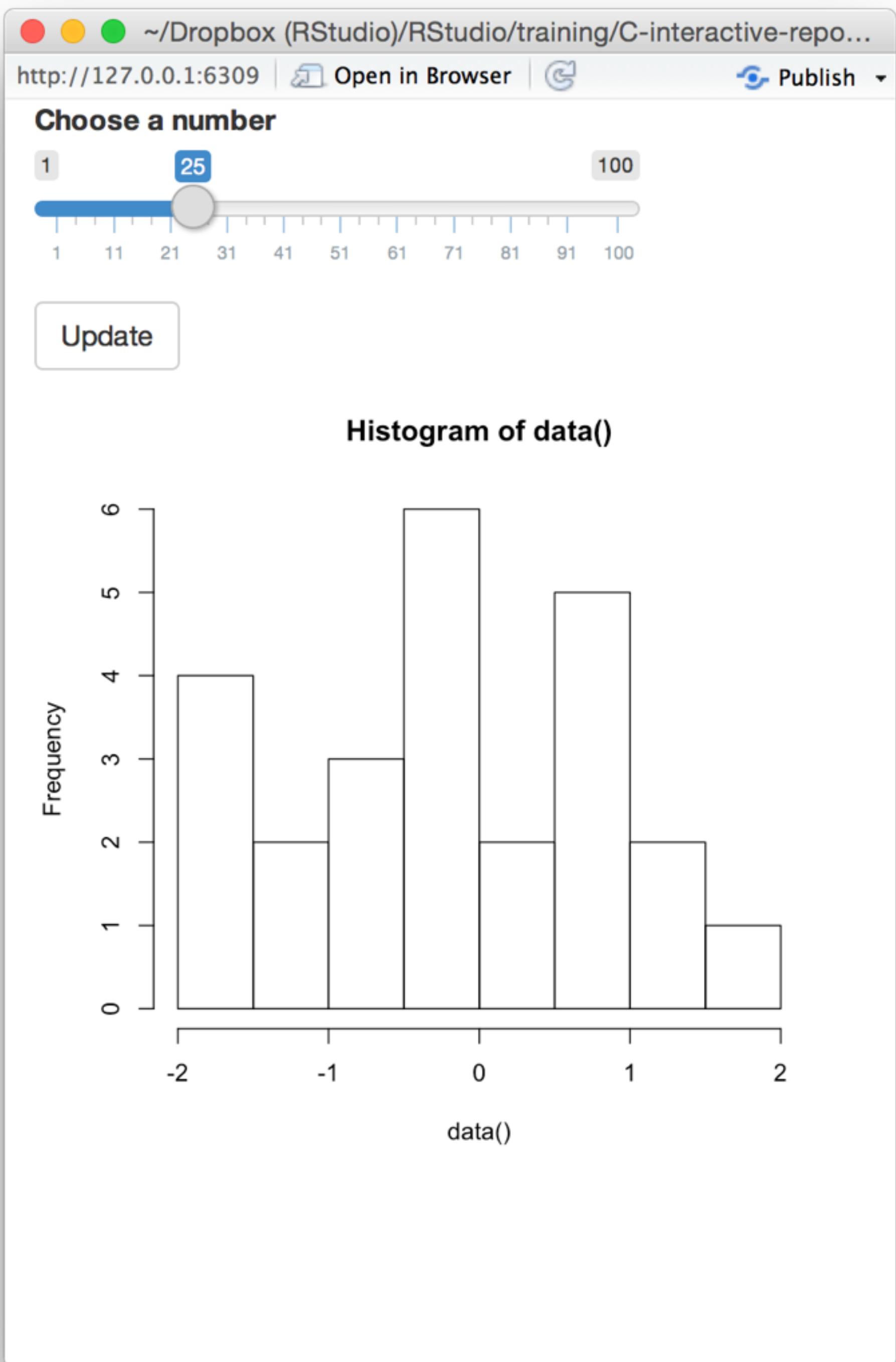
```
# 07-eventReactive

library(shiny)

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

server <- function(input, output) {
  data <- eventReactive(input$go, {
    rnorm(input$num)
  })
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



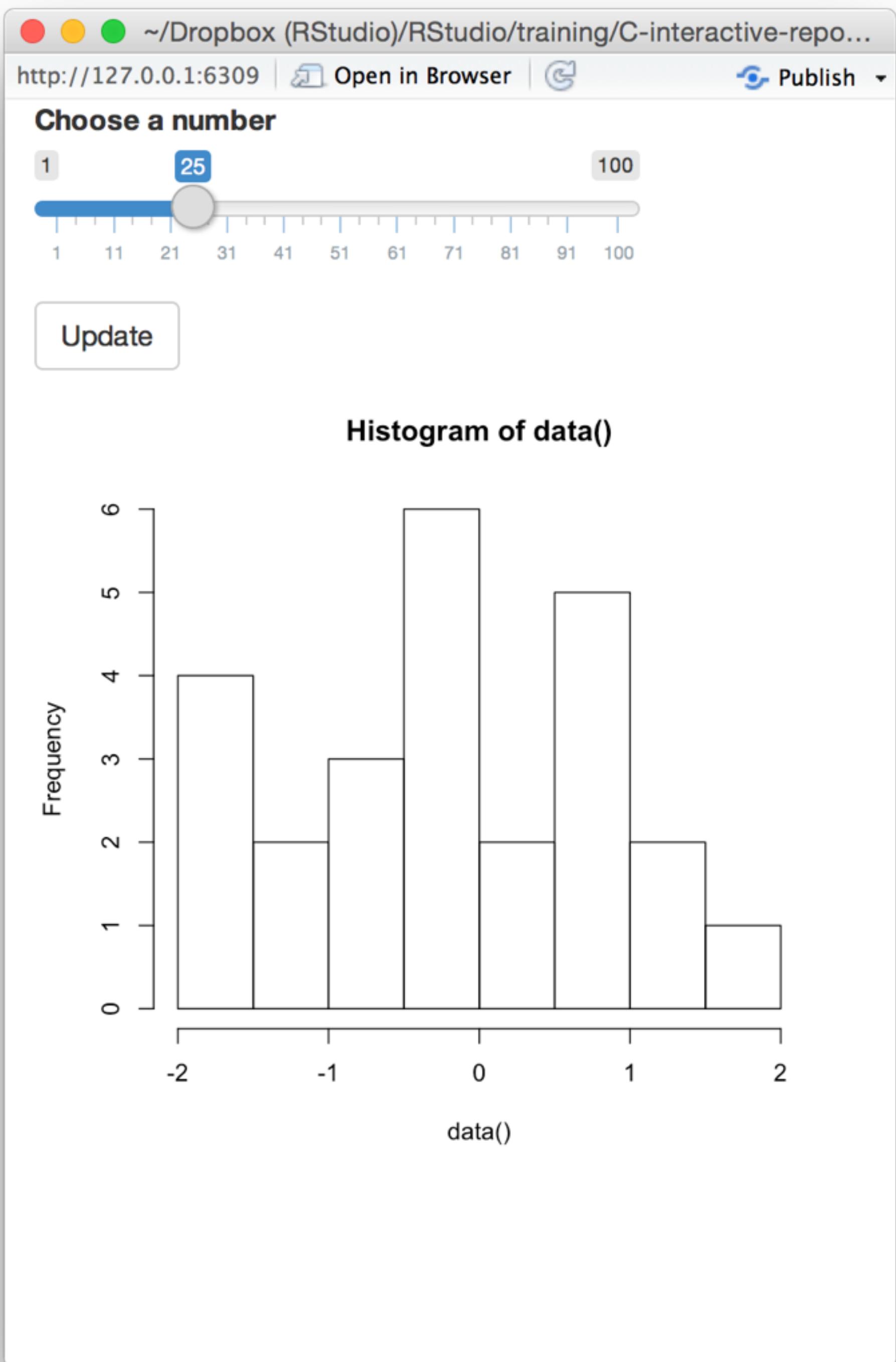
```
# 07-eventReactive

library(shiny)

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

server <- function(input, output) {
  data <- eventReactive(input$go, {
    rnorm(input$num)
  })
  output$hist <- renderPlot({
    hist(data())
  })
}

shinyApp(ui = ui, server = server)
```

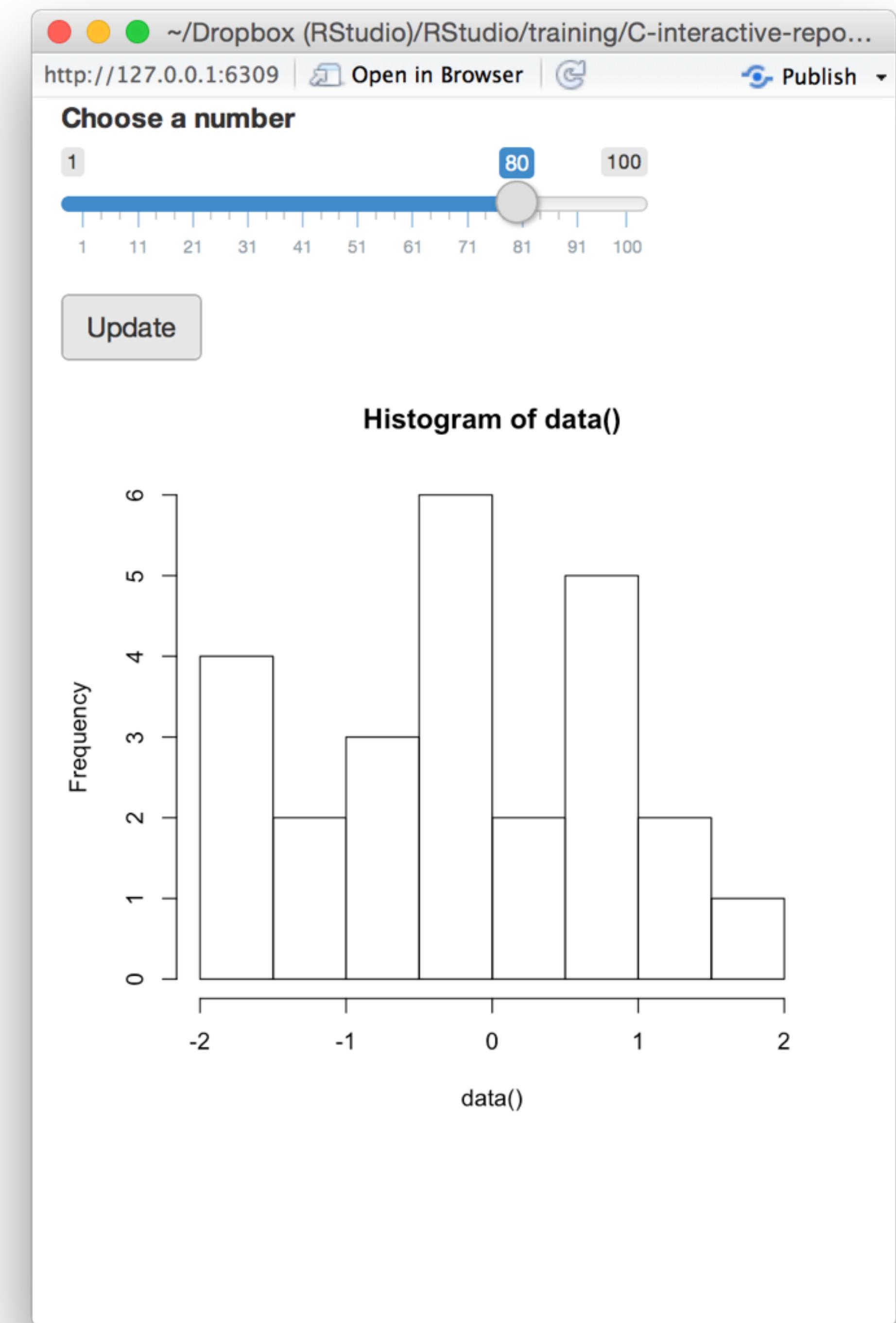


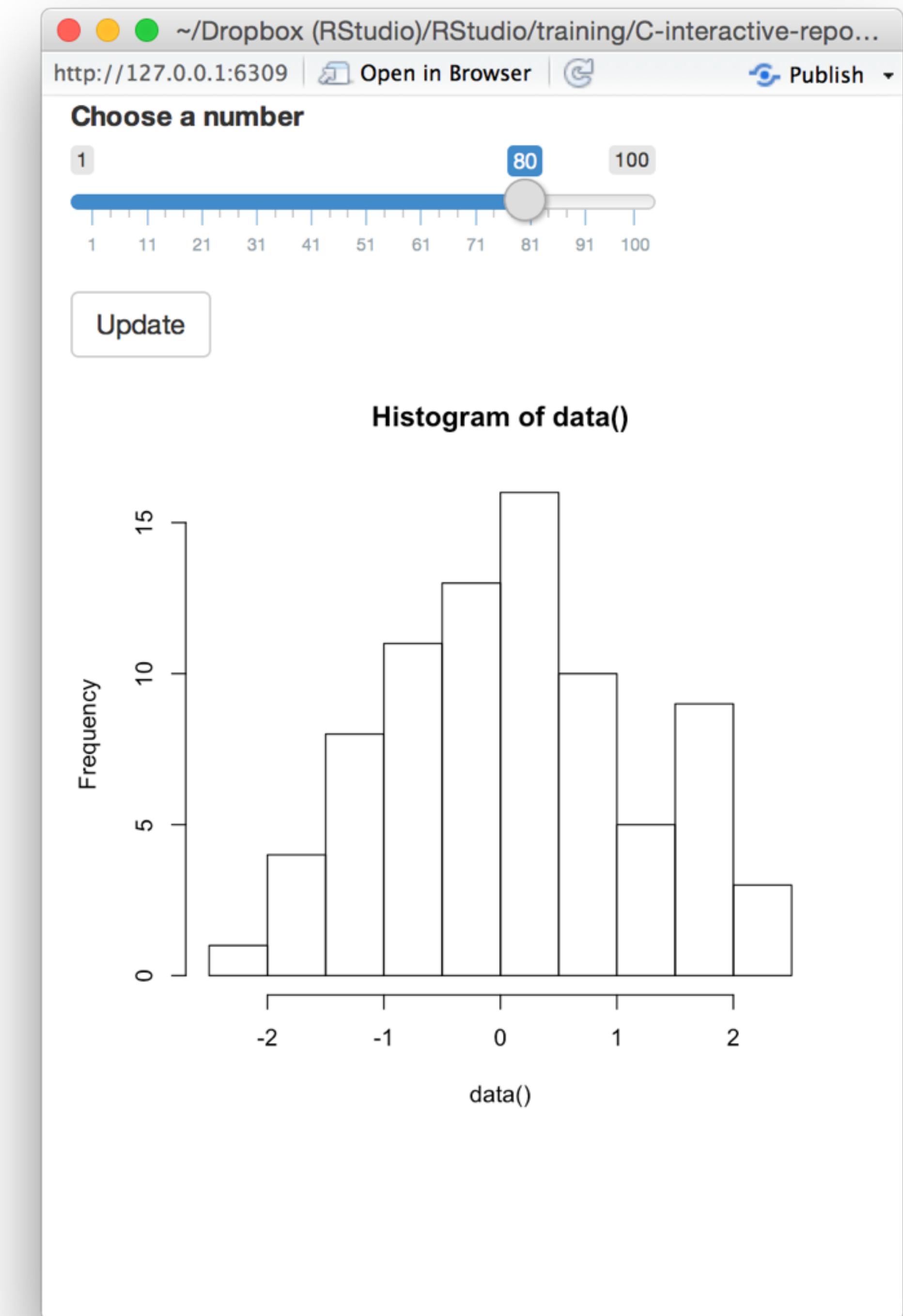
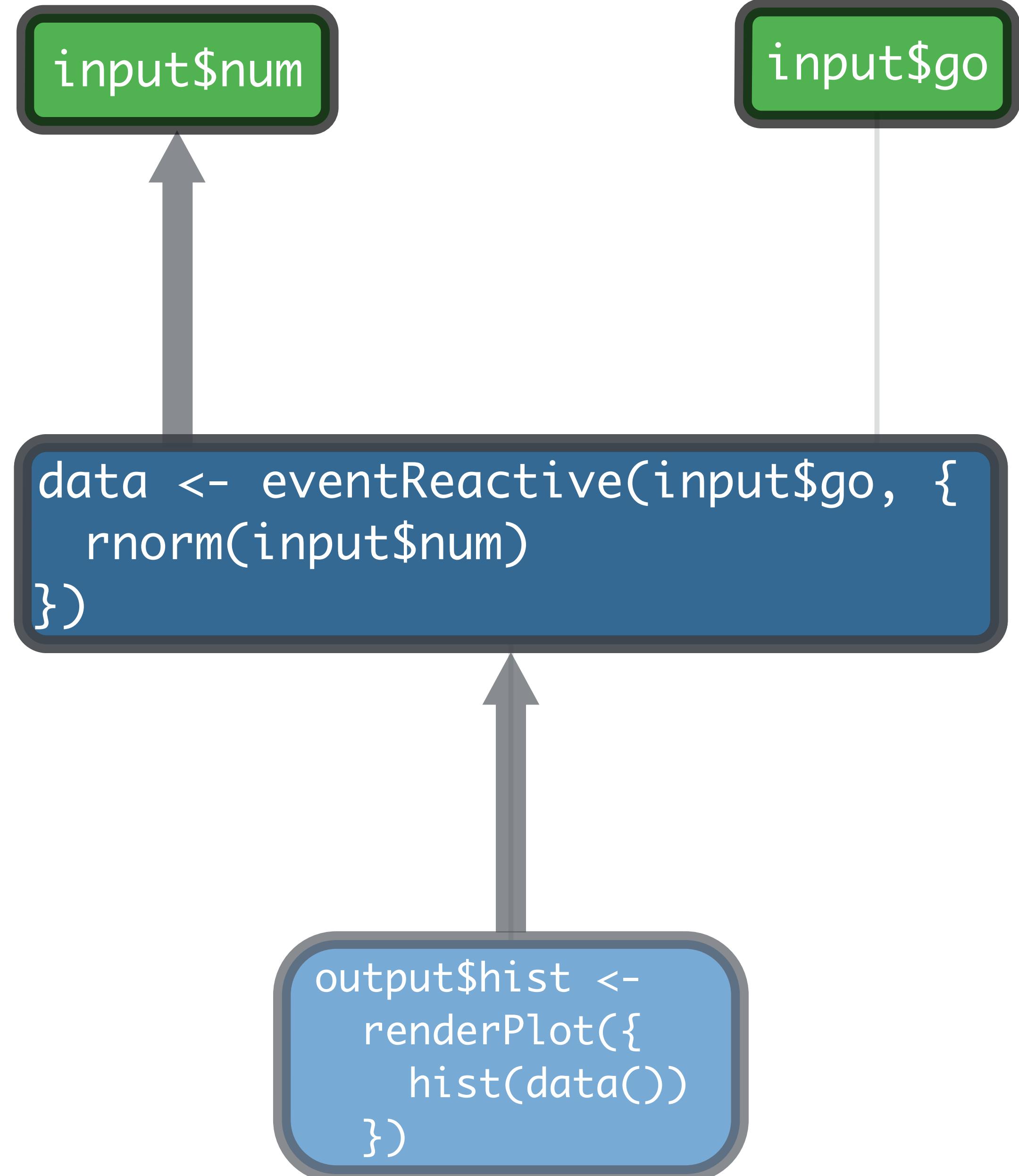
```
input$num
```

```
input$go
```

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

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





Recap: eventReactive()

Update

Use `eventReactive()` to **delay reactions**

`data()`

`eventReactive()` creates a **reactive expression**

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

reactive value(s)
to respond to

You can specify **precisely** which reactive values should invalidate the expression

Manage state with reactiveValues()

Input values

The input value changes whenever a user changes the input.

Choose a number

A slider input with a blue handle set to the value 25. The slider scale ranges from 1 to 100 with major tick marks every 10 units. The current value is displayed as 25 in a blue box above the slider.

input\$num = 25

Choose a number

A slider input with a blue handle set to the value 50. The slider scale ranges from 1 to 100 with major tick marks every 10 units. The current value is displayed as 50 in a blue box above the slider.

input\$num = 50

Choose a number

A slider input with a blue handle set to the value 75. The slider scale ranges from 1 to 100 with major tick marks every 10 units. The current value is displayed as 75 in a blue box above the slider.

input\$num = 75

You cannot set these
values in your code

reactiveValues()

Creates a list of reactive values to manipulate programmatically

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

(optional) elements
to add to the list

```

# 08-reactiveValues

library(shiny)

ui <- fluidPage(
  actionButton(inputId = "norm", label = "Normal"),
  actionButton(inputId = "unif", label = "Uniform"),
  plotOutput("hist")
)

server <- function(input, output) {

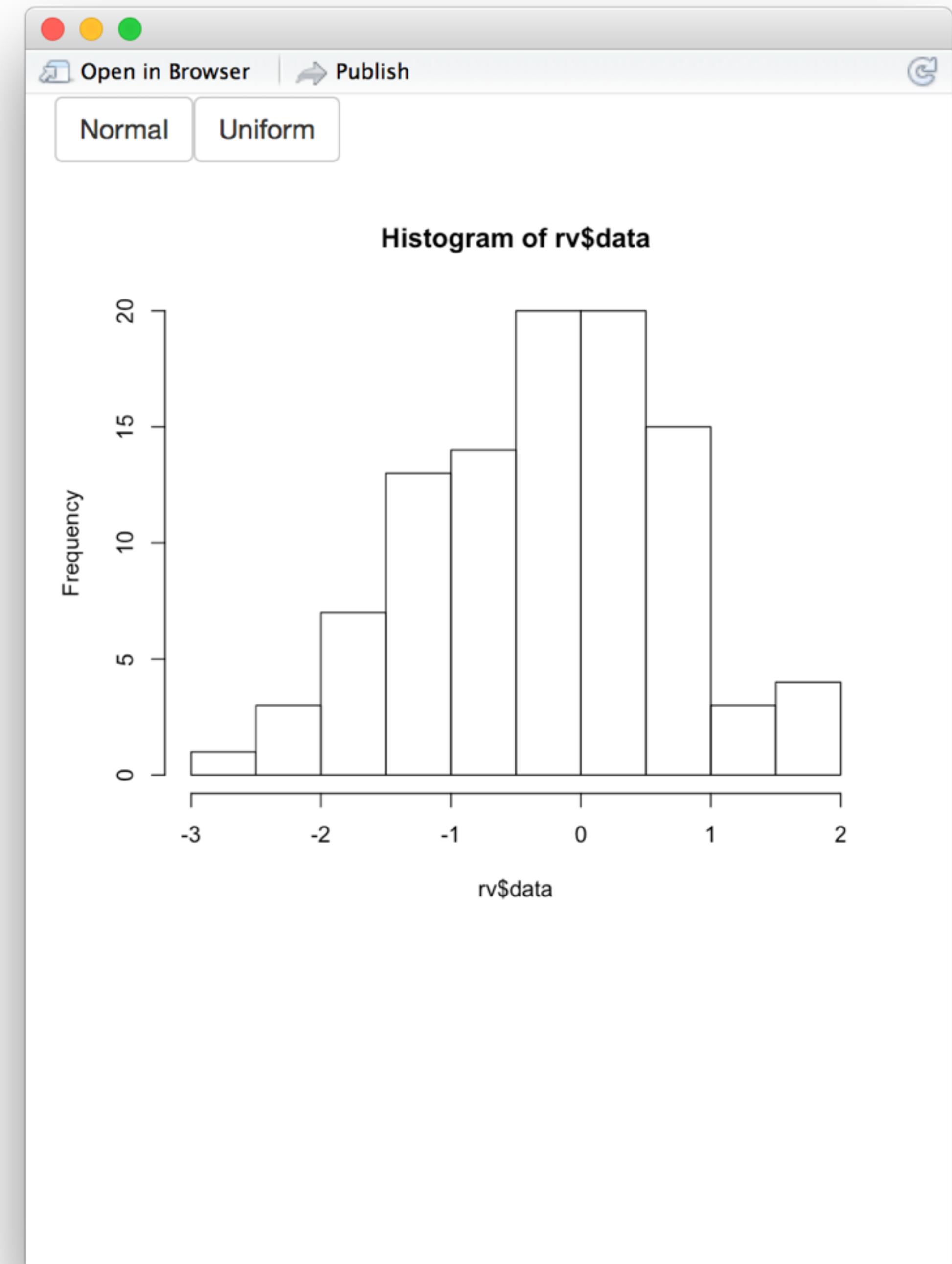
  rv <- reactiveValues(data = rnorm(100))

  observeEvent(input$norm, { rv$data <- rnorm(100) })
  observeEvent(input$unif, { rv$data <- runif(100) })

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

shinyApp(ui = ui, server = server)

```

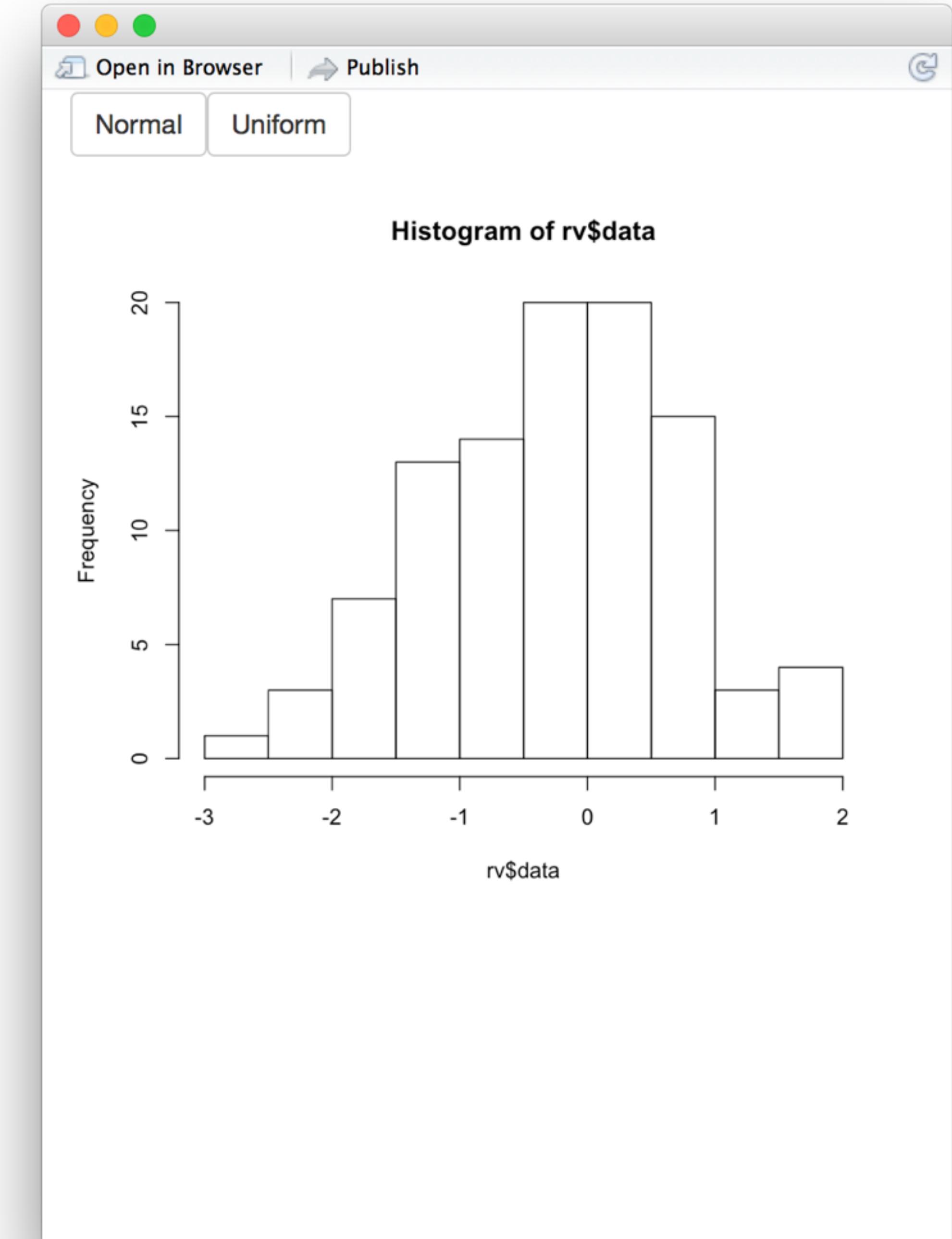


```
input$norm
```

```
rv$data  
rnorm(100)
```

```
input$unif
```

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

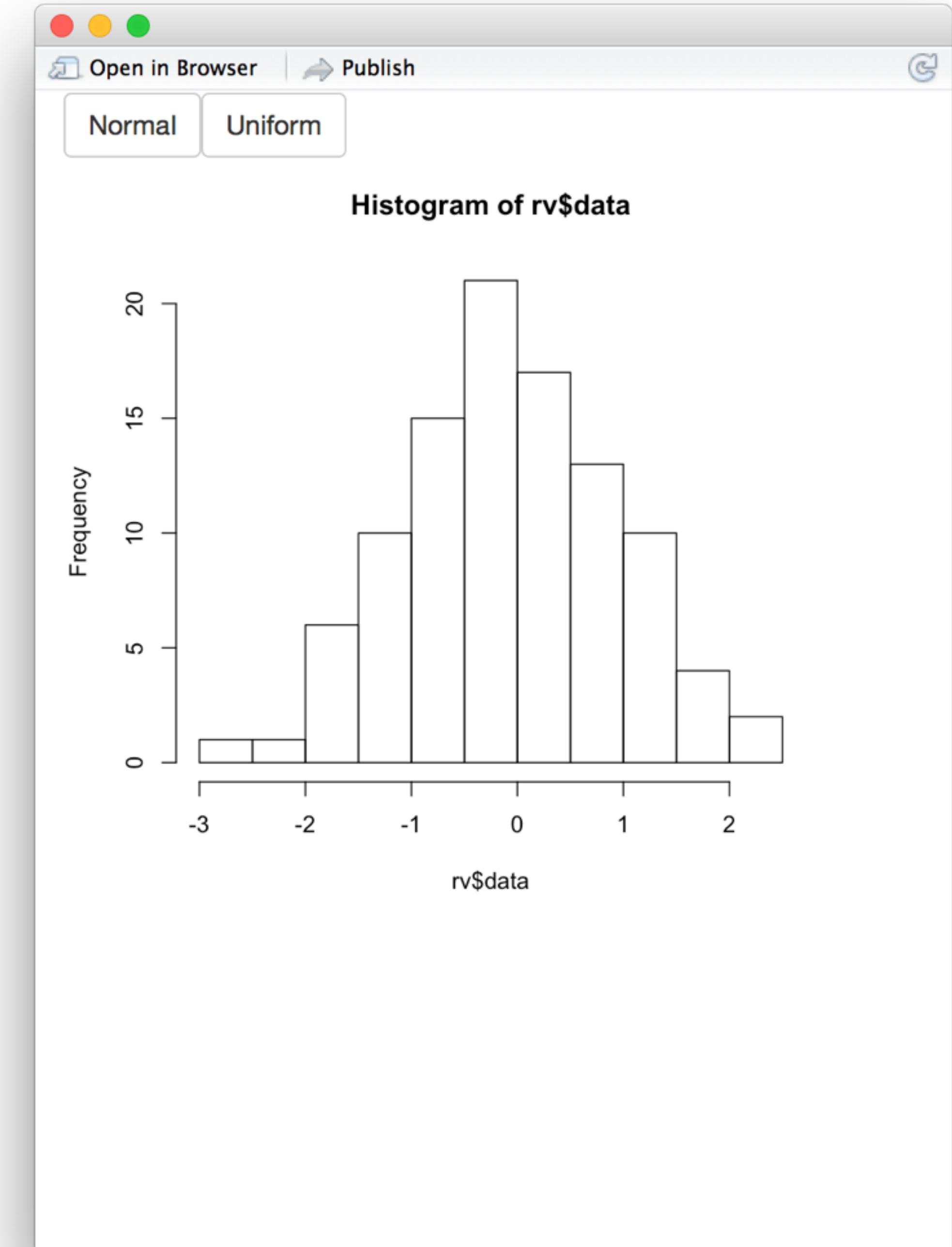


```
input$norm
```

```
rv$data  
rnorm(100)
```

```
input$unif
```

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

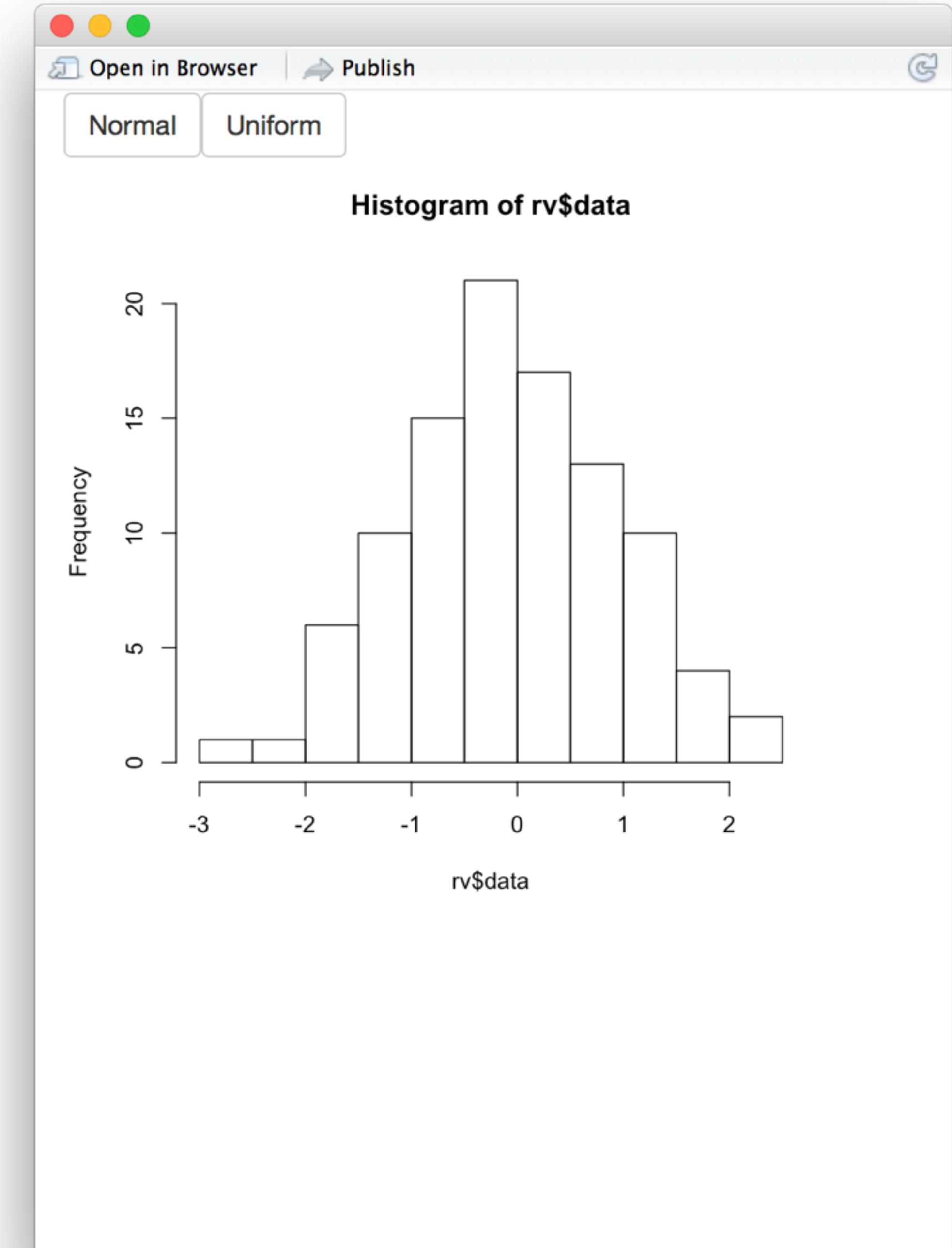


```
input$norm
```

```
rv$data  
runif(100)
```

```
input$unif
```

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

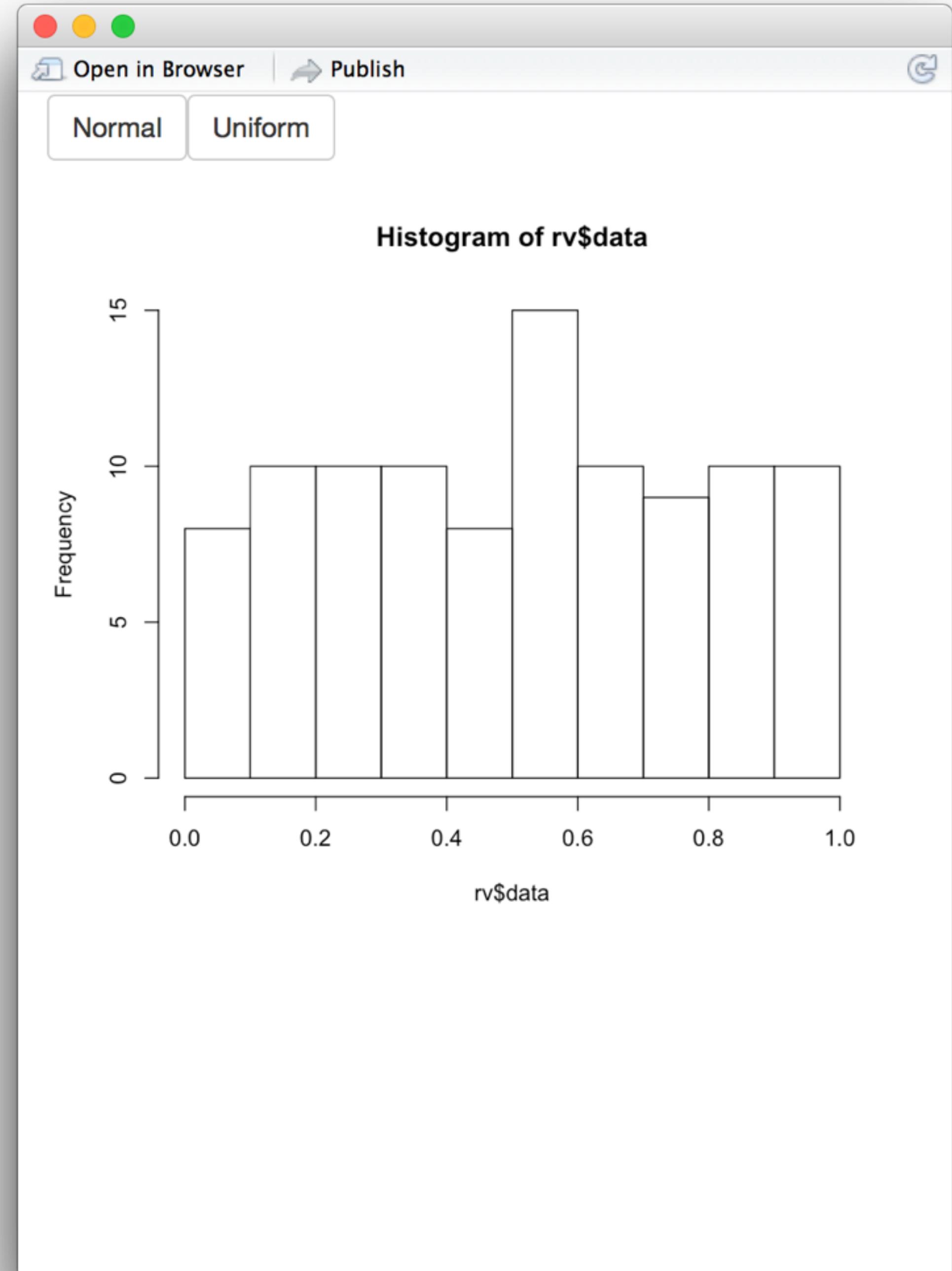


input\$norm

rv\$data
runif(100)

input\$unif

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



```

# 08-reactiveValues

library(shiny)

ui <- fluidPage(
  actionButton(inputId = "norm", label = "Normal"),
  actionButton(inputId = "unif", label = "Uniform"),
  plotOutput("hist")
)

server <- function(input, output) {

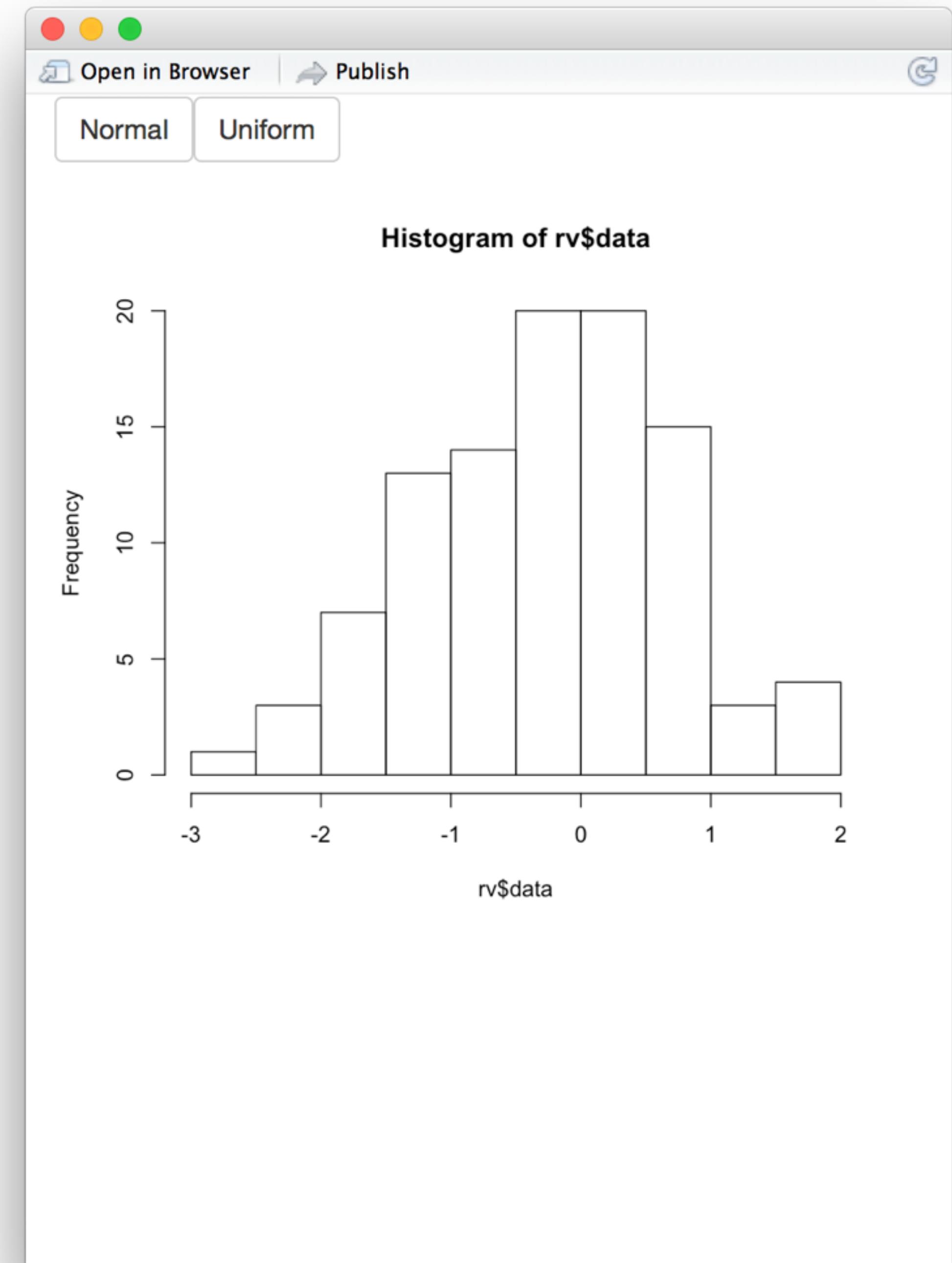
  rv <- reactiveValues(data = rnorm(100))

  observeEvent(input$norm, { rv$data <- rnorm(100) })
  observeEvent(input$unif, { rv$data <- runif(100) })

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

shinyApp(ui = ui, server = server)

```



```

# 08-reactiveValues

library(shiny)

ui <- fluidPage(
  actionButton(inputId = "norm", label = "Normal"),
  actionButton(inputId = "unif", label = "Uniform"),
  plotOutput("hist")
)

server <- function(input, output) {

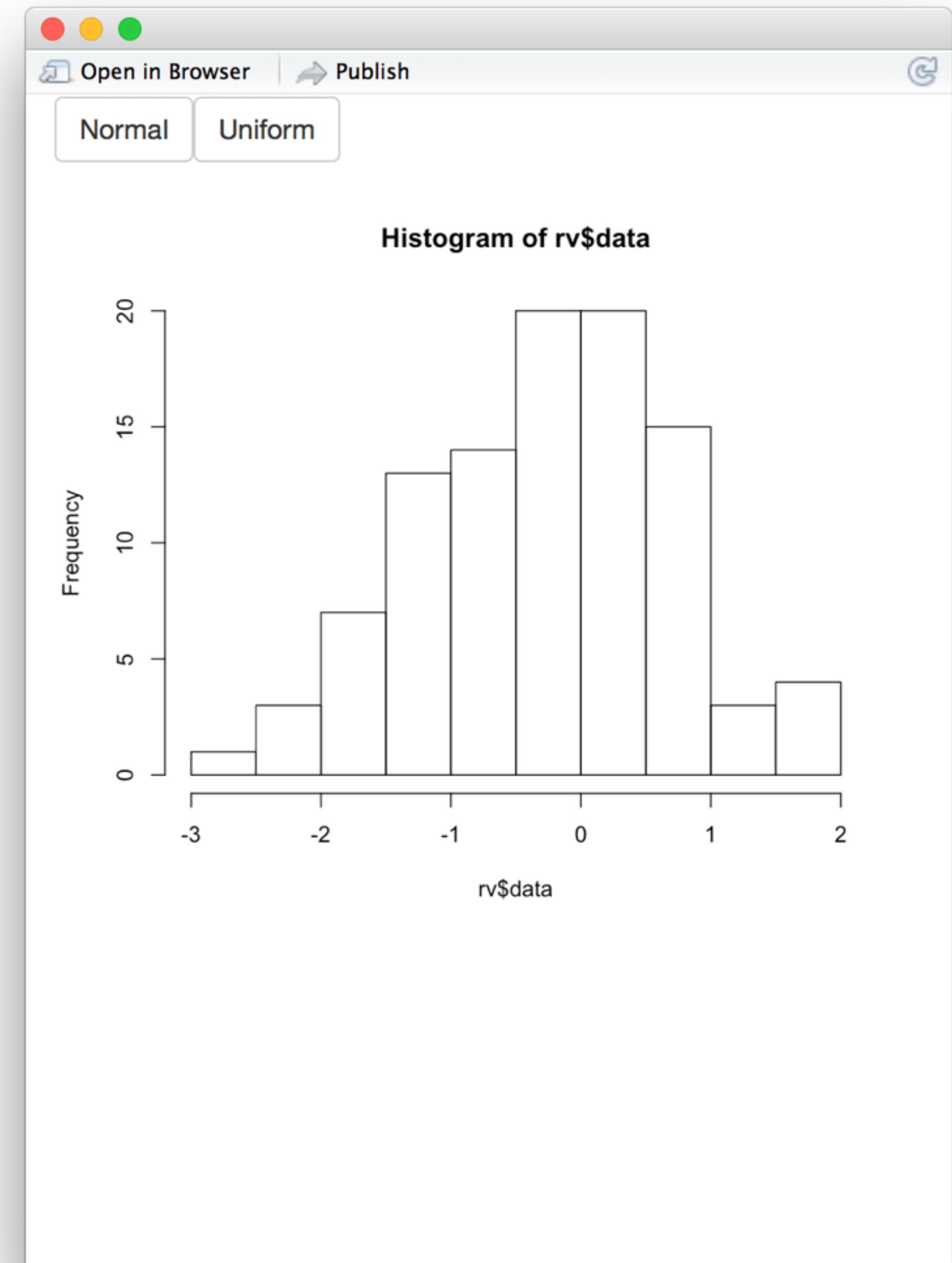
  rv <- reactiveValues(data = rnorm(100))

  observeEvent(input$norm, { rv$data <- rnorm(100) })
  observeEvent(input$unif, { rv$data <- runif(100) })

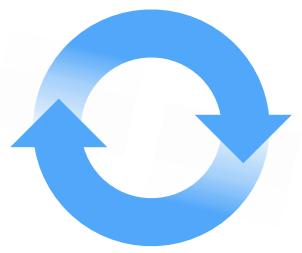
  output$hist <- renderPlot({
    hist(rv$data)
  })
}

shinyApp(ui = ui, server = server)

```



Recap: reactiveValues()

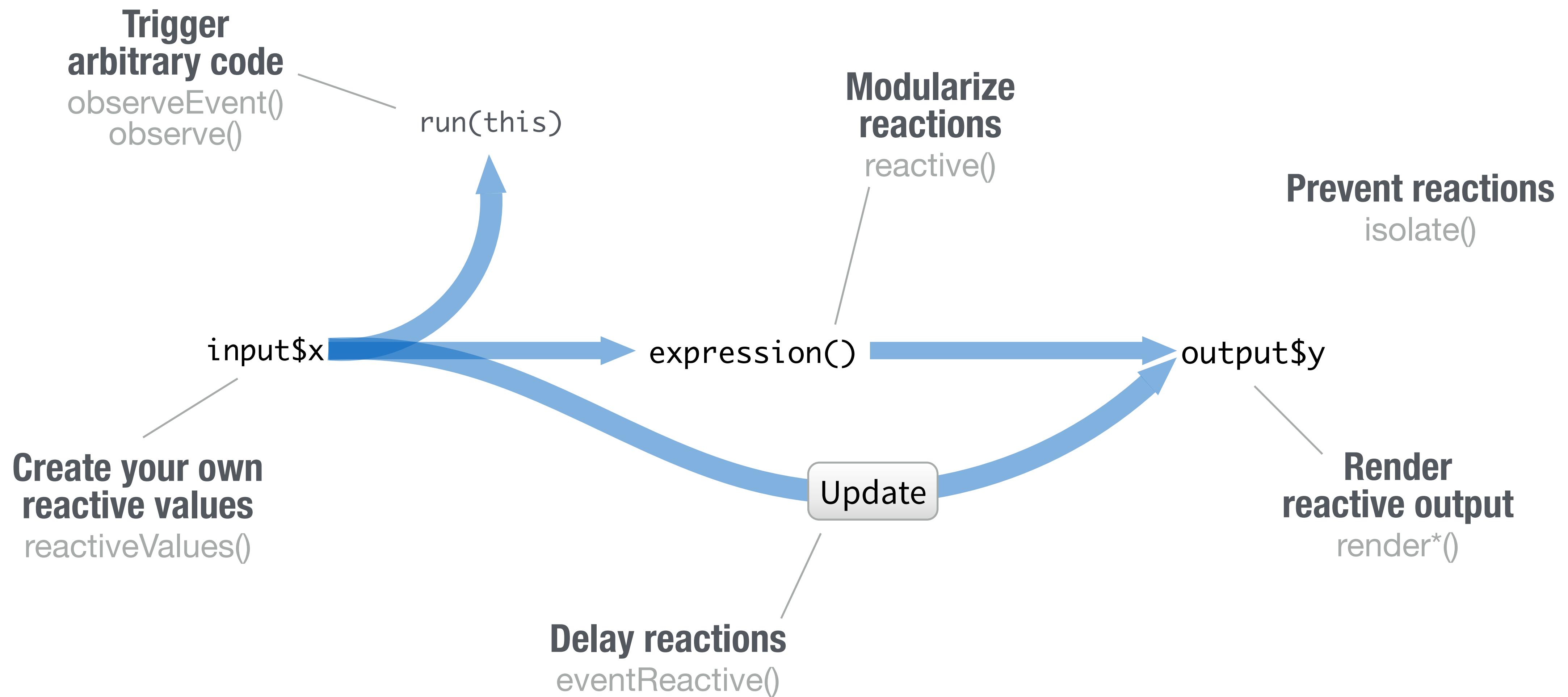


reactiveValues() creates a list of **reactive values**

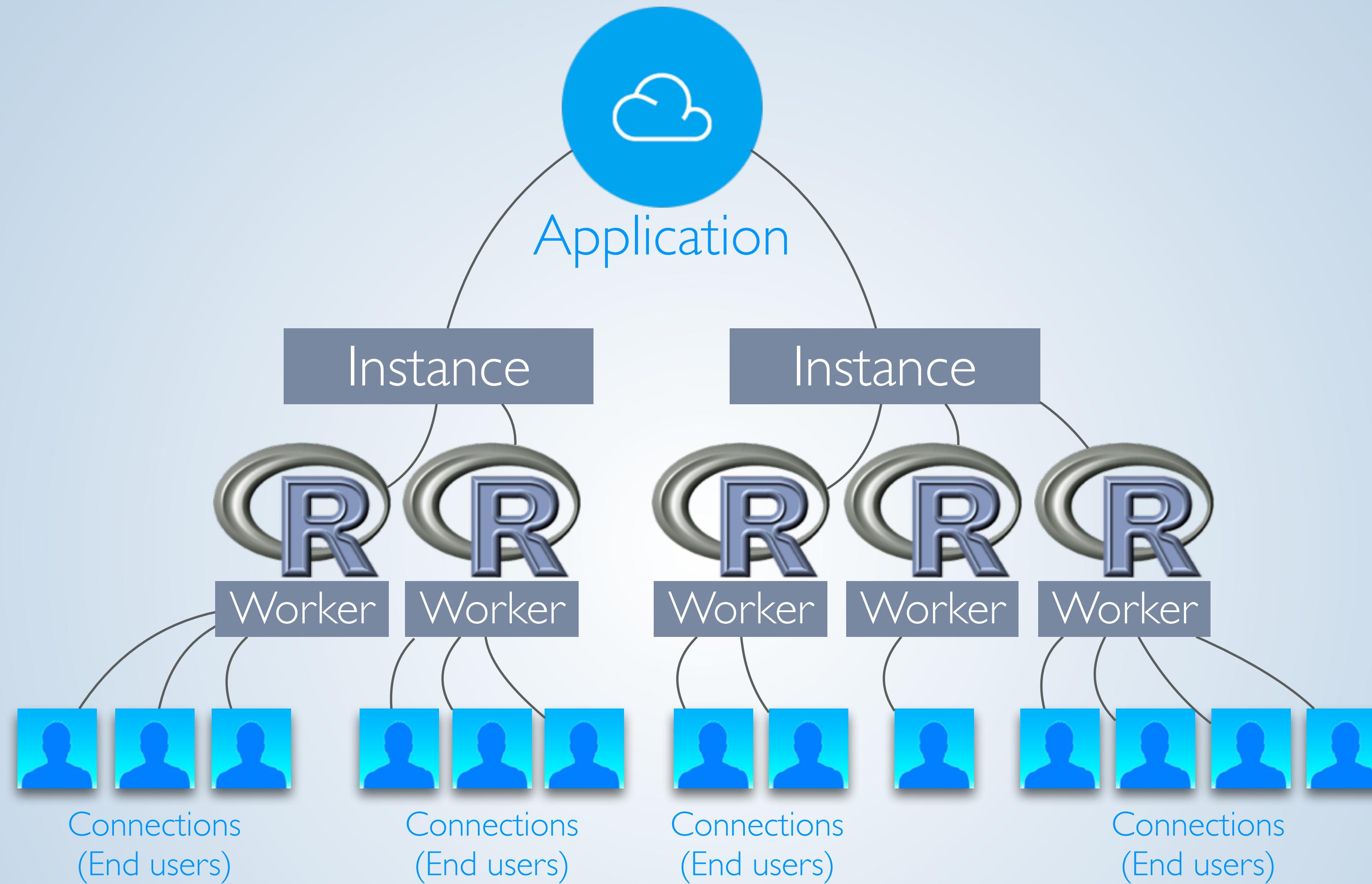
`rv$data <-`

You can manipulate these values (usually with observeEvent())

You now how to



Parting tips



Reduce repetition

Place code where it will be re-run as little as necessary

```
library(shiny)

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

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

shinyApp(ui = ui, server = server)
```

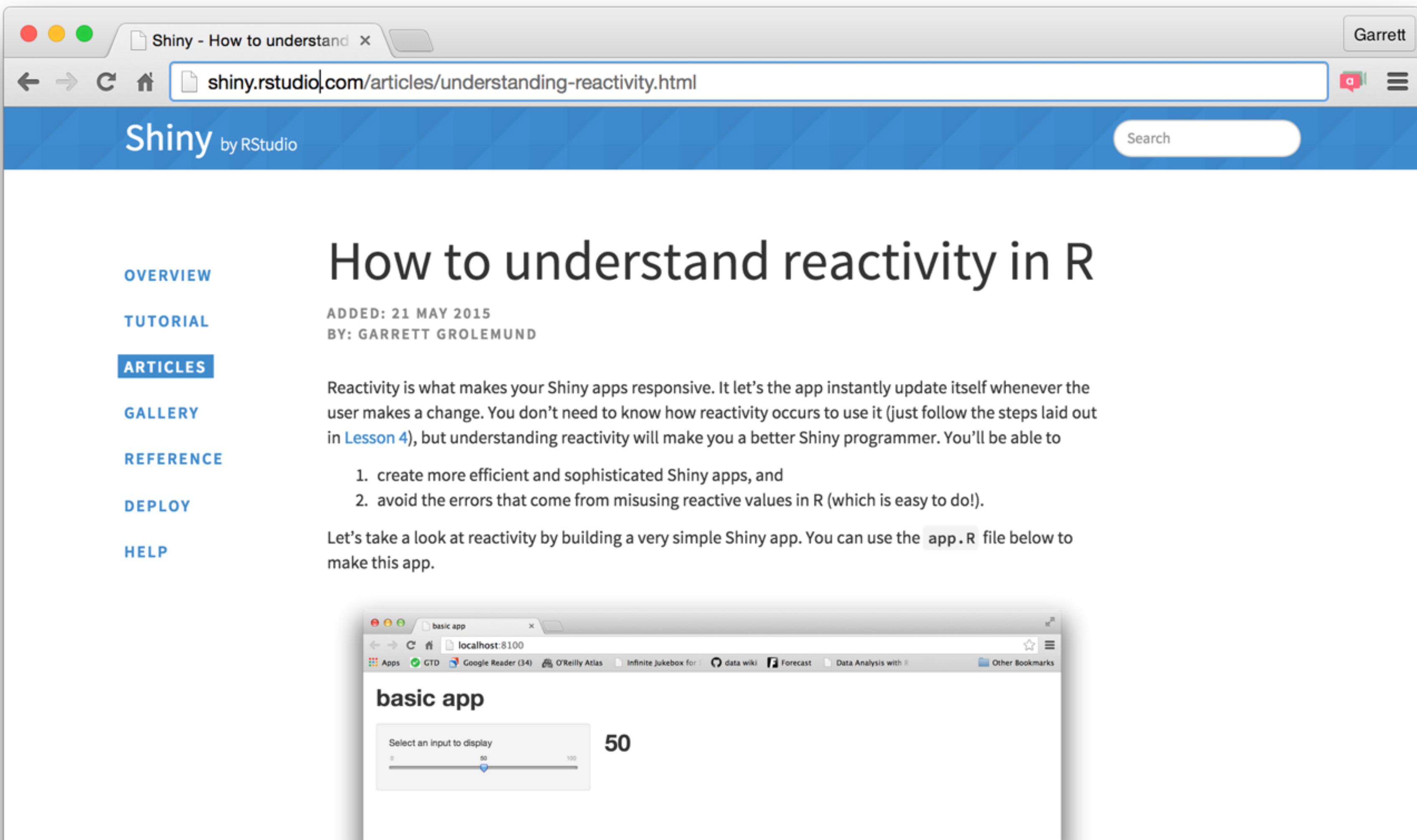
Code outside the server function will
be run once per R session (worker)

Code inside the server function will be
run once per end user (connection)

Code inside a reactive function will be
run once per reaction
(e.g. many times)

How can R possibly implement reactivity?

<http://shiny.rstudio.com/articles/understanding-reactivity.html>



The screenshot shows a web browser window with the title bar "Shiny - How to understand" and the address bar containing the URL "shiny.rstudio.com/articles/understanding-reactivity.html". The browser interface includes standard controls like back, forward, and search. The main content area features a blue header with the "Shiny by RStudio" logo and a search bar. The main heading is "How to understand reactivity in R". To the left is a sidebar with navigation links: OVERVIEW, TUTORIAL, ARTICLES (which is selected), GALLERY, REFERENCE, DEPLOY, and HELP. Below the sidebar, the article details are listed: "ADDED: 21 MAY 2015" and "BY: GARRETT GROLEMUND". The article text explains what reactivity is and how it makes Shiny apps responsive. It also lists two goals for understanding reactivity: creating more efficient apps and avoiding errors from misusing reactive values. At the bottom, there's a note about building a simple Shiny app using an `app.R` file. A smaller inset screenshot at the bottom shows a "basic app" window with a slider control set to 50.

Garrett

shiny.rstudio.com/articles/understanding-reactivity.html

Shiny by RStudio

Search

OVERVIEW

TUTORIAL

ARTICLES

GALLERY

REFERENCE

DEPLOY

HELP

How to understand reactivity in R

ADDED: 21 MAY 2015
BY: GARRETT GROLEMUND

Reactivity is what makes your Shiny apps responsive. It lets the app instantly update itself whenever the user makes a change. You don't need to know how reactivity occurs to use it (just follow the steps laid out in [Lesson 4](#)), but understanding reactivity will make you a better Shiny programmer. You'll be able to

1. create more efficient and sophisticated Shiny apps, and
2. avoid the errors that come from misusing reactive values in R (which is easy to do!).

Let's take a look at reactivity by building a very simple Shiny app. You can use the `app.R` file below to make this app.

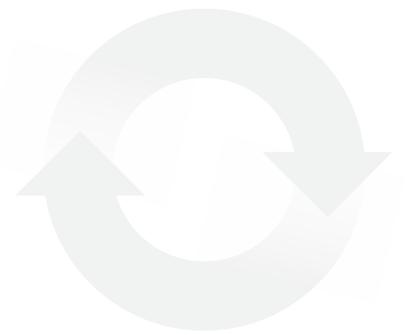
basic app

Select an input to display

50

Learn
more

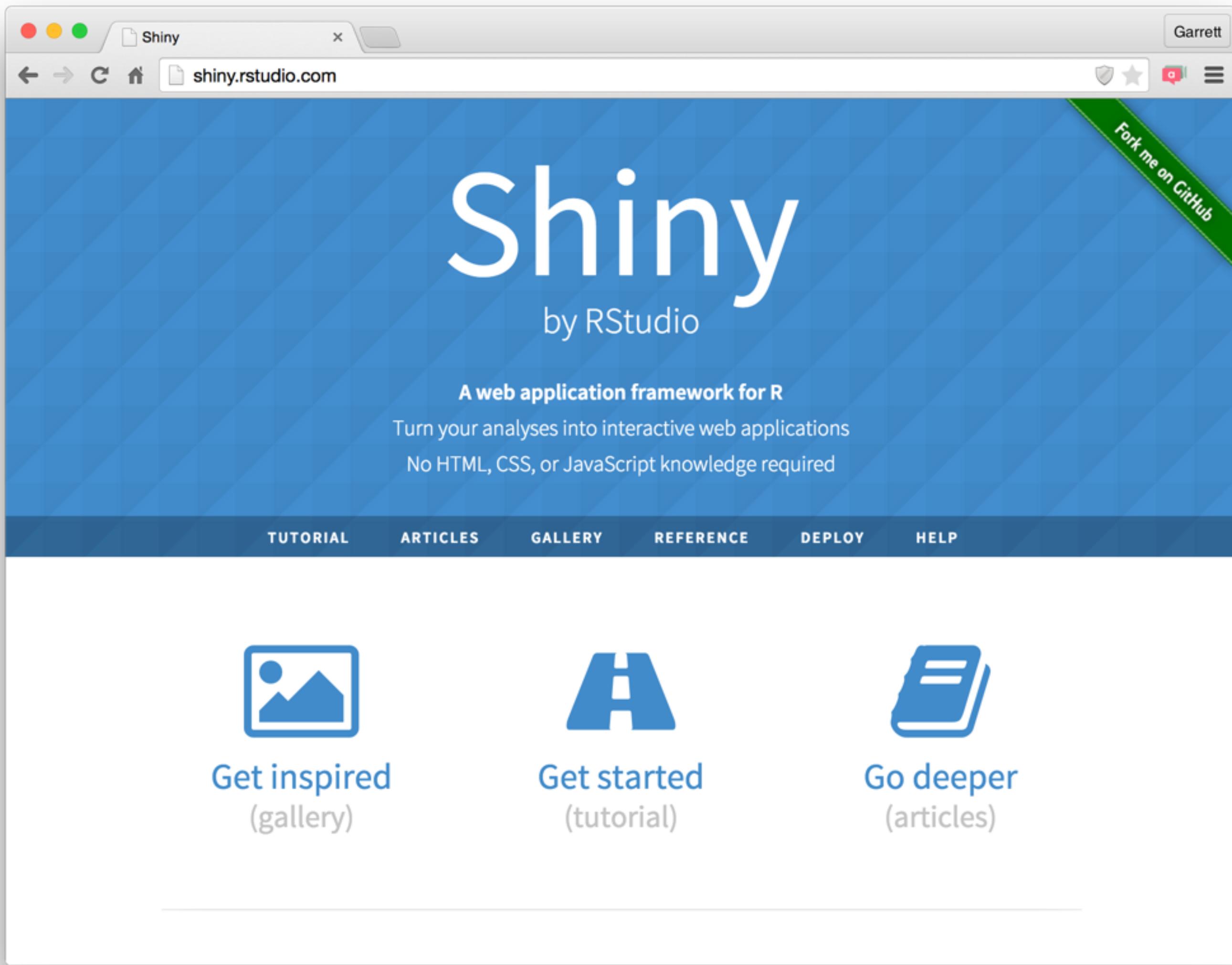
How to start with Shiny



1. How to build a Shiny app (www.rstudio.com/resources/webinars/)
2. How to customize reactions (Today)
3. How to customize appearance (June 17)

The Shiny Development Center

shiny.rstudio.com



All Training materials are provided "as is" and without warranty and RStudio disclaims any and all express and implied warranties including without limitation the implied warranties of title, fitness for a particular purpose, merchantability and noninfringement.

The Training Materials are licensed under the Creative Commons Attribution-Noncommercial 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

How to start with Shiny, Part 3

How to customize appearance



Garrett Grolemund

Data Scientist and Master Instructor

May 2015

Email: garrett@rstudio.com

Twitter: @StatGarrett

Code and slides at:

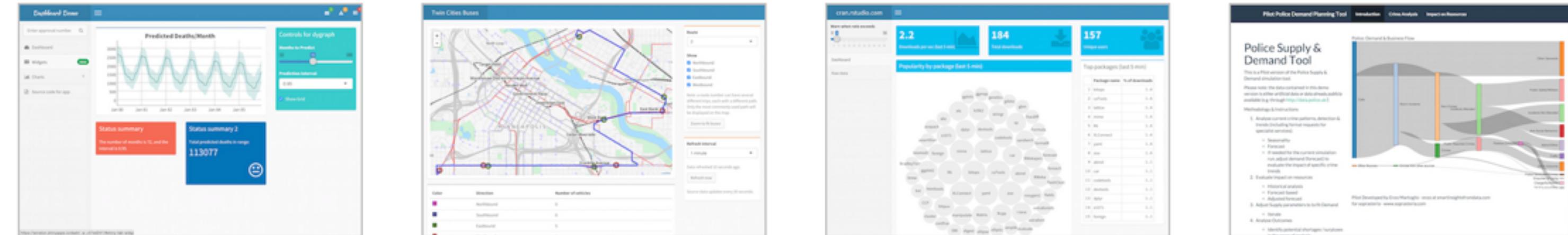
bit.ly/shiny-quickstart-3



Shiny Showcase

www.rstudio.com/products/shiny/shiny-user-showcase/

Shiny Apps for the Enterprise



[Shiny Dashboard Demo](#)

A dashboard built with Shiny.

[Location tracker](#)

Track locations over time with streaming data.

[Download monitor](#)

Streaming download rates visualized as a bubble chart.

[Supply and Demand](#)

Forecast demand to plan resource allocation.

Industry Specific Shiny Apps



[Economic Dashboard](#)

Economic forecasting with macroeconomic indicators.

[ER Optimization](#)

An app that models patient flow.

[CDC Disease Monitor](#)

Alert thresholds and automatic weekly updates.

[Ebola Model](#)

An epidemiological simulation.



[Pharmacometrics: some Shiny applications](#)

Introduction to pharmacokinetics modelling

Modeling the absorption process

Modeling the elimination process

Building a pharmacokinetics model

[USGS](#)

Lake Erie Biological Station - Western Basin Trawl Survey

Lake Erie Western Basin Map

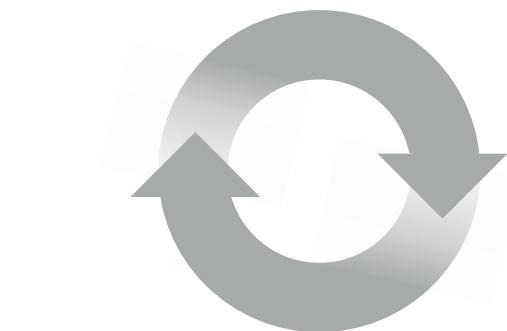
Give user point to display station number and station value for each plot

© CC 2015 RStudio, Inc.

How to start with Shiny

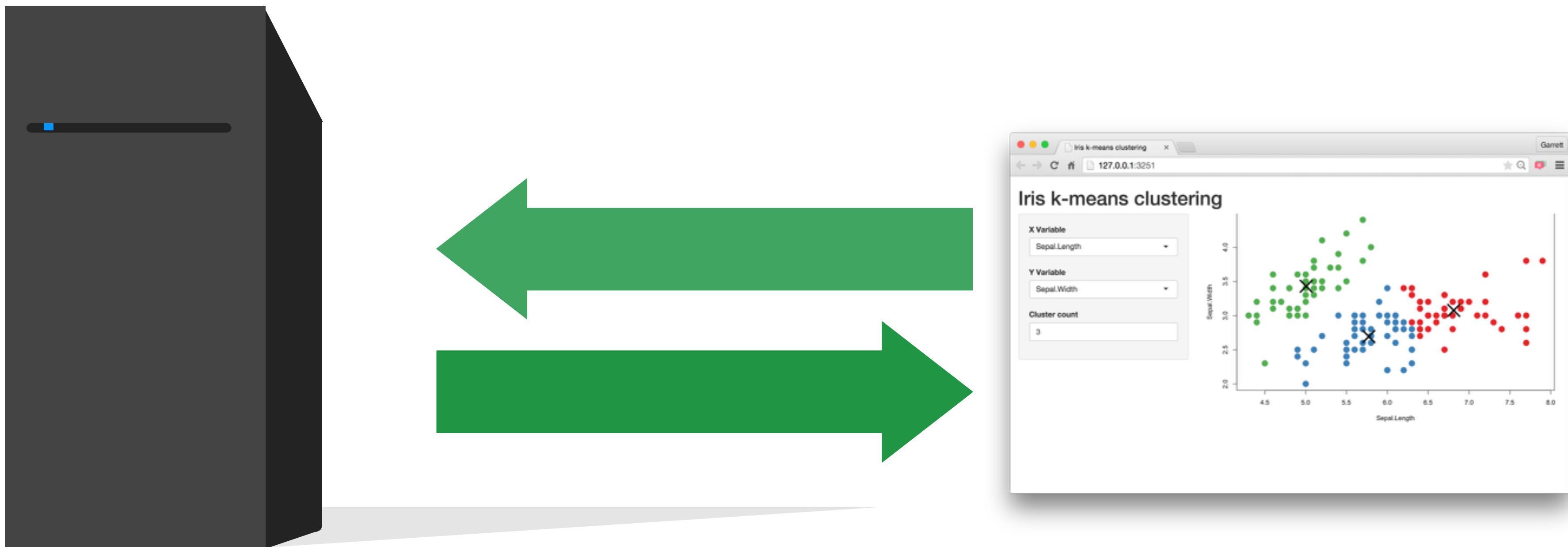


1. How to build a Shiny app (www.rstudio.com/resources/webinars/)
2. How to customize reactions (www.rstudio.com/resources/webinars/)
3. How to customize appearance (Today)



**The story
so far**

Every Shiny app is maintained by a computer running R



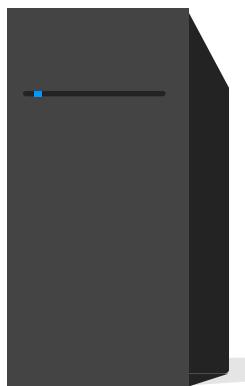
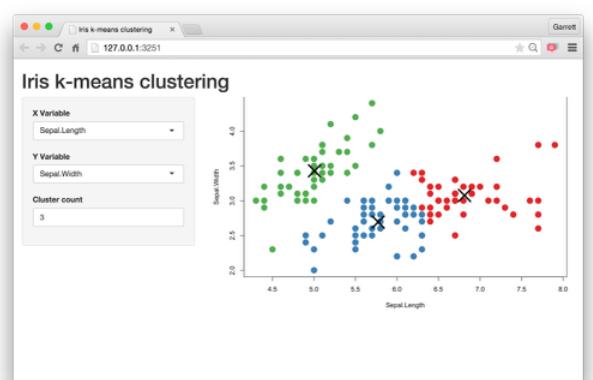
Sharing apps



Shiny Server (Pro)

App template

The shortest viable shiny app



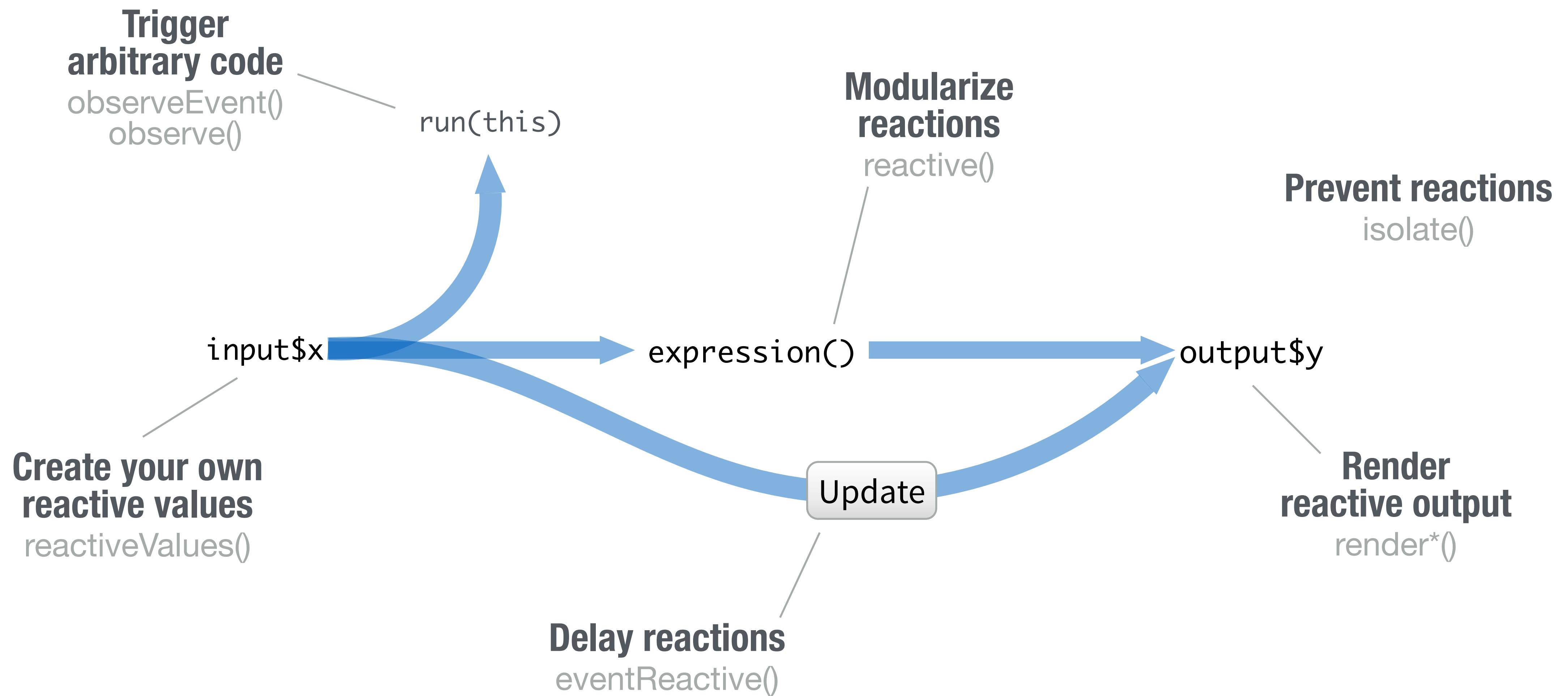
```
library(shiny)
```

```
ui <- fluidPage()
```

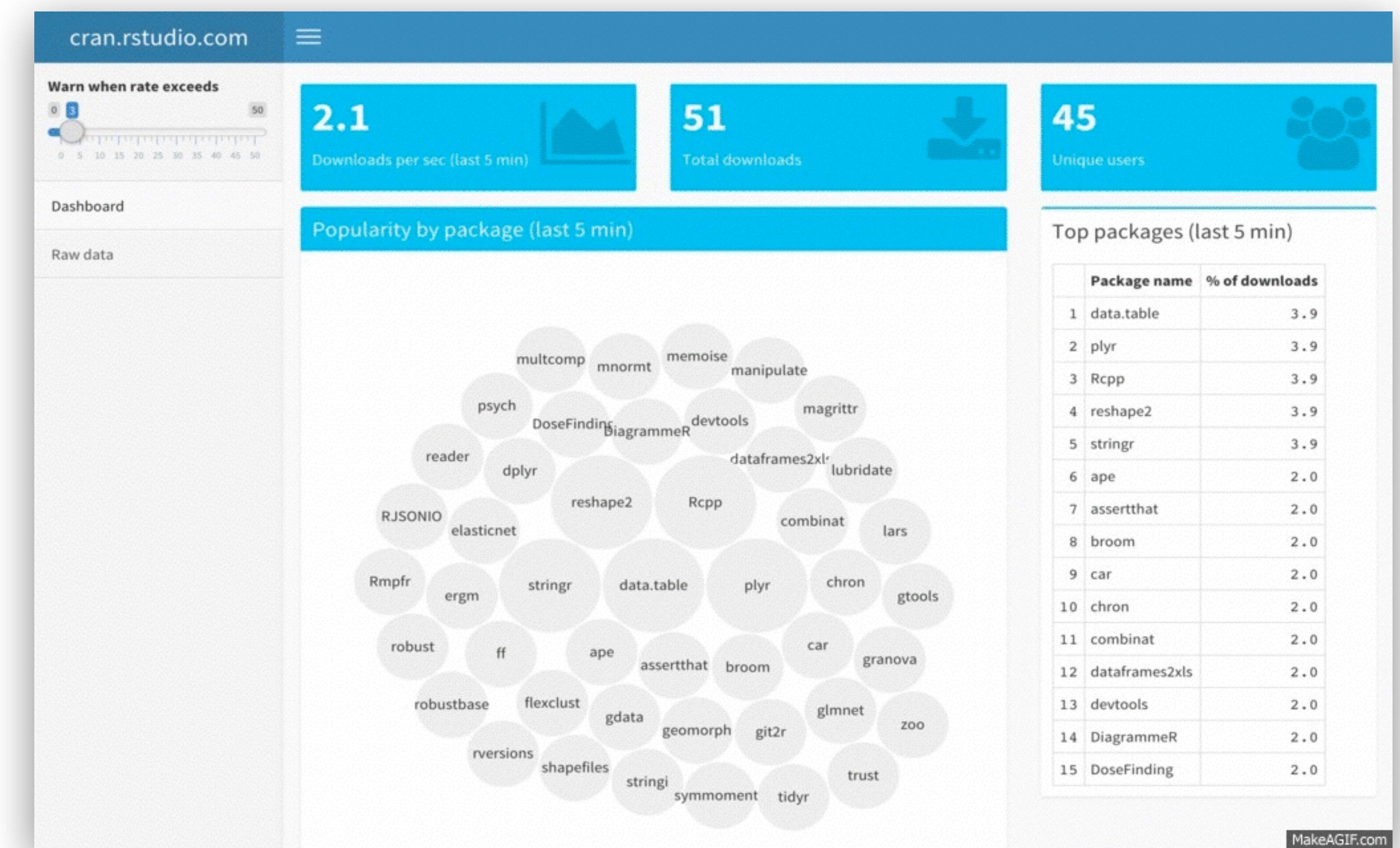
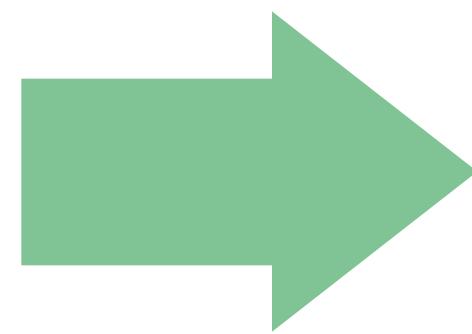
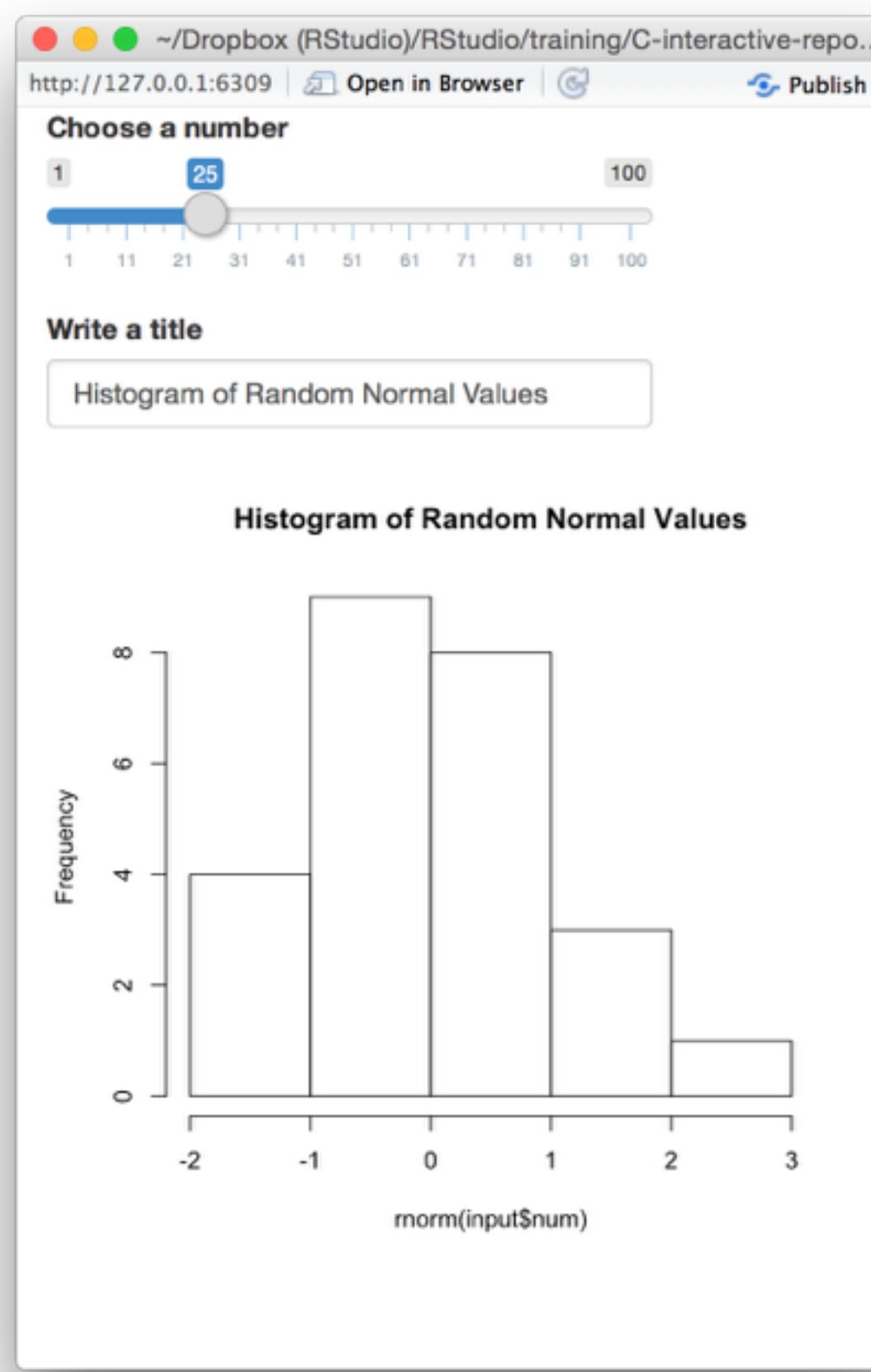
```
server <- function(input, output) {}
```

```
shinyApp(ui = ui, server = server)
```

Reactivity



User Interface



Work with the
ITMI UI

The HTML that builds the user interface for your app

```
ui <- fluidPage()
```

```
fluidPage()
```

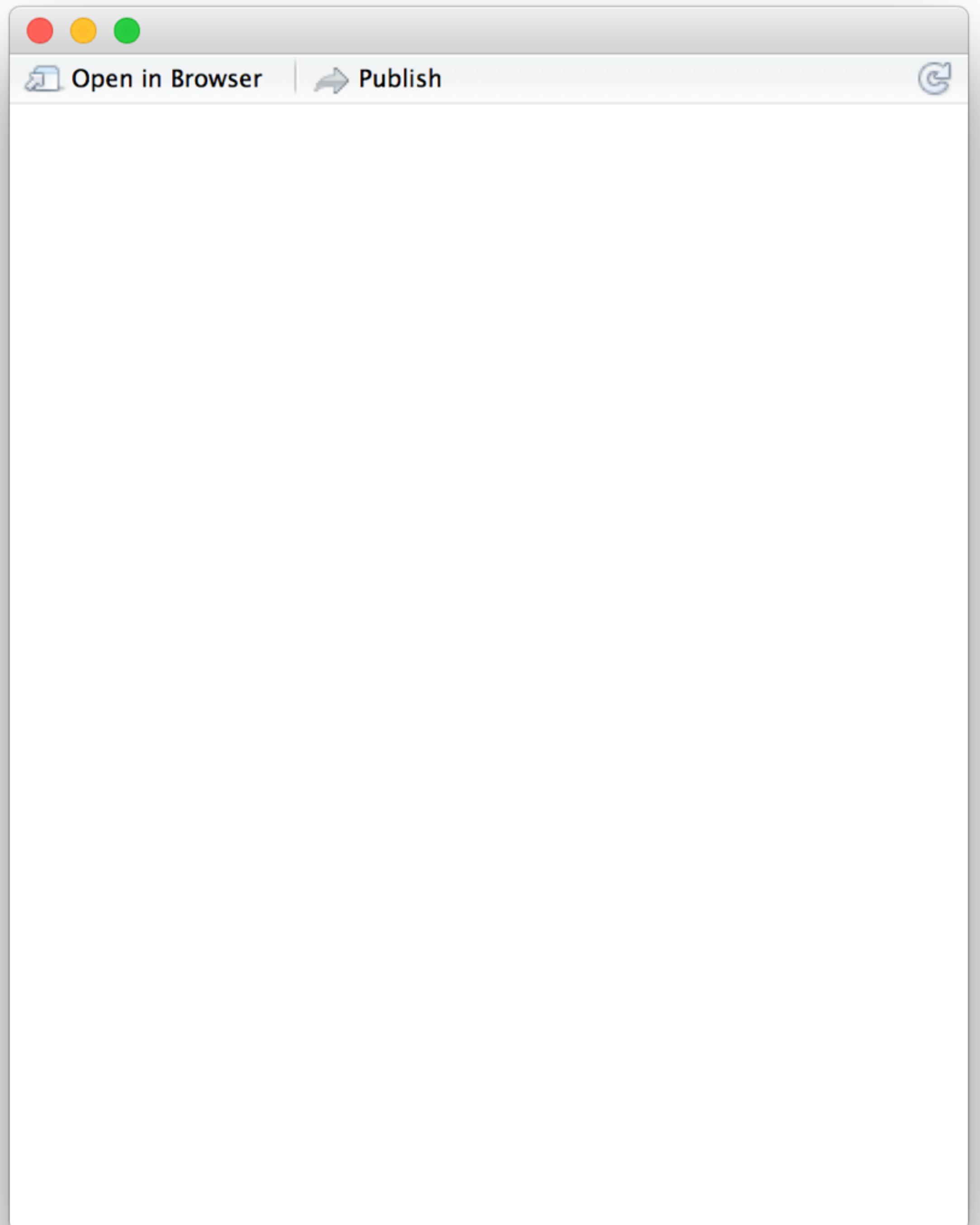
```
<div class="container-fluid"></div>
```

```
library(shiny)

ui <- fluidPage(
  )

server <- function(input, output) {
  }

shinyApp(ui = ui, server = server)
```



```
library(shiny)

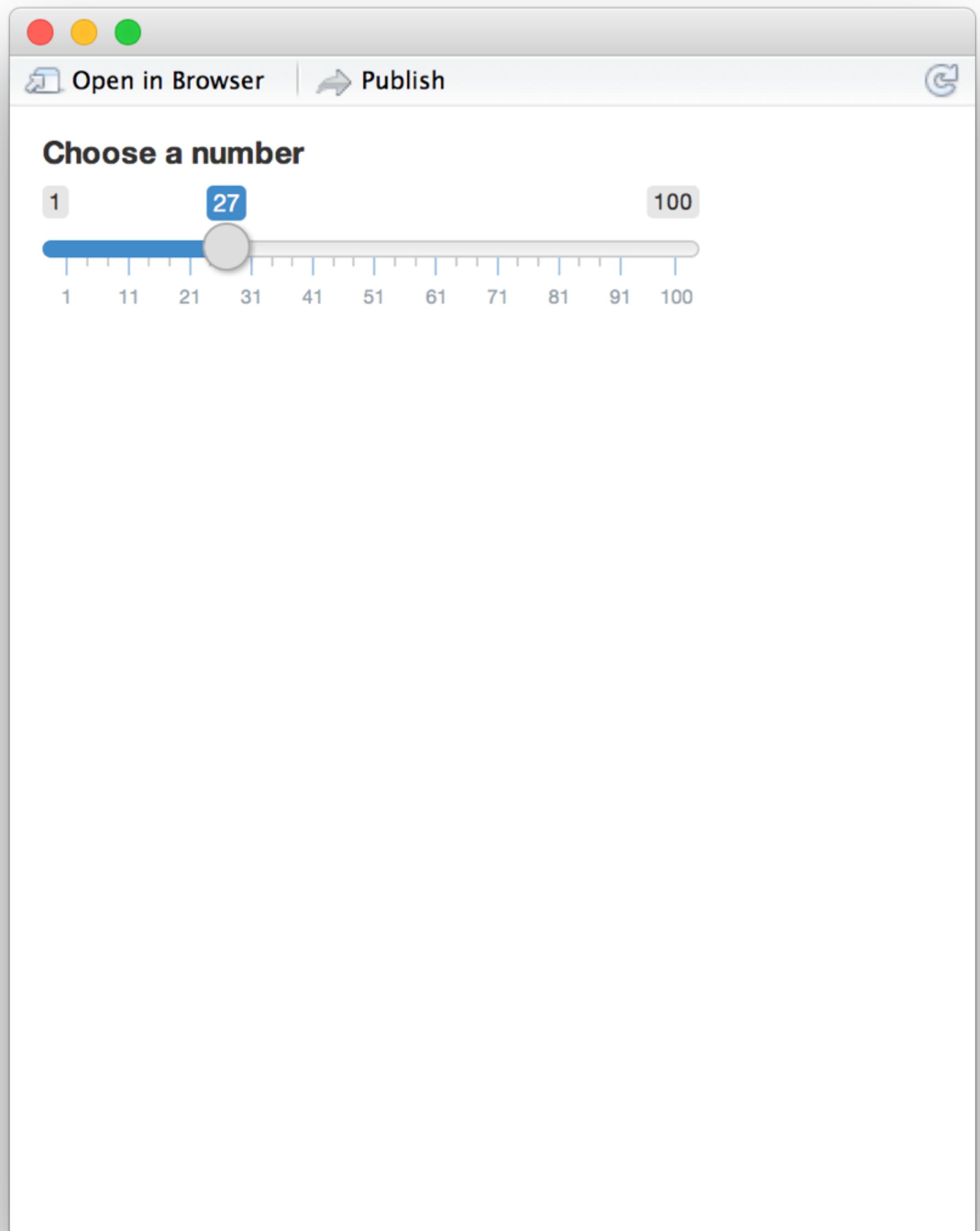
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100)

)

server <- function(input, output) {

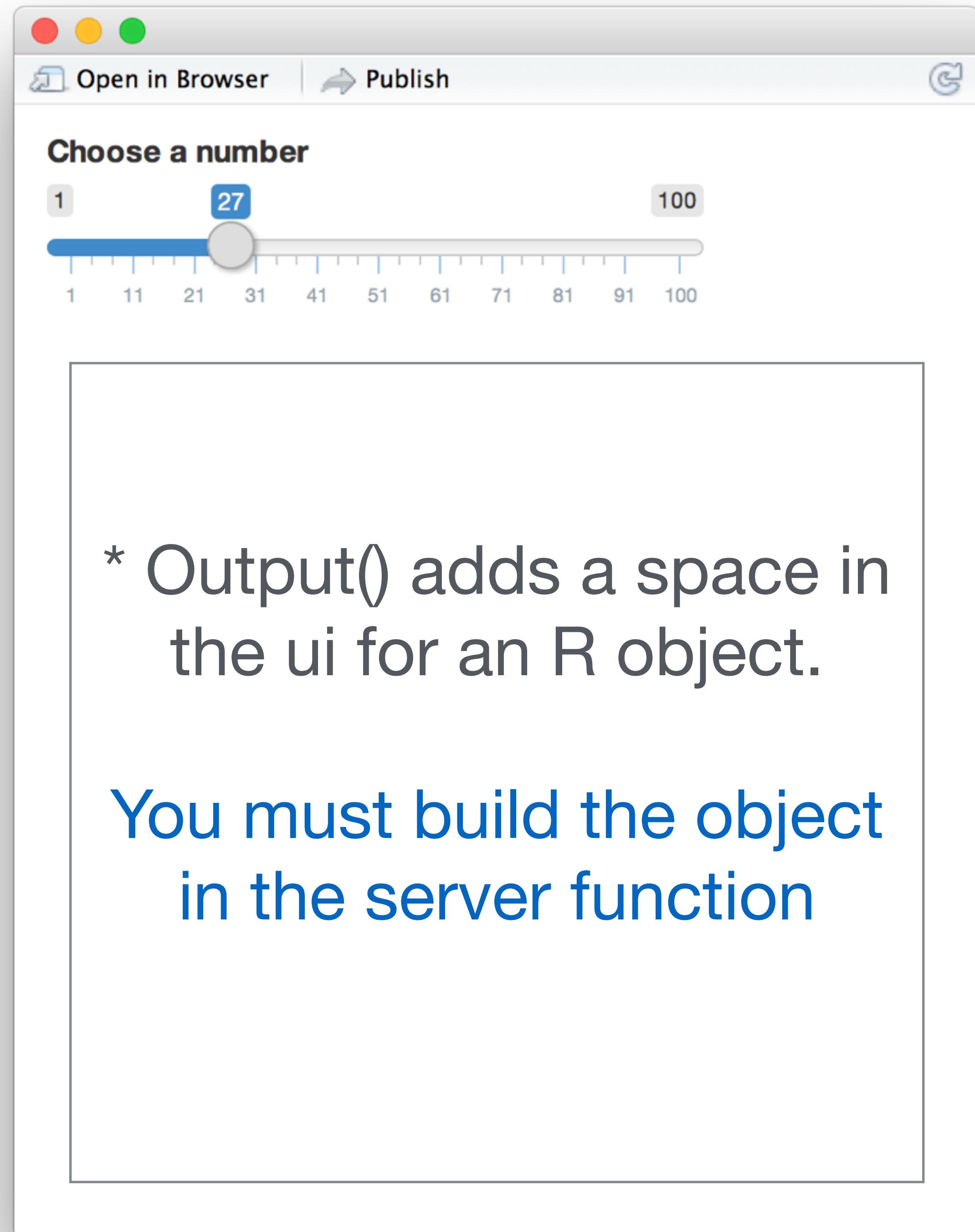
}

shinyApp(ui = ui, server = server)
```



```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
server <- function(input, output) {
}
shinyApp(ui = ui, server = server)
```



```
sliderInput(inputId = "num",
  label = "Choose a number",
  value = 25, min = 1, max = 100)
```

```
<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-prettyify-separator="," data-keyboard="true"
    data-keyboard-step="1.010101010101"/>
</div>
```

```
plotOutput("hist")
```

```
<div id="hist" class="shiny-plot-output" style="width: 100% ; height: 400px"></div>
```

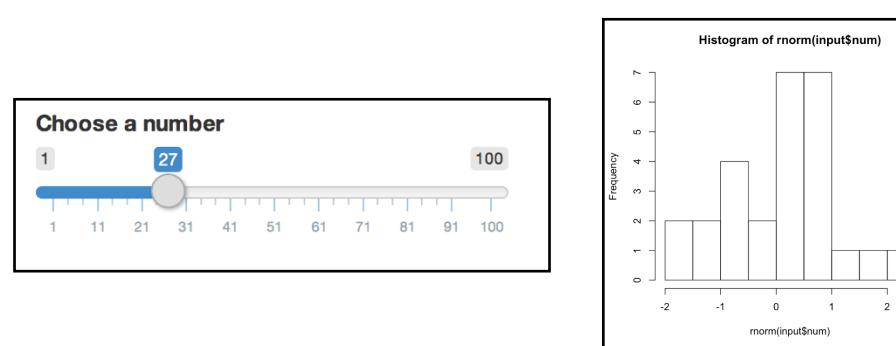
Recap: User Interface



Call R functions to assemble HTML

`fluidPage()`

Use **fluidPage()** to quickly set up a UI



Use **input***() and **output***() functions to add reactive content.

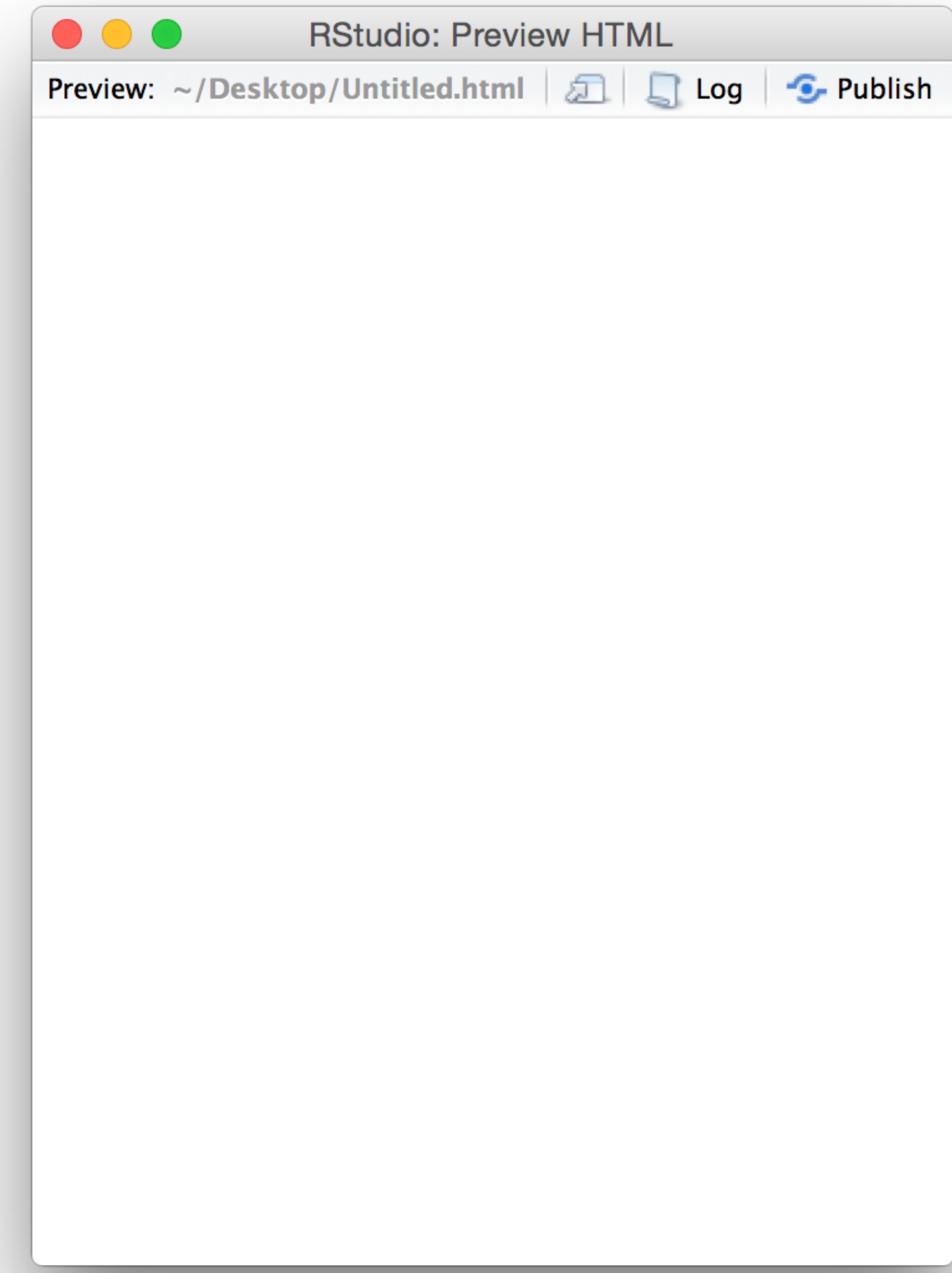
`ui <-`

Save the output to **ui**

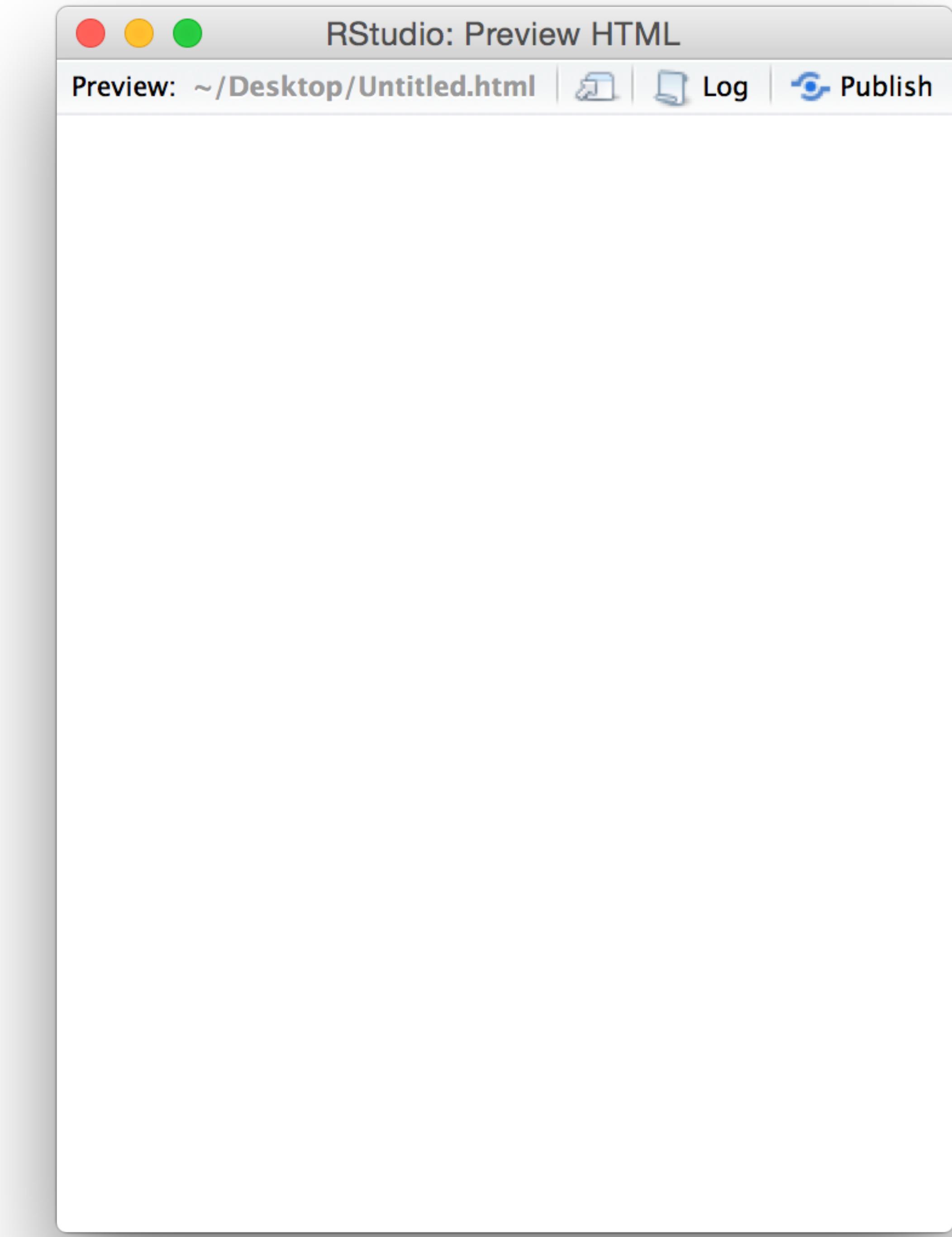
Add
static content

How do you add content to a web page?

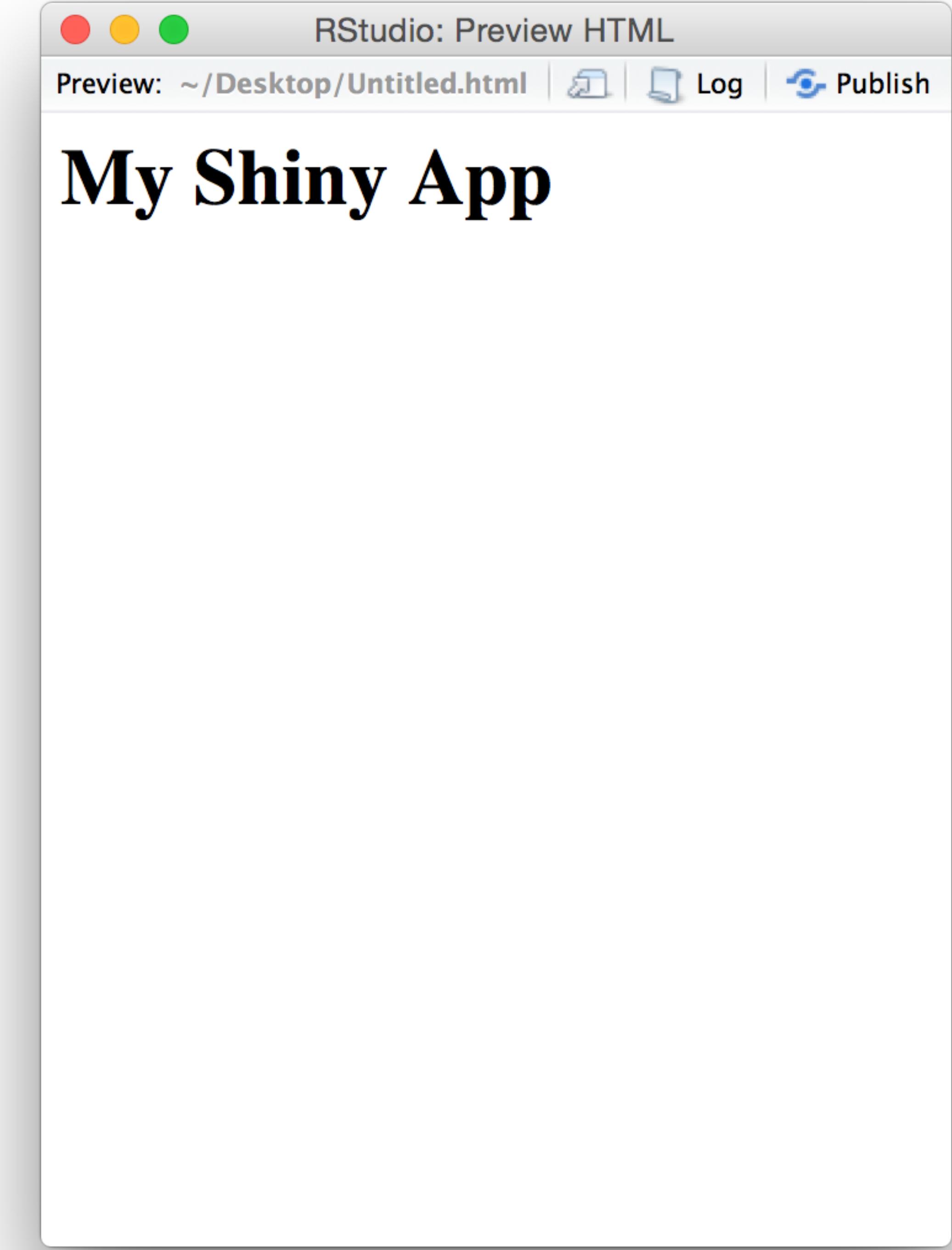
```
<div class="container-fluid"></div>
```



```
<div class="container-fluid">  
  
</div>
```



```
<div class="container-fluid">  
  <h1>My Shiny App</h1>  
</div>
```



```
<div class="container-fluid">  
  <h1>My Shiny App</h1>  
  <p>See other apps in the</p>  
</div>
```

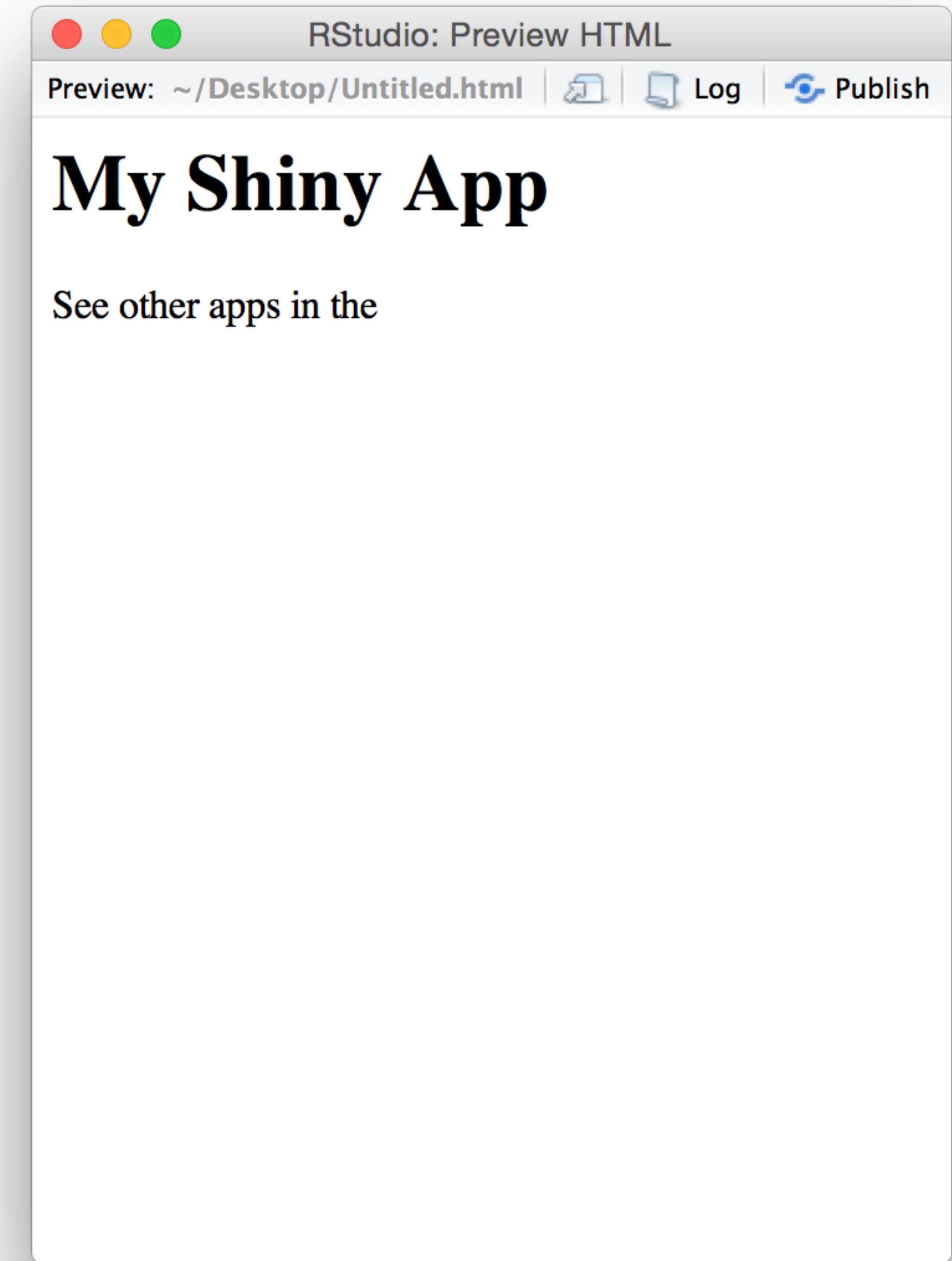
RStudio: Preview HTML

Preview: ~/Desktop/Untitled.html | Log | Publish

My Shiny App

See other apps in the

```
<div class="container-fluid">  
  <h1>My Shiny App</h1>  
  <p>See other apps in the  
    </p>  
</div>
```



```
<div class="container-fluid">  
  <h1>My Shiny App</h1>  
  <p>See other apps in the  
    <a href="http://www.rstudio.com/  
      products/shiny/shiny-user-  
      showcase/">Shiny Showcase</a>  
  </p>  
</div>
```

RStudio: Preview HTML

Preview: ~/Desktop/Untitled.html | Log | Publish

My Shiny App

See other apps in the [Shiny Showcase](#)

```
<div class="container-fluid">  
  <h1>My Shiny App</h1>  
  <p>See other apps in the  
    <a href="http://www.rstudio.com/  
      products/shiny/shiny-user-  
      showcase/">Shiny Showcase</a>  
  </p>  
</div>
```

RStudio: Preview HTML

Preview: ~/Desktop/Untitled.html | Log | Publish

My Shiny App

See other apps in the [Shiny Showcase](#)

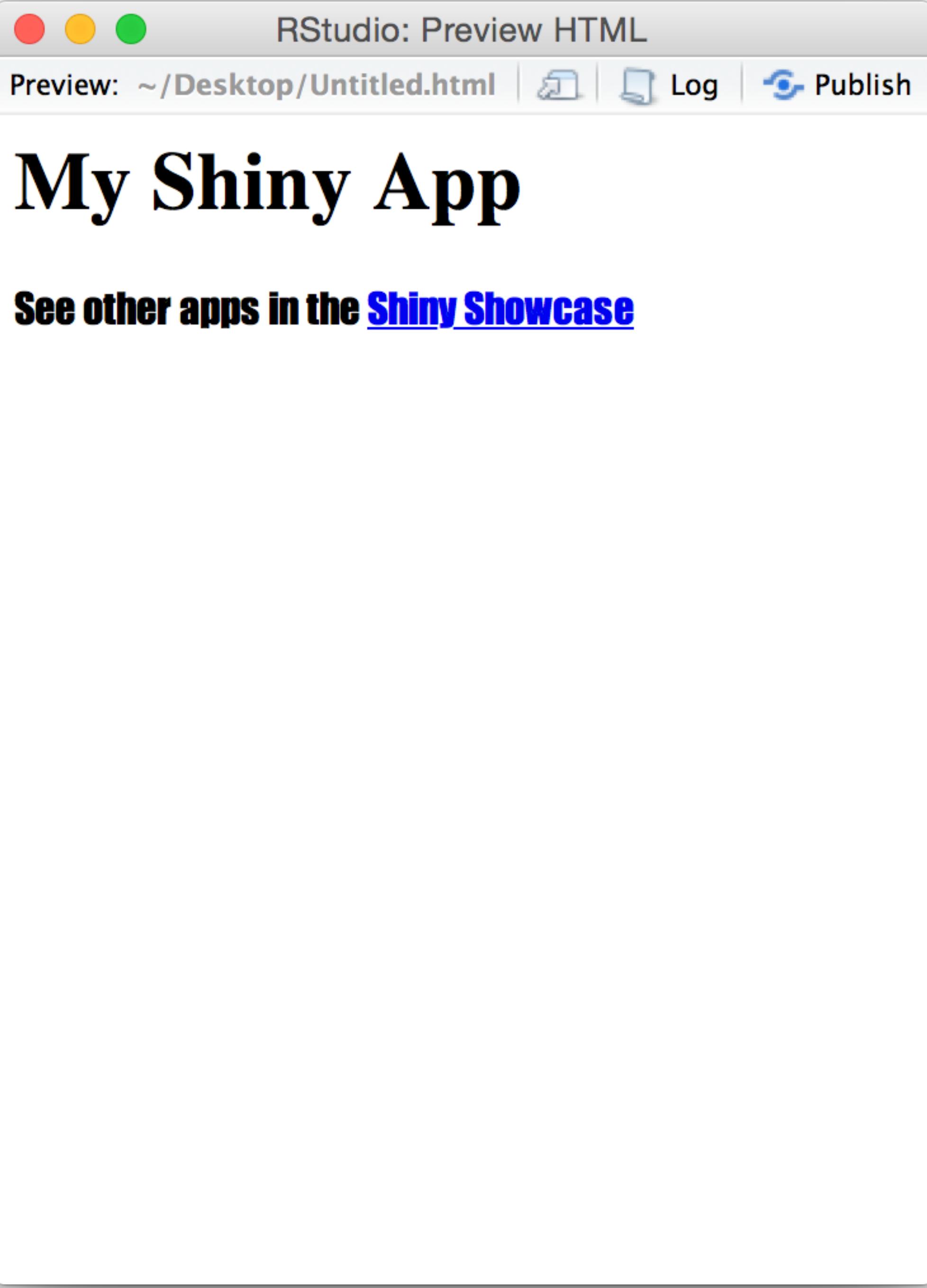
```
<div class="container-fluid">  
  <h1>My Shiny App</h1>  
  <p style="font-family:Impact">  
    See other apps in the  
    <a href="http://www.rstudio.com/  
      products/shiny/shiny-user-  
      showcase/">Shiny Showcase</a>  
  </p>  
</div>
```

RStudio: Preview HTML

Preview: ~/Desktop/Untitled.html | Log | Publish

My Shiny App

See other apps in the [Shiny Showcase](#)

A screenshot of the RStudio Preview HTML interface. The title bar says "RStudio: Preview HTML". Below it, "Preview: ~/Desktop/Untitled.html" is shown along with icons for Log and Publish. The main content area displays the rendered HTML. It features a large, bold, dark font header "My Shiny App". Below it is a paragraph with bold text "See other apps in the" followed by a blue, underlined link "Shiny Showcase". The rest of the page is blank.

How do you add content to a web page?

When writing HTML, you add content with **tags**.

`<h1></h1>`

`<a>`

How do you add content to a web page?

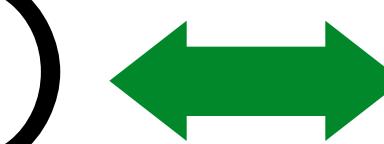
When writing R, you add content with **tags** functions.

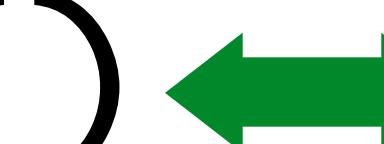
`tags$h1()` `<h1></h1>`

`tags$a()` `<a>`

Shiny HTML tag functions

Shiny provides R functions to recreate HTML tags.

`tags$h1()`  `<h1></h1>`

`tags$a()`  `<a>`

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"
```

A list of functions

```
tags$h1
```

```
function (...)  
tag("h1", list(...))  
<environment: namespace:htmltools>
```

```
tags$h1()
```

```
<h1></h1>
```

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)

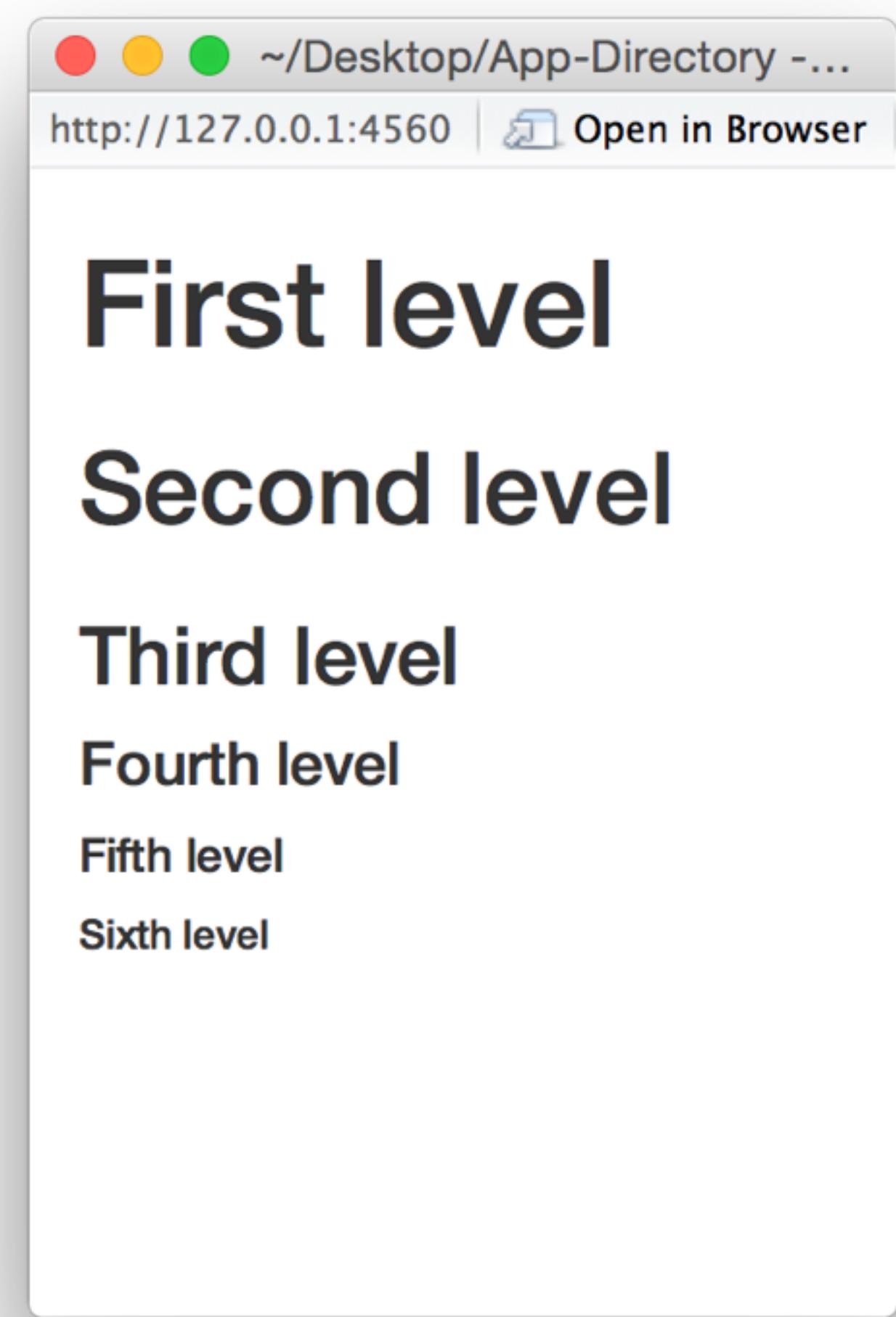
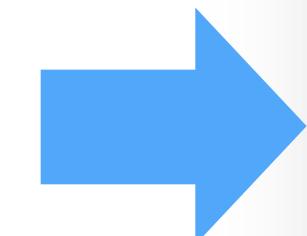
unnamed arguments
appear inside the tags
(call tags\$...() to create nested tags)

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

h1() - h6()

Headers

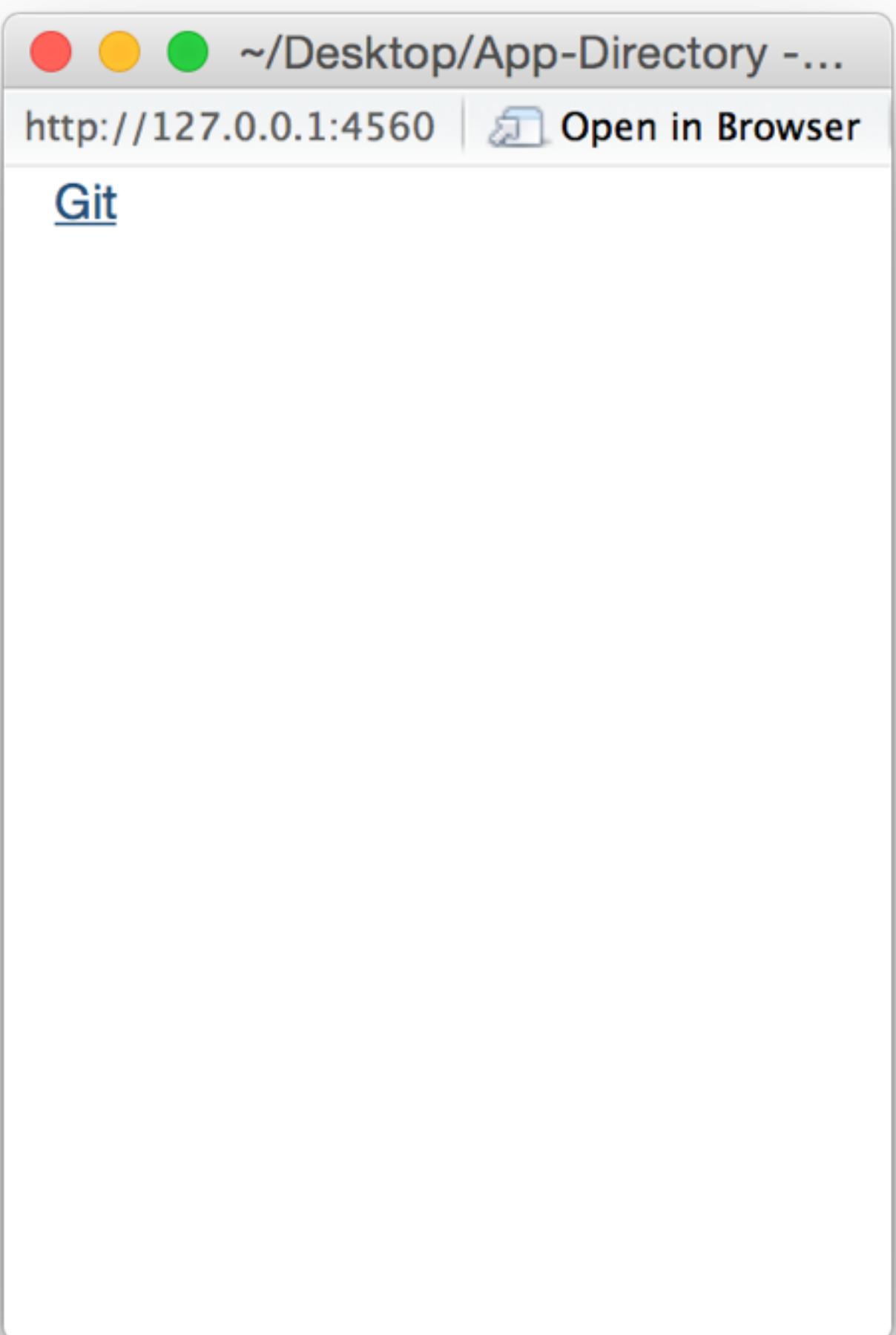
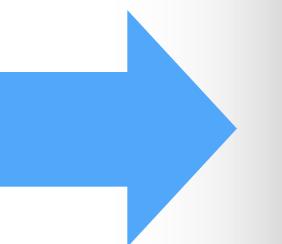
```
fluidPage(  
  tags$h1("First level"),  
  tags$h2("Second level"),  
  tags$h3("Third level"),  
  tags$h4("Fourth level"),  
  tags$h5("Fifth level"),  
  tags$h6("Sixth level"))
```



a()

hyperlink with the href argument

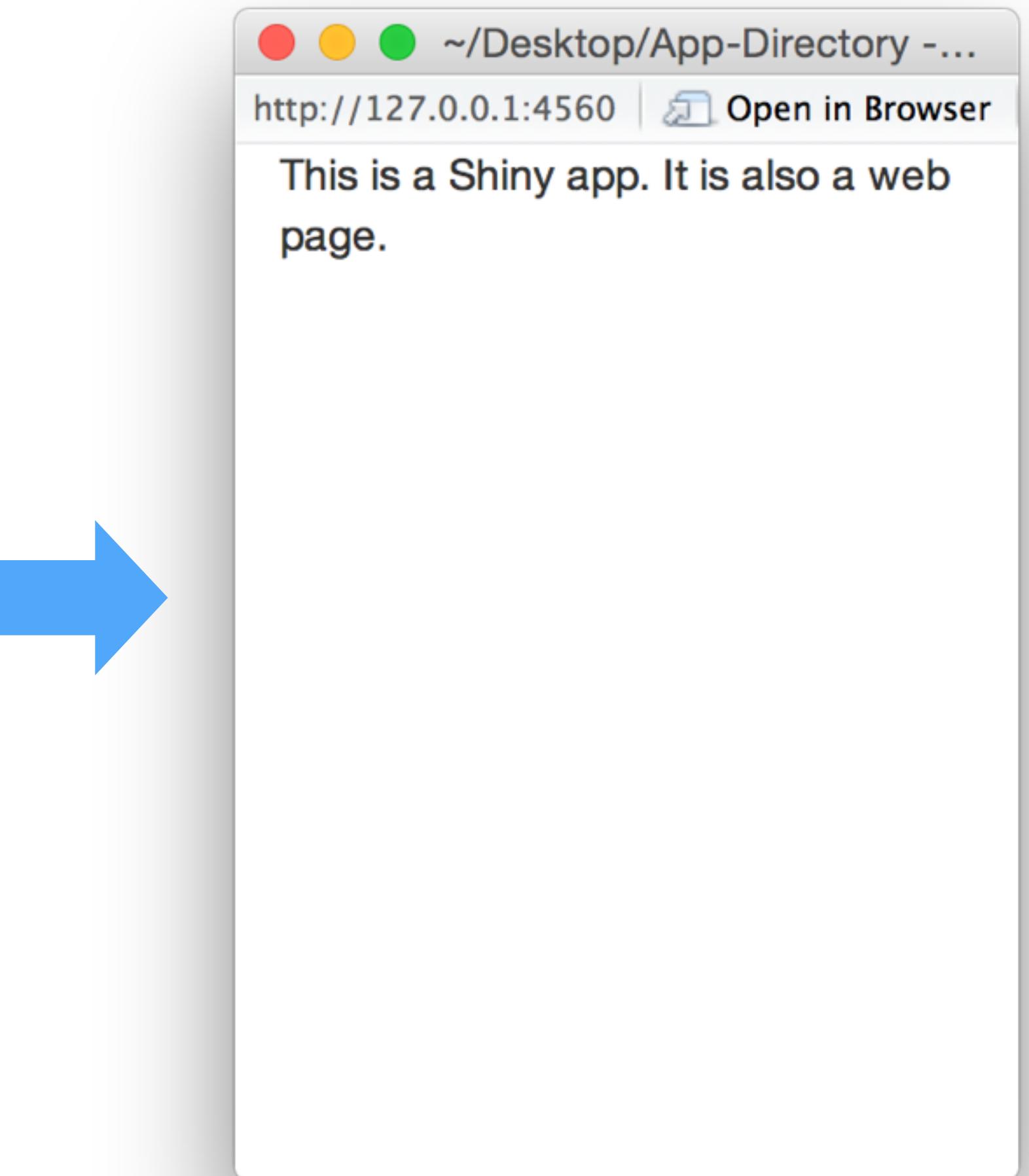
```
fluidPage(  
  tags$a(href= "http://www.git.com",  
         "Git")  
)
```



text

Character strings do not need a tag.

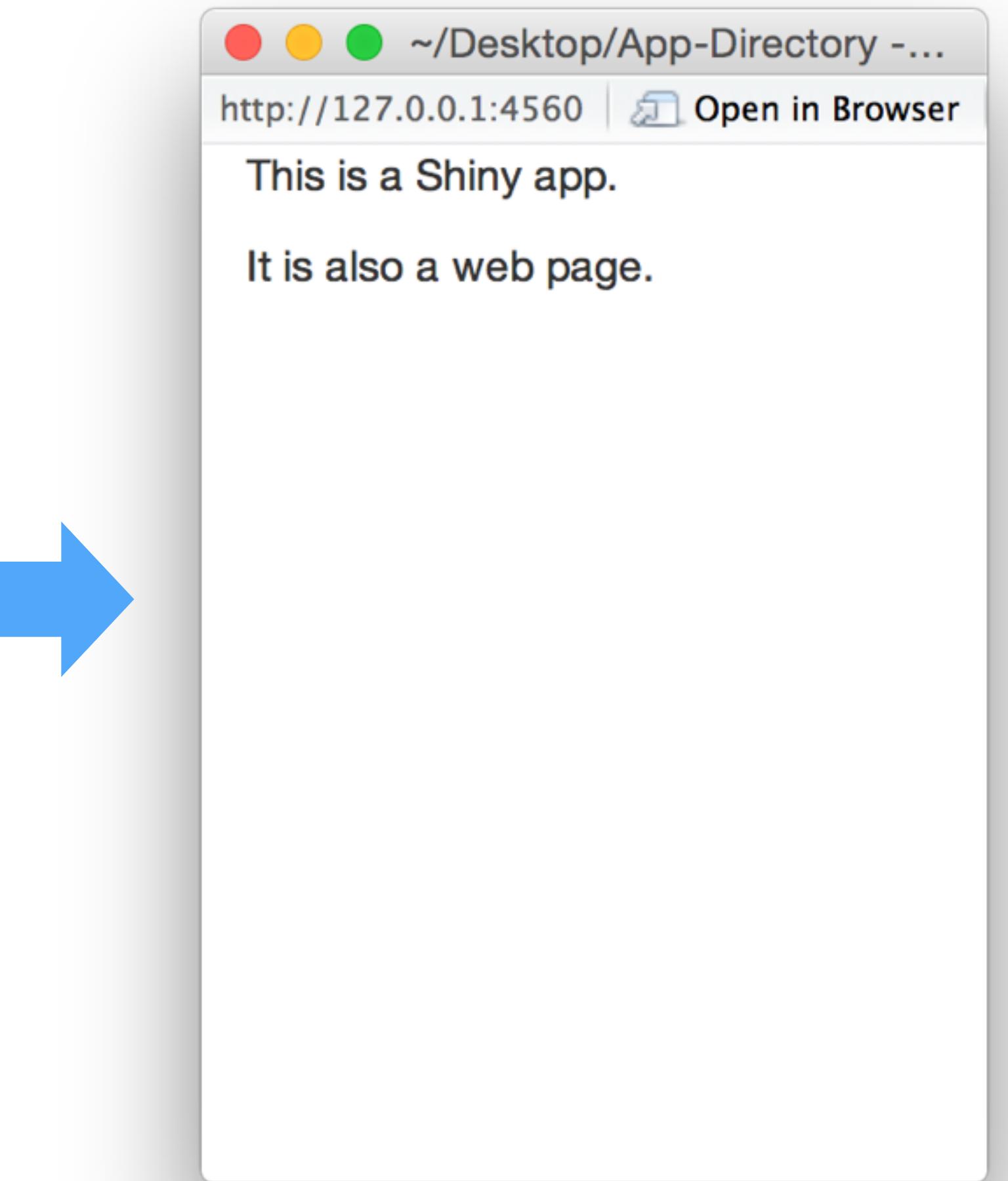
```
fluidPage(  
  "This is a Shiny app.",  
  "It is also a web page."  
)
```



p()

A new paragraph

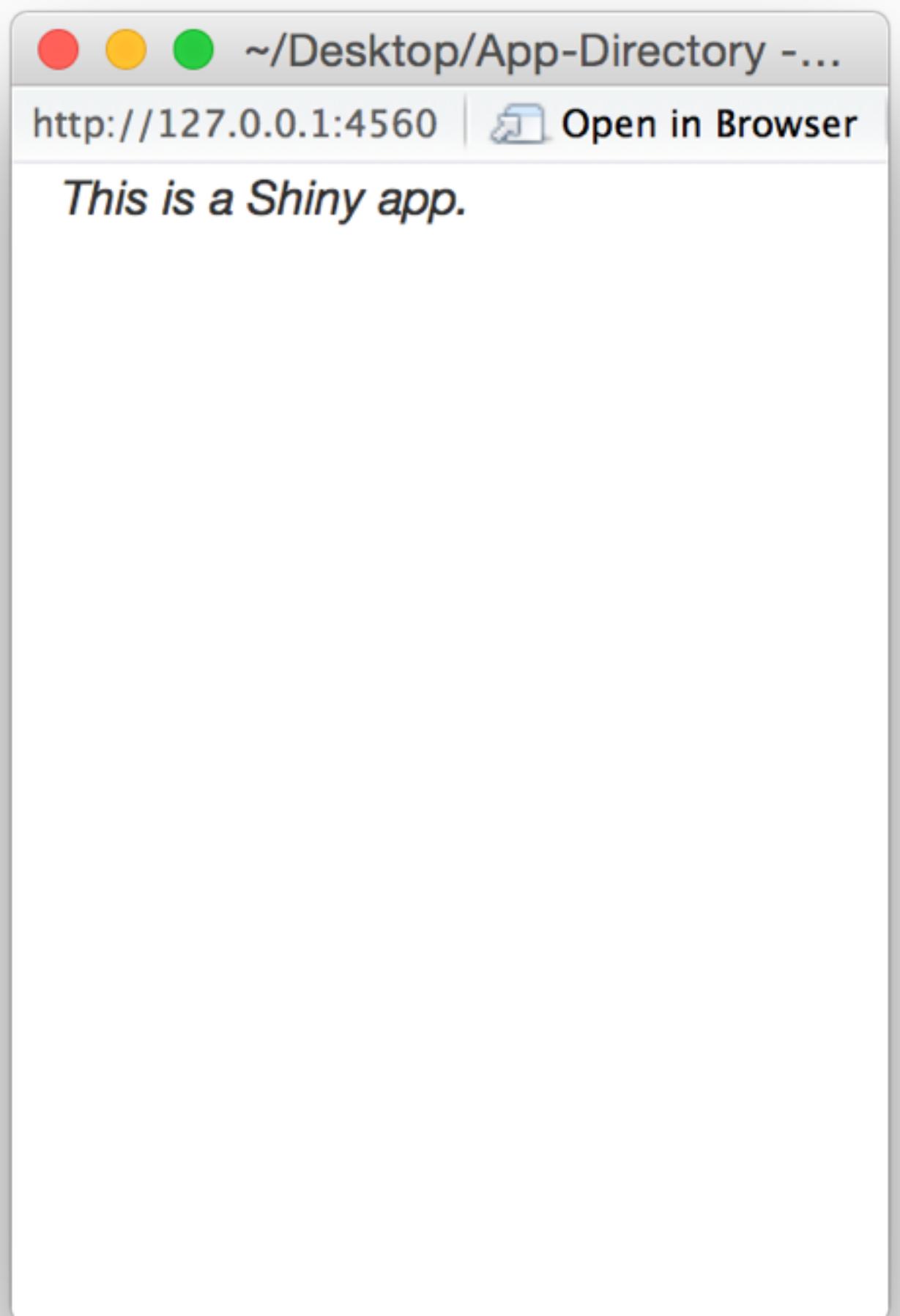
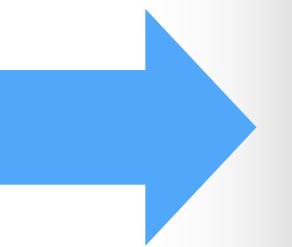
```
fluidPage(  
  tags$p("This is a Shiny app."),  
  tags$p("It is also a web page.")  
)
```



em()

Emphasized (*italic*) text

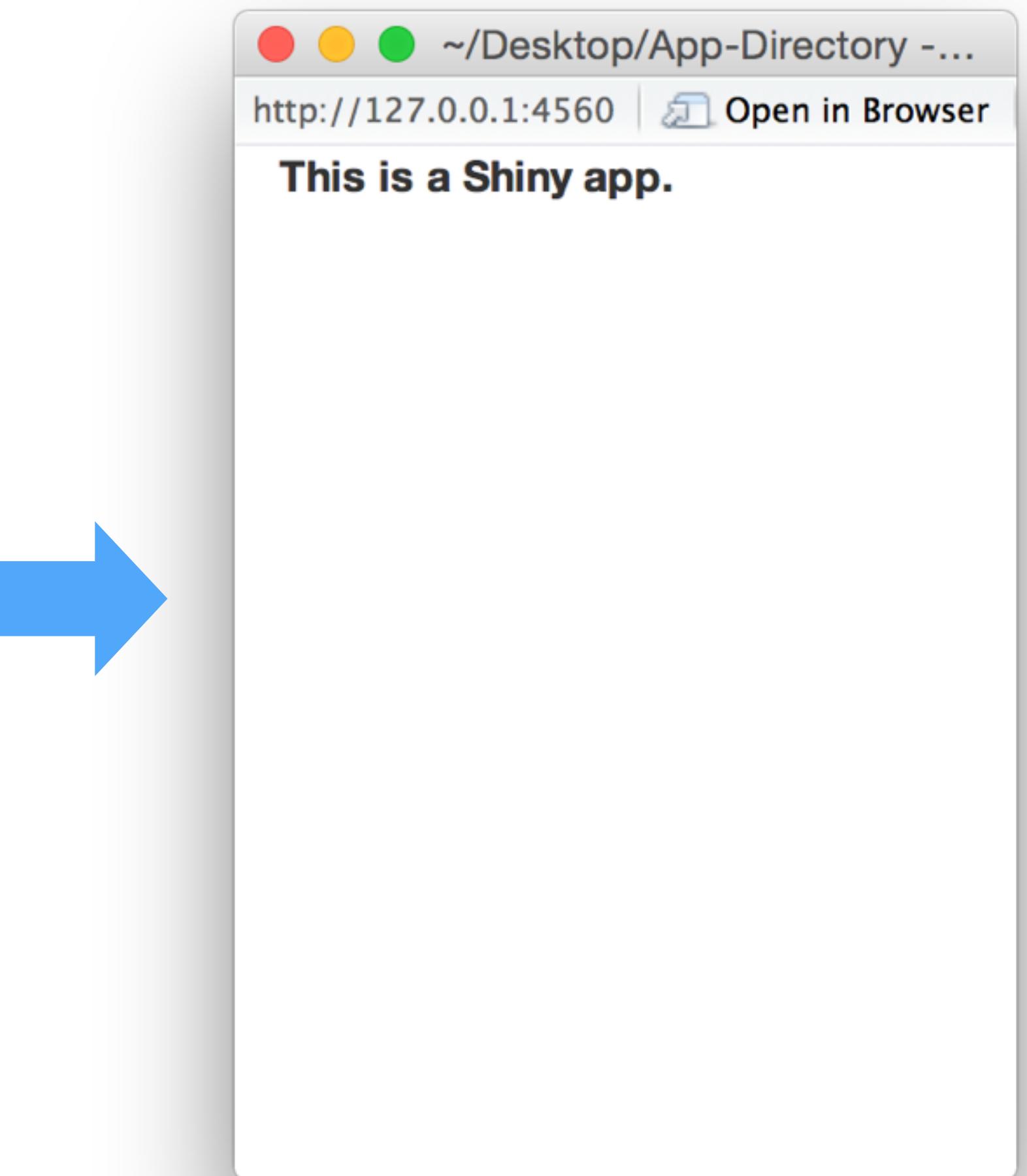
```
fluidPage(  
  tags$em("This is a Shiny app."))
```



strong()

Strong (**bold**) text

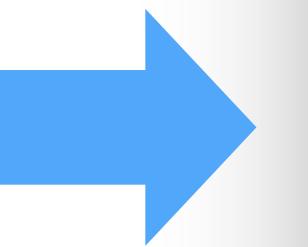
```
fluidPage(  
  tags$strong("This is a Shiny app.")  
)
```



code()

Monospaced text (code)

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



nesting

You can also nest functions inside of others

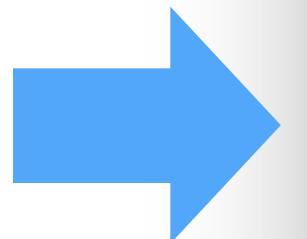
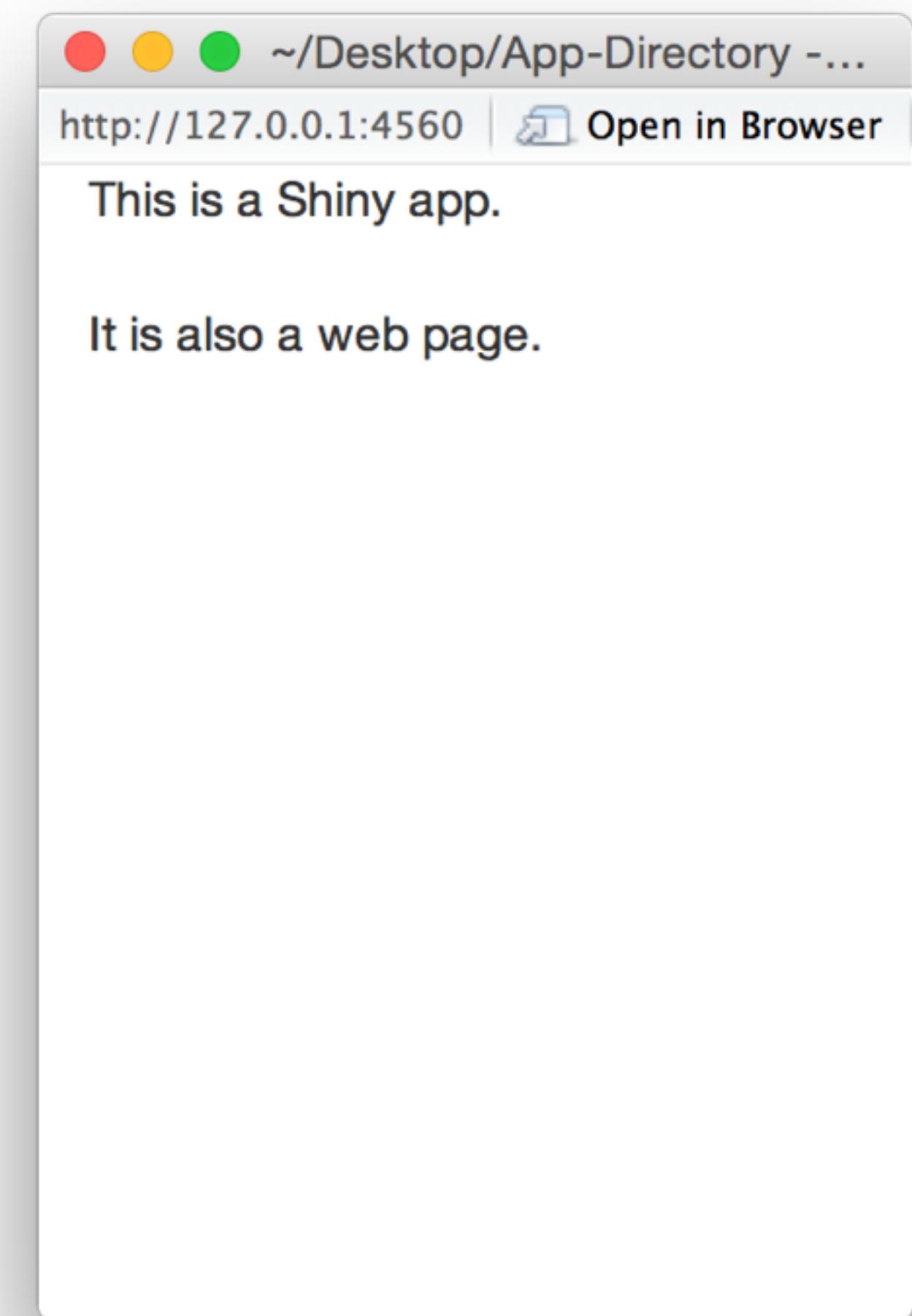
```
fluidPage(  
  tags$p("This is a",  
         tags$strong("Shiny"),  
         "app."))
```



br()

A line break

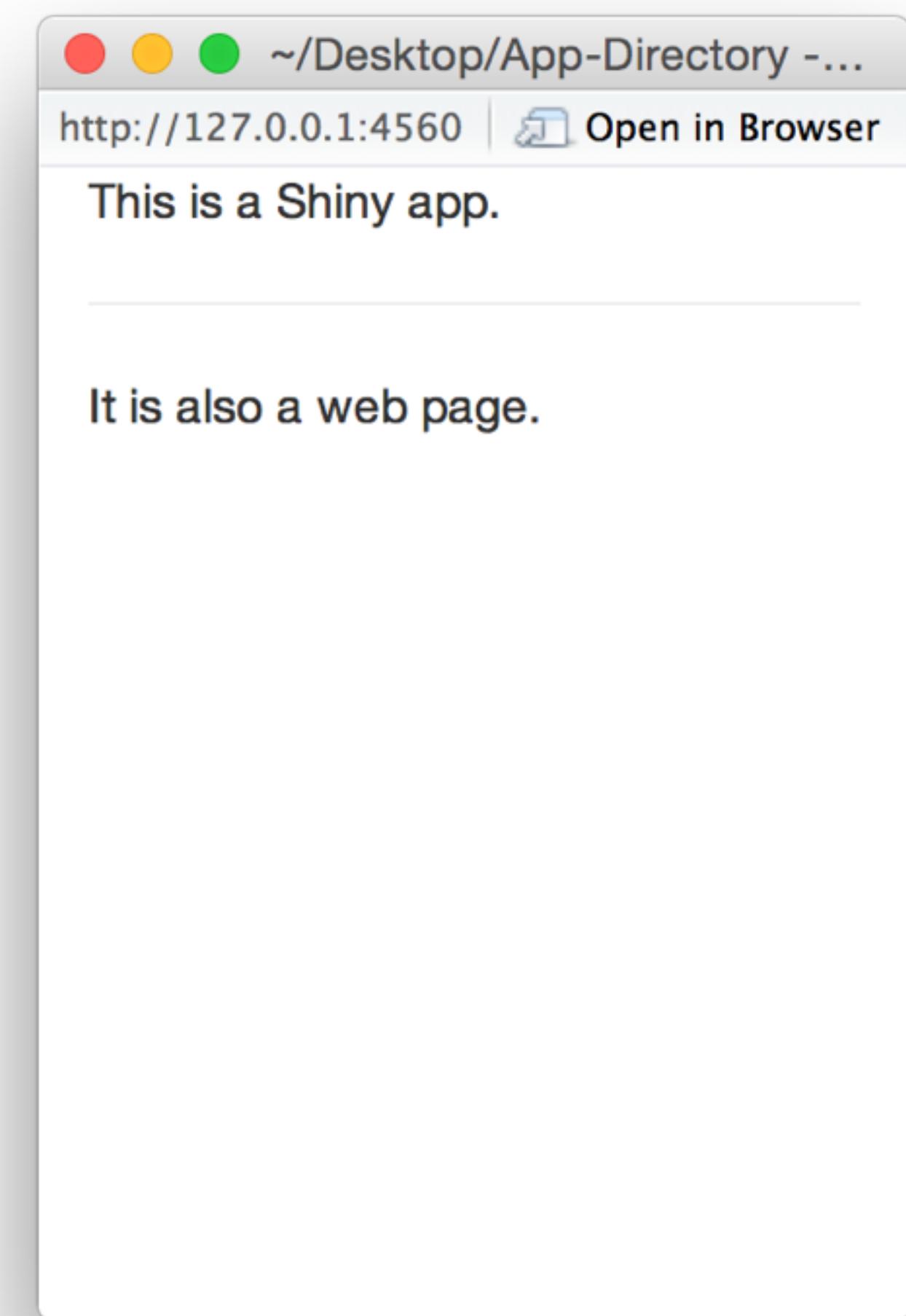
```
fluidPage(  
  "This is a Shiny app.",  
  tags$br(),  
  "It is also a web page."  
)
```



hr()

A horizontal rule

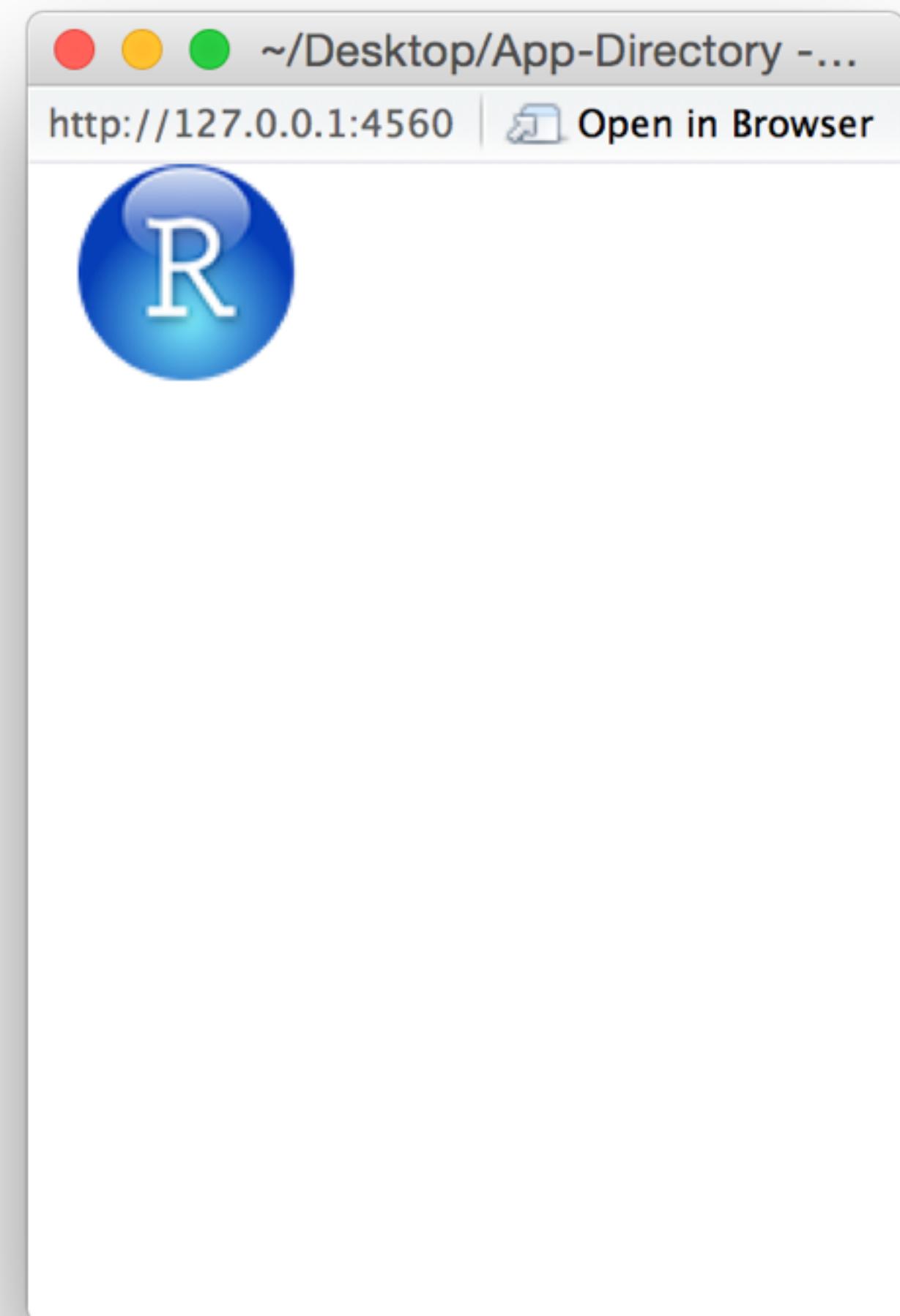
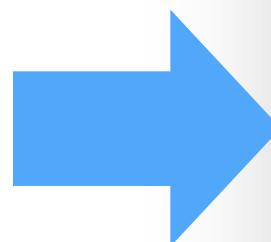
```
fluidPage(  
  "This is a Shiny app.",  
  tags$hr(),  
  "It is also a web page."  
)
```



img()

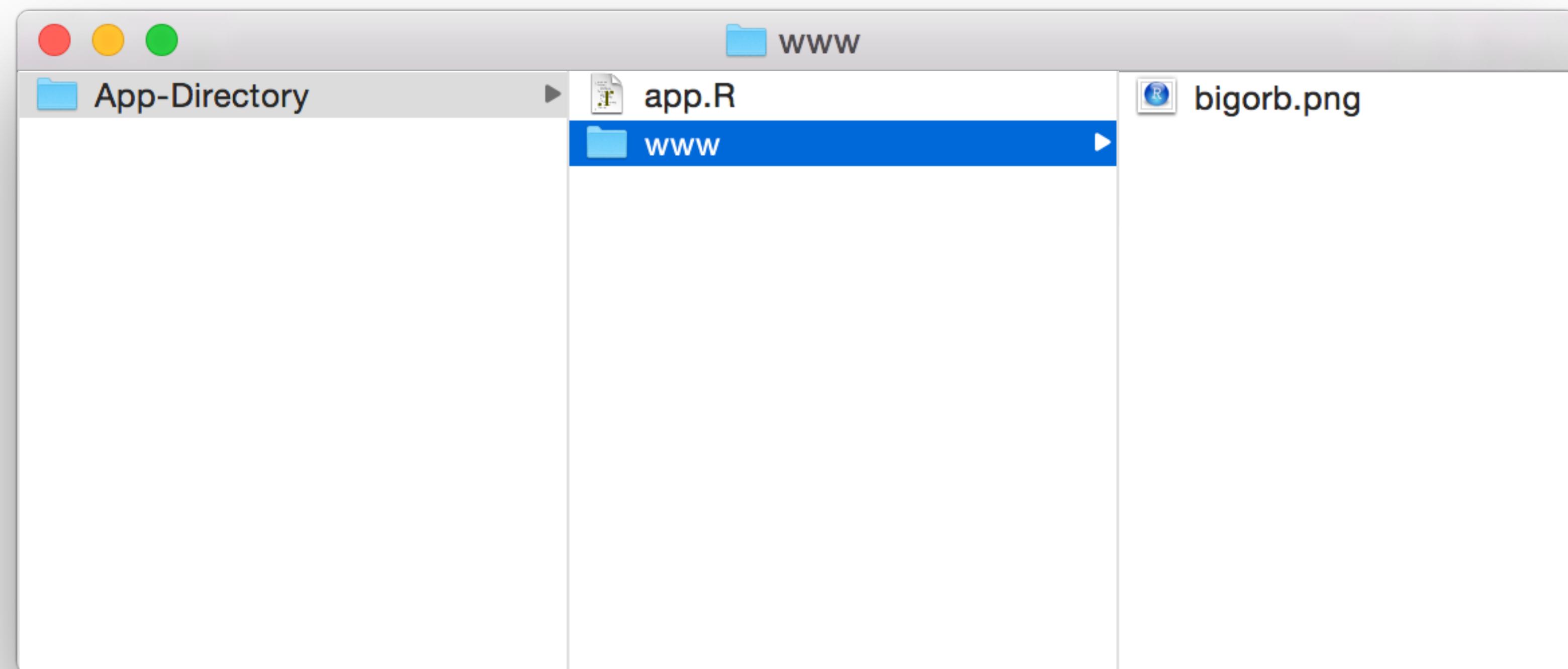
Add an image. Use **src** argument to point to the image URL.

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



Adding Images

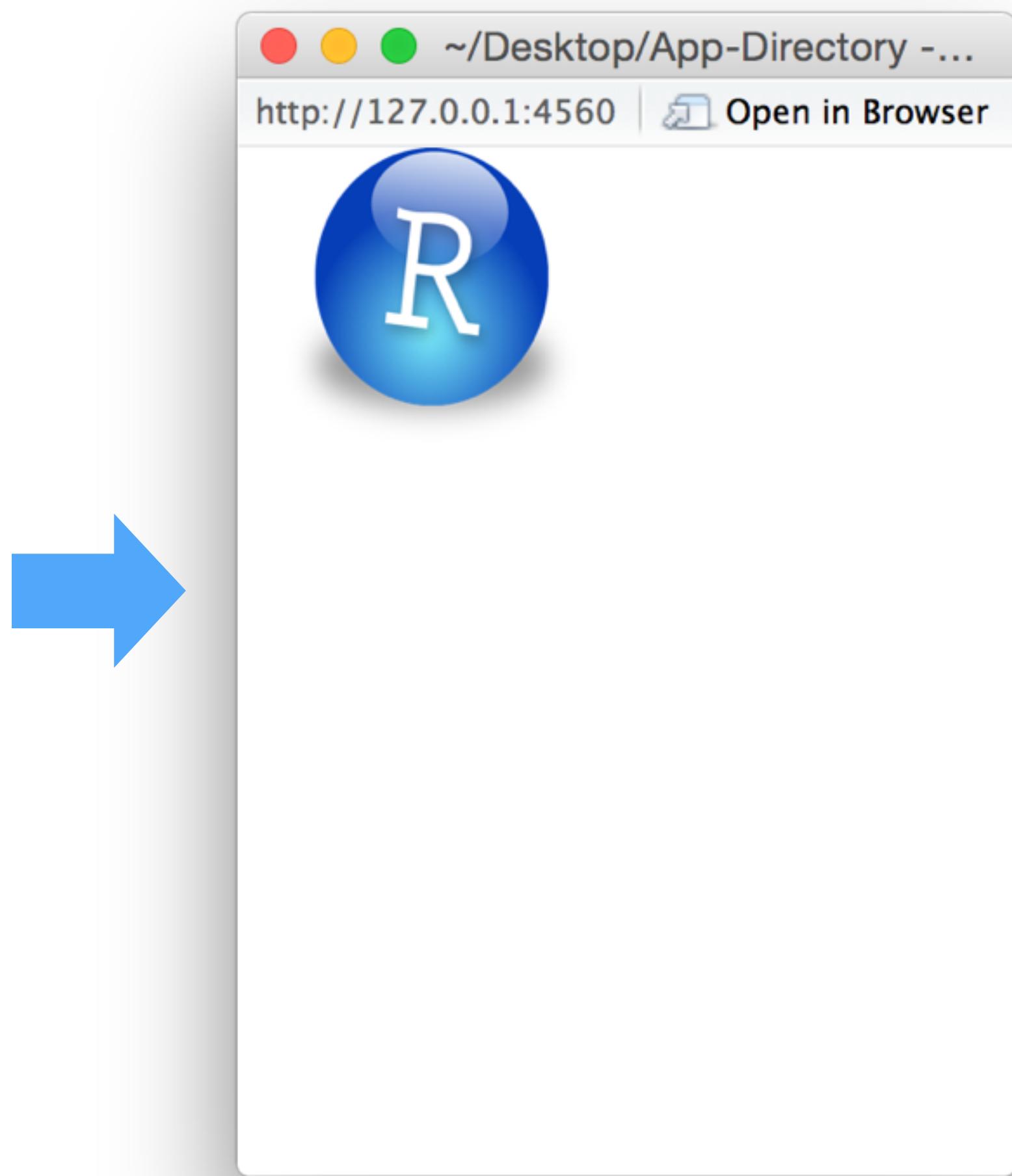
To add an image from a file, save the file in a subdirectory named **www**



img()

Add an img from a file in www

```
fluidPage(  
  tags$img(height = 100,  
            width = 100,  
            src = "bigorb.png")  
)
```



Some tags functions come with a wrapper function, so you do not need to call tags\$

```
h1
```

```
function (...)  
tags$h1(...)  
<environment: namespace:htmltools>
```

Function	Creates
a()	A Hyperlink
br()	A line break
code()	Text formatted like computer code
em()	Italicized (emphasized) text
h1(), h2(), h3(), h4(), h5(), h6()	Headers (First level to sixth)
hr()	A horizontal rule (line)
img()	An image
p()	A new paragraph
strong()	Bold (strong) text

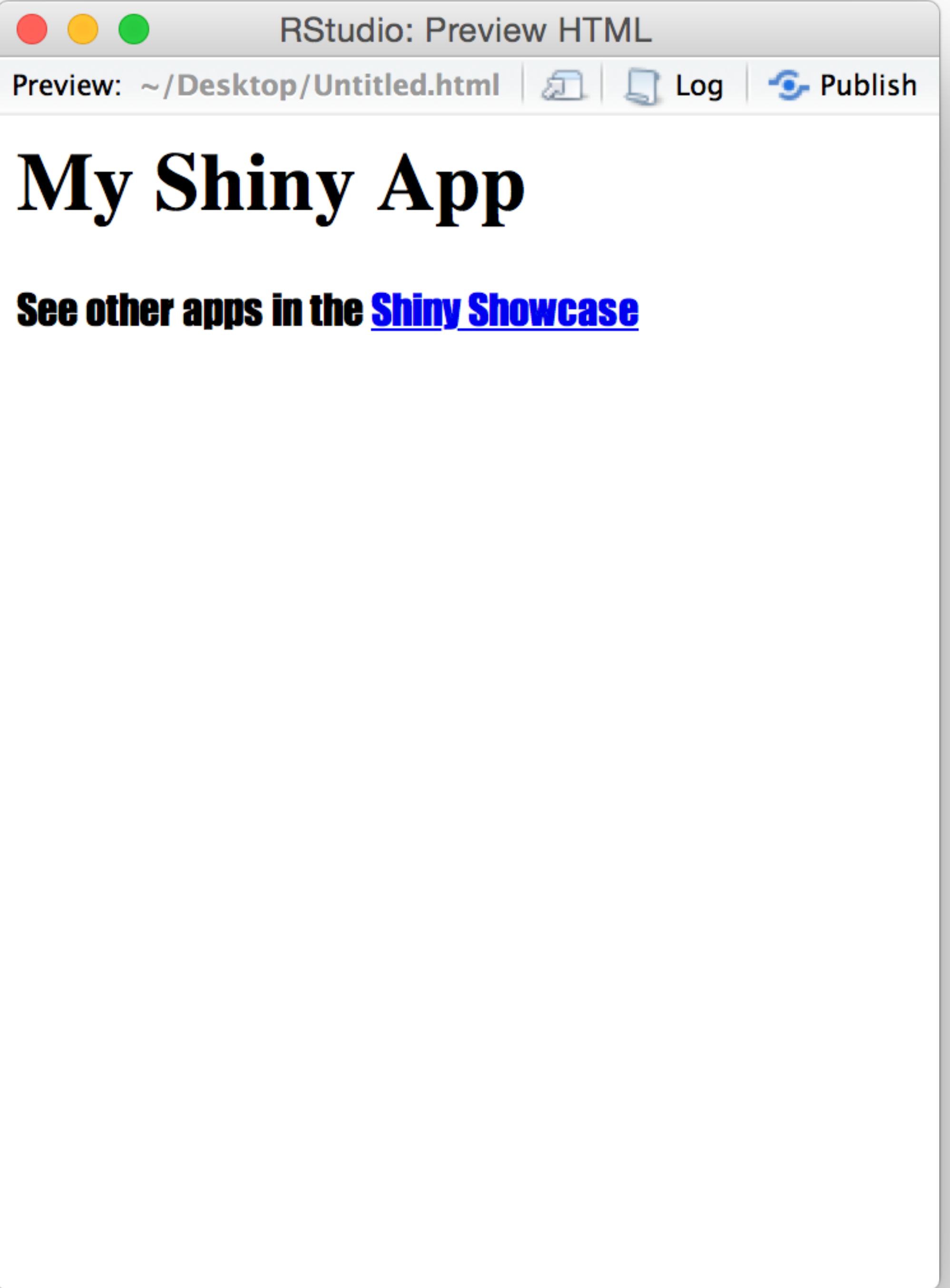
```
<div class="container-fluid">  
  <h1>My Shiny App</h1>  
  <p style="font-family:Impact">  
    See other apps in the  
    <a href="http://www.rstudio.com/  
      products/shiny/shiny-user-  
      showcase/">Shiny Showcase</a>  
  </p>  
</div>
```

RStudio: Preview HTML

Preview: ~/Desktop/Untitled.html | Log | Publish

My Shiny App

See other apps in the [Shiny Showcase](#)



The screenshot shows the RStudio interface with the 'Preview' tab selected. The preview window displays the generated HTML code as follows:

```
<div class="container-fluid">  
  <h1>My Shiny App</h1>  
  <p style="font-family:Impact">  
    See other apps in the  
    <a href="http://www.rstudio.com/  
      products/shiny/shiny-user-  
      showcase/">Shiny Showcase</a>  
  </p>  
</div>
```

The output is rendered with the following styling:

- The title "My Shiny App" is displayed in a large, bold, black font.
- The descriptive text "See other apps in the Shiny Showcase" is displayed in a standard black font.
- The link "Shiny Showcase" is displayed in blue and is underlined, indicating it is a hyperlink.

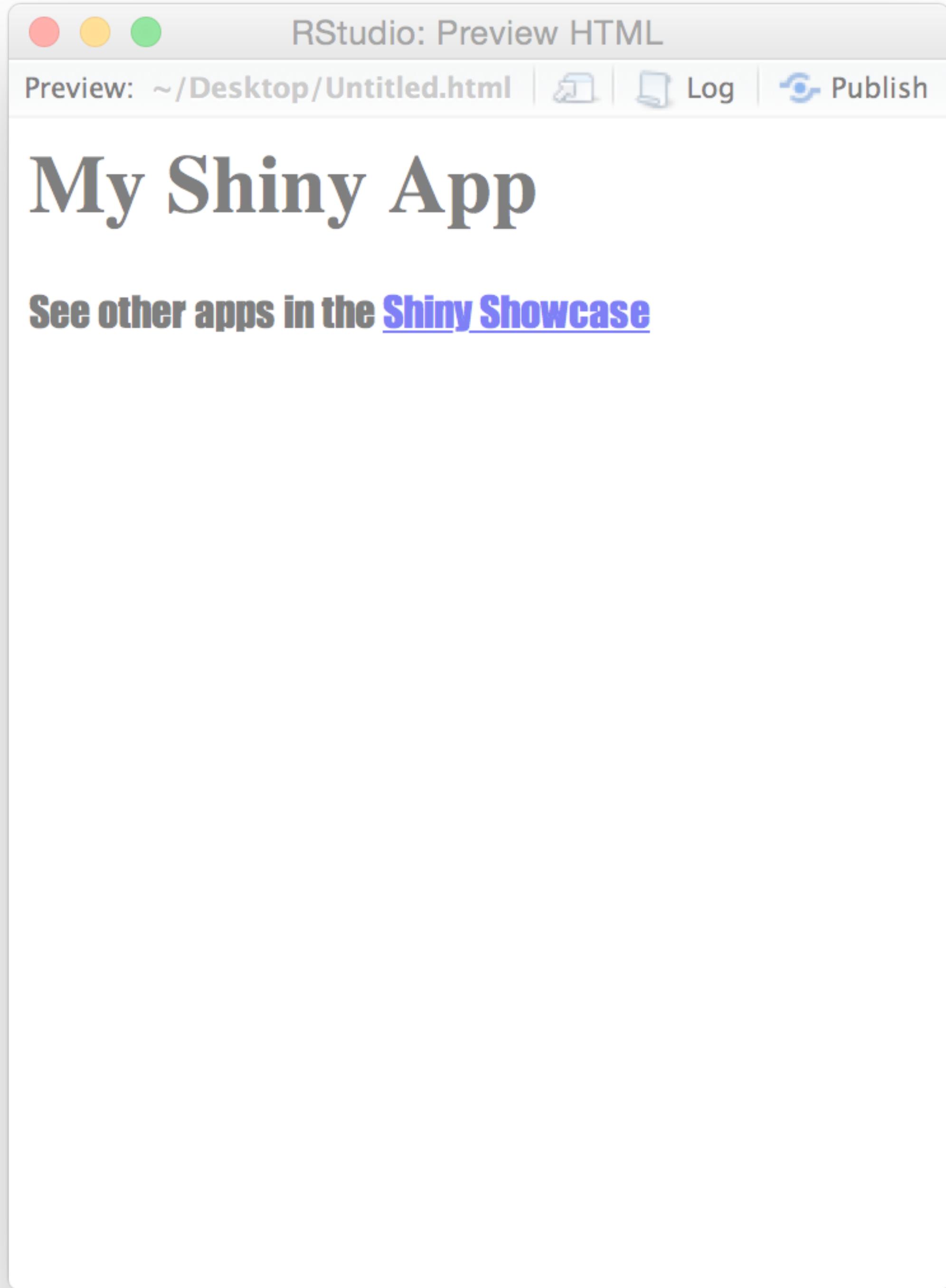
```
fluidPage(  
  <h1>My Shiny App</h1>  
  <p style="font-family:Impact">  
    See other apps in the  
    <a href="http://www.rstudio.com/  
      products/shiny/shiny-user-  
      showcase/">Shiny Showcase</a>  
  </p>  
)
```

RStudio: Preview HTML

Preview: ~/Desktop/Untitled.html | Log | Publish

My Shiny App

See other apps in the [Shiny Showcase](#)



```
fluidPage(  
  h1("My Shiny App"),  
  <p style="font-family:Impact">  
    See other apps in the  
    <a href="http://www.rstudio.com/  
      products/shiny/shiny-user-  
      showcase/">Shiny Showcase</a>  
  </p>  
)
```

RStudio: Preview HTML

Preview: ~/Desktop/Untitled.html | Log | Publish

My Shiny App

See other apps in the [Shiny Showcase](#)

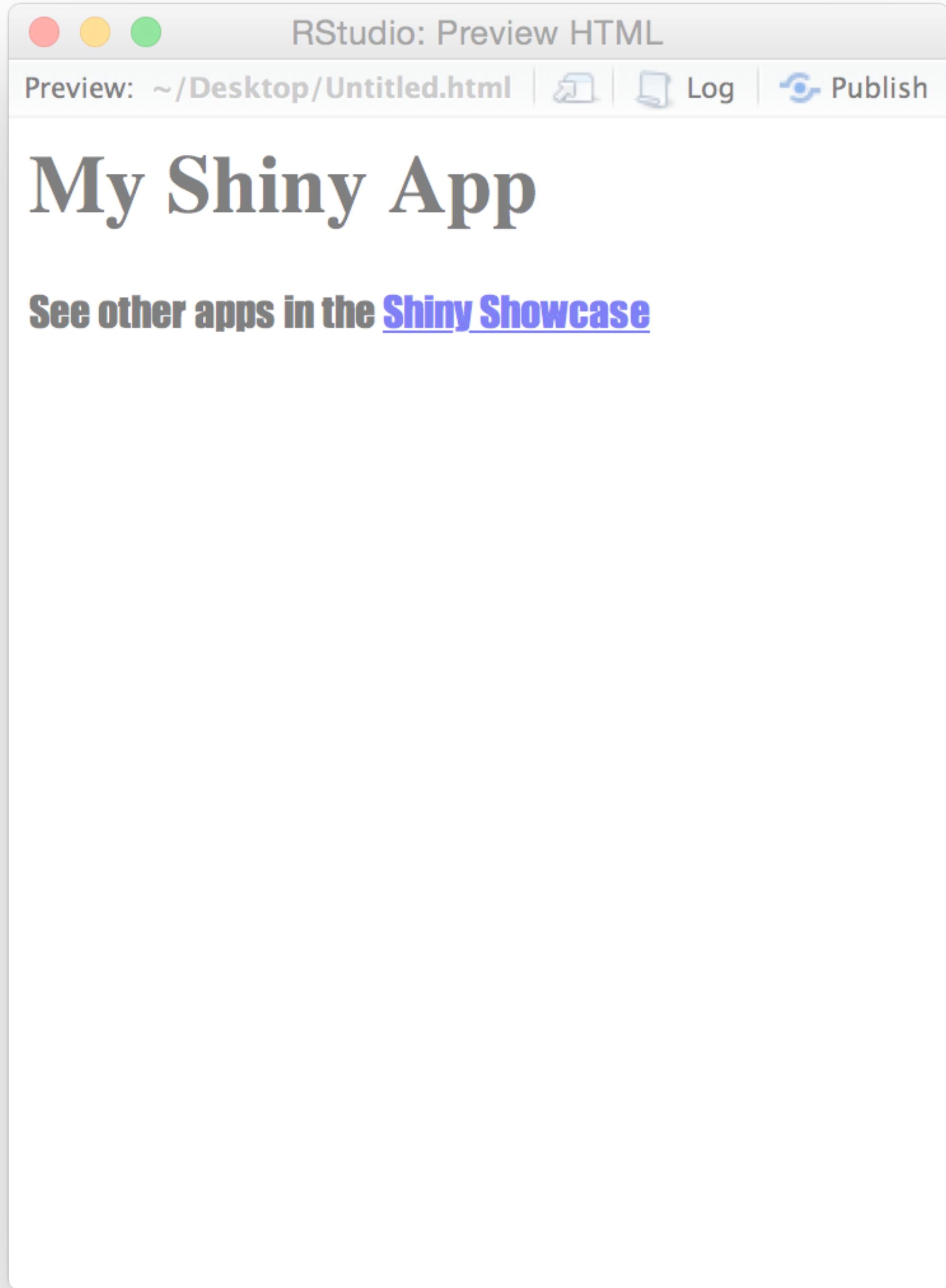
```
fluidPage(  
  h1("My Shiny App"),  
  p(style = "font-family:Impact",  
    "See other apps in the",  
    <a href="http://www.rstudio.com/  
      products/shiny/shiny-user-  
      showcase/">Shiny Showcase</a>  
)  
)
```

RStudio: Preview HTML

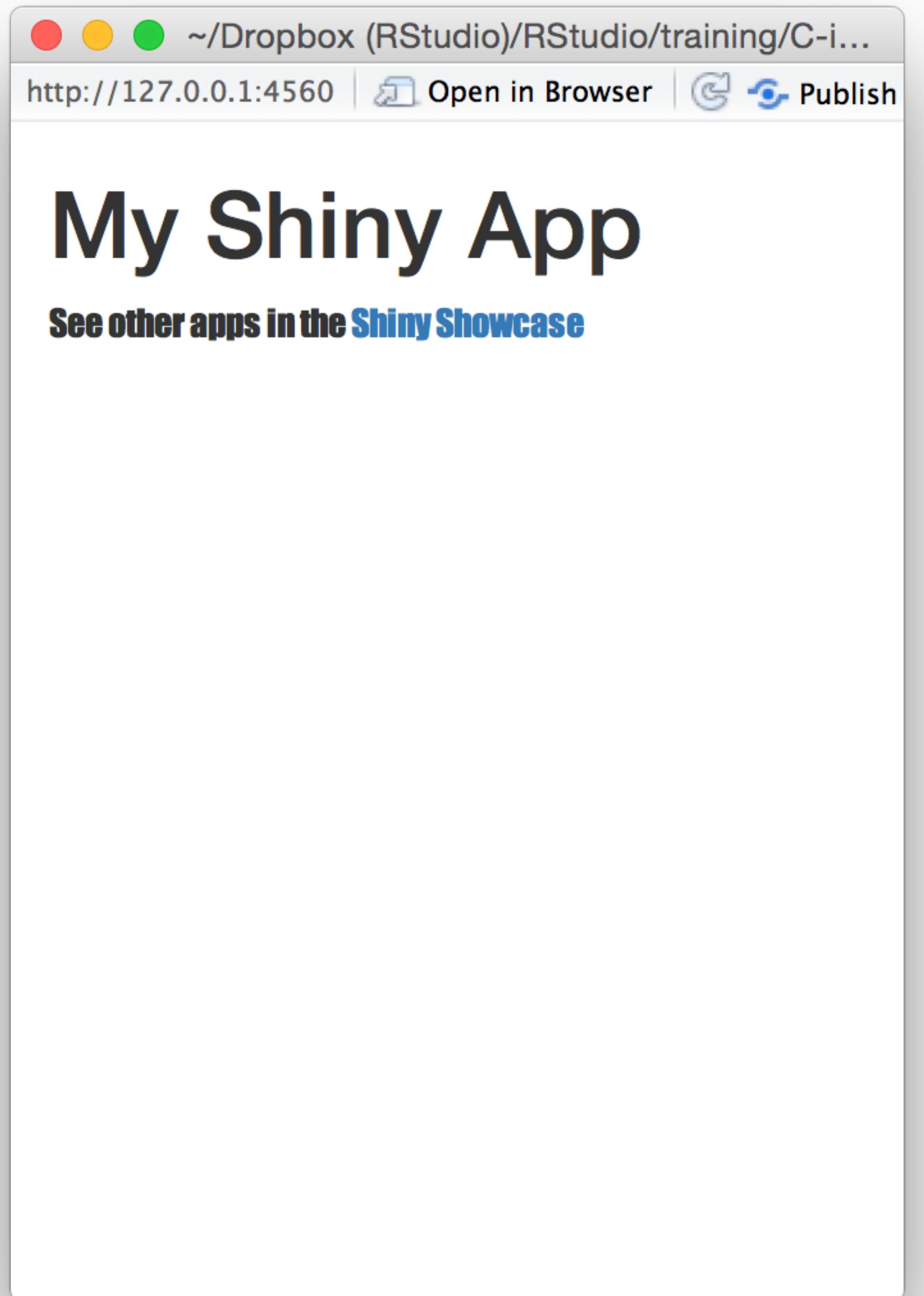
Preview: ~/Desktop/Untitled.html | Log | Publish

My Shiny App

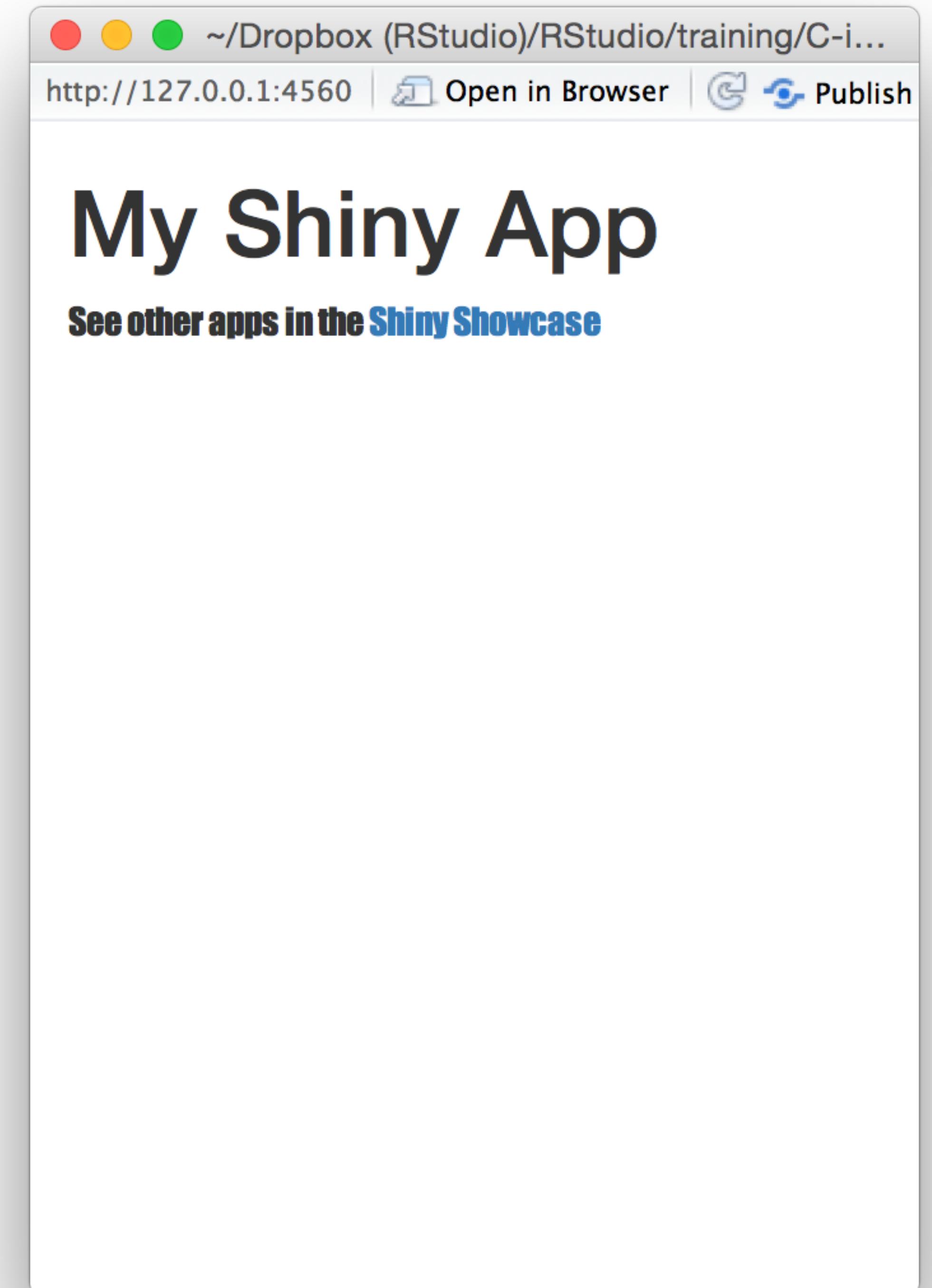
See other apps in the [Shiny Showcase](#)



```
fluidPage(  
  h1("My Shiny App"),  
  p(style = "font-family:Impact",  
    "See other apps in the",  
    a("Shiny Showcase",  
      href = "http://www.rstudio.com/  
products/shiny/shiny-user-showcase/"))  
)  
)
```



```
fluidPage(  
  tags$h1("My Shiny App"),  
  tags$p(style = "font-family:Impact",  
         "See other apps in the",  
  tags$a("Shiny Showcase",  
        href = "http://www.rstudio.com/  
products/shiny/shiny-user-showcase/"))  
)  
)
```



```
fluidPage(
```

```
<div class="container-fluid">
  <h1>My Shiny App</h1>
  <p style="font-family:Impact">
    See other apps in the
    <a href="http://www.rstudio.com/
      products/shiny/shiny-user-
      showcase/">Shiny Showcase</a>
  </p>
</div>
```

```
)
```

My Shiny App

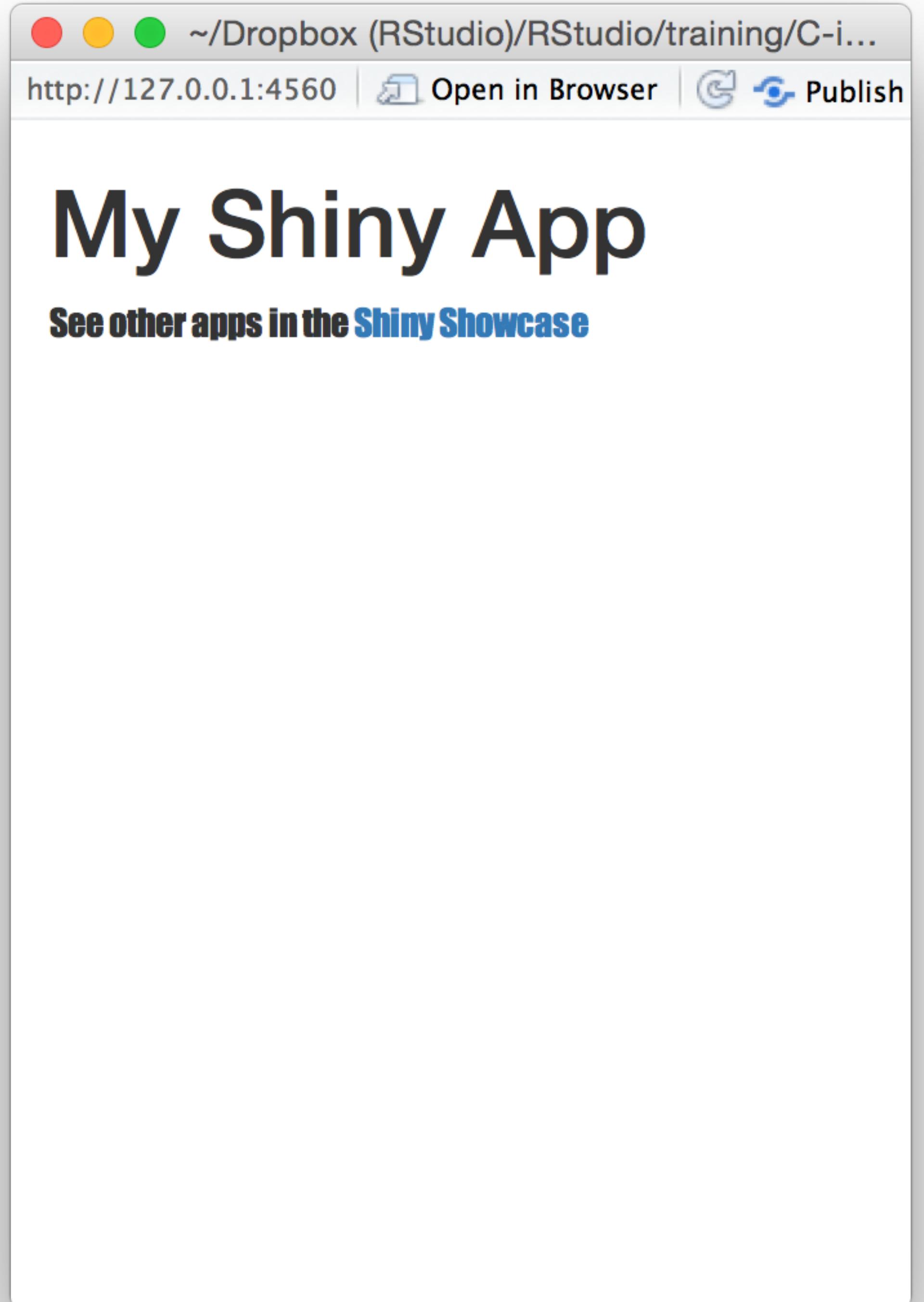
See other apps in the [Shiny Showcase](#)

http://127.0.0.1:4560 | [Open in Browser](#) | [Publish](#)

```
fluidPage(
```

```
'<div class="container-fluid">
  <h1>My Shiny App</h1>
  <p style="font-family:Impact">
    See other apps in the
    <a href="http://www.rstudio.com/
      products/shiny/shiny-user-
      showcase/">Shiny Showcase</a>
  </p>
</div>'
```

```
)
```



```
fluidPage(  
  HTML(  
    '<div class="container-fluid">  
      <h1>My Shiny App</h1>  
      <p style="font-family:Impact">  
        See other apps in the  
        <a href="http://www.rstudio.com/  
          products/shiny/shiny-user-  
          showcase/">Shiny Showcase</a>  
      </p>  
    </div>'  
)  
)
```

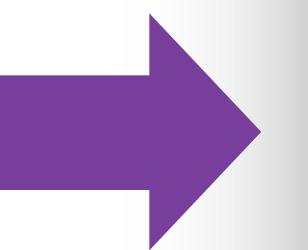
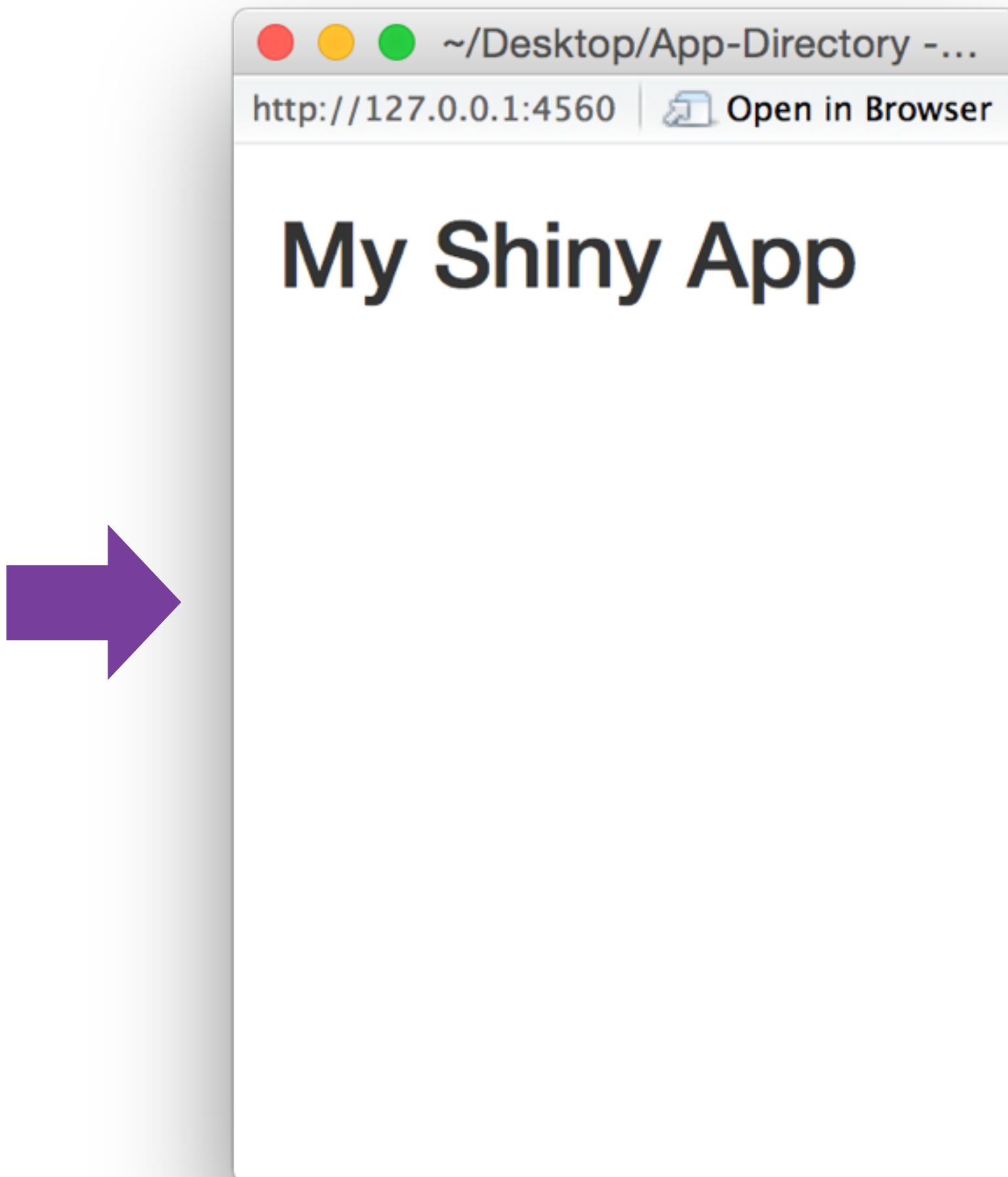
My Shiny App

See other apps in the [Shiny Showcase](#)

Raw HTML

Use `HTML()` to pass a character string as raw HTML

```
fluidPage(  
  HTML("<h1>My Shiny App</h1>")  
)
```



Recap: Reactive values

```
names(tags)
```

```
## [1] "a"          "abbr"       "address"    "area"
## [5] "article"    "aside"      "audio"      "br"
## [9] "b"          "bdi"        "bdo"        "blockquote"
## [13] "body"       "br"         "button"     "canvas"
## [17] "caption"    "cite"       "code"       "col"
## [21] "colgroup"   "command"    "datalist"   "del"
## [25] "div"        "em"         "form"      "hr"
## [29] "div"        "i"          "input"     "label"
## [33] "embed"     "labelset"   "menuitem"  "p"
## [37] "figure"    "menuitem"  "noscript"  "ol"
## [41] "h2"        "p"         "pre"       "strong"
## [45] "h3"        "pre"       "script"    "span"
```

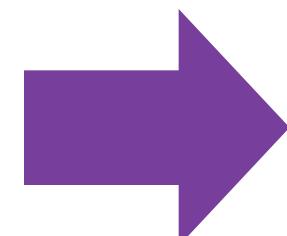
Add elements with the **tags\$** functions

```
<p id="foo">
  foo
</p>
```

unnamed arguments are passed into
HTML tags

```
<p id="foo">
  foo
</p>
```

named arguments are passed as HTML tag
attributes



Add raw html with **HTML()**

HTML UI

shiny.rstudio.com/articles/html-ui.html

Build your entire UI from HTML

```
<html>

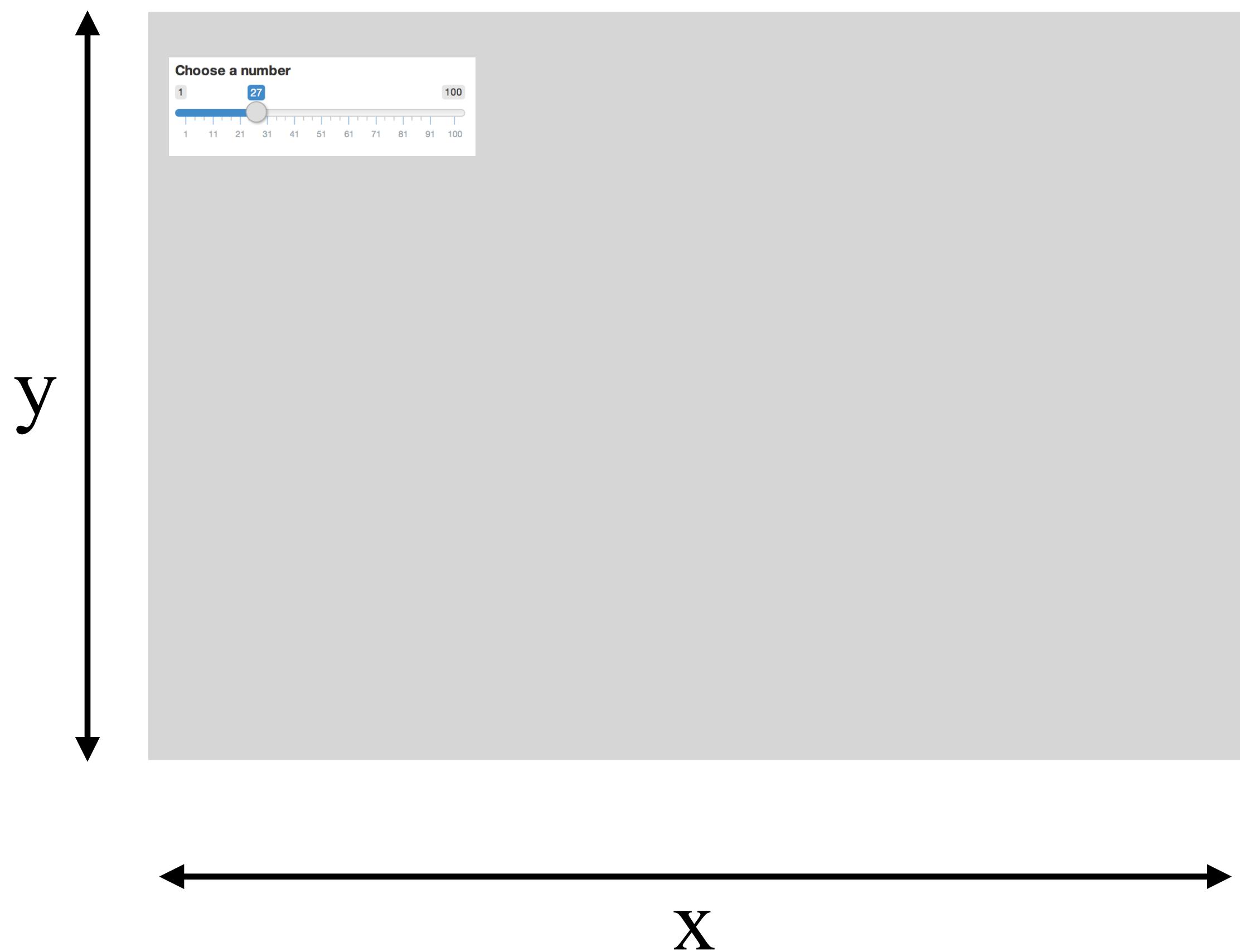
  <head>
    <script src="shared/jquery.js" type="text/javascript"></script>
    <script src="shared/shiny.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css" href="shared/shiny.css"/>
  </head>

  <body>
    <h1>HTML UI</h1>

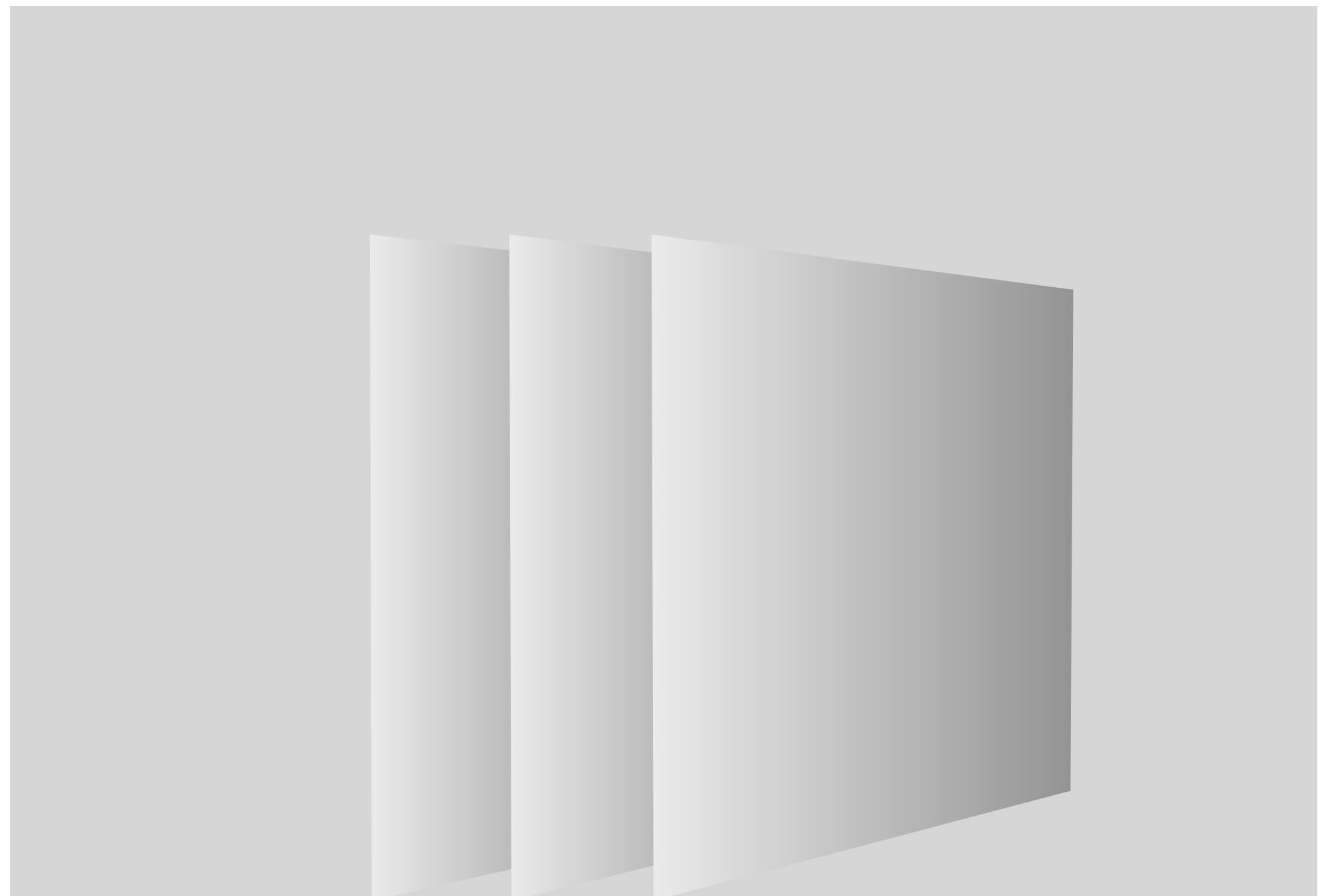
    <p>
      <label>Distribution type:</label><br />
```

Create a
layout

Use layout functions to position elements within your app



Use layout functions to position elements within your app



Layout functions

Add HTML that divides the UI into a grid

`fluidRow()`

```
<div class="row"></div>
```

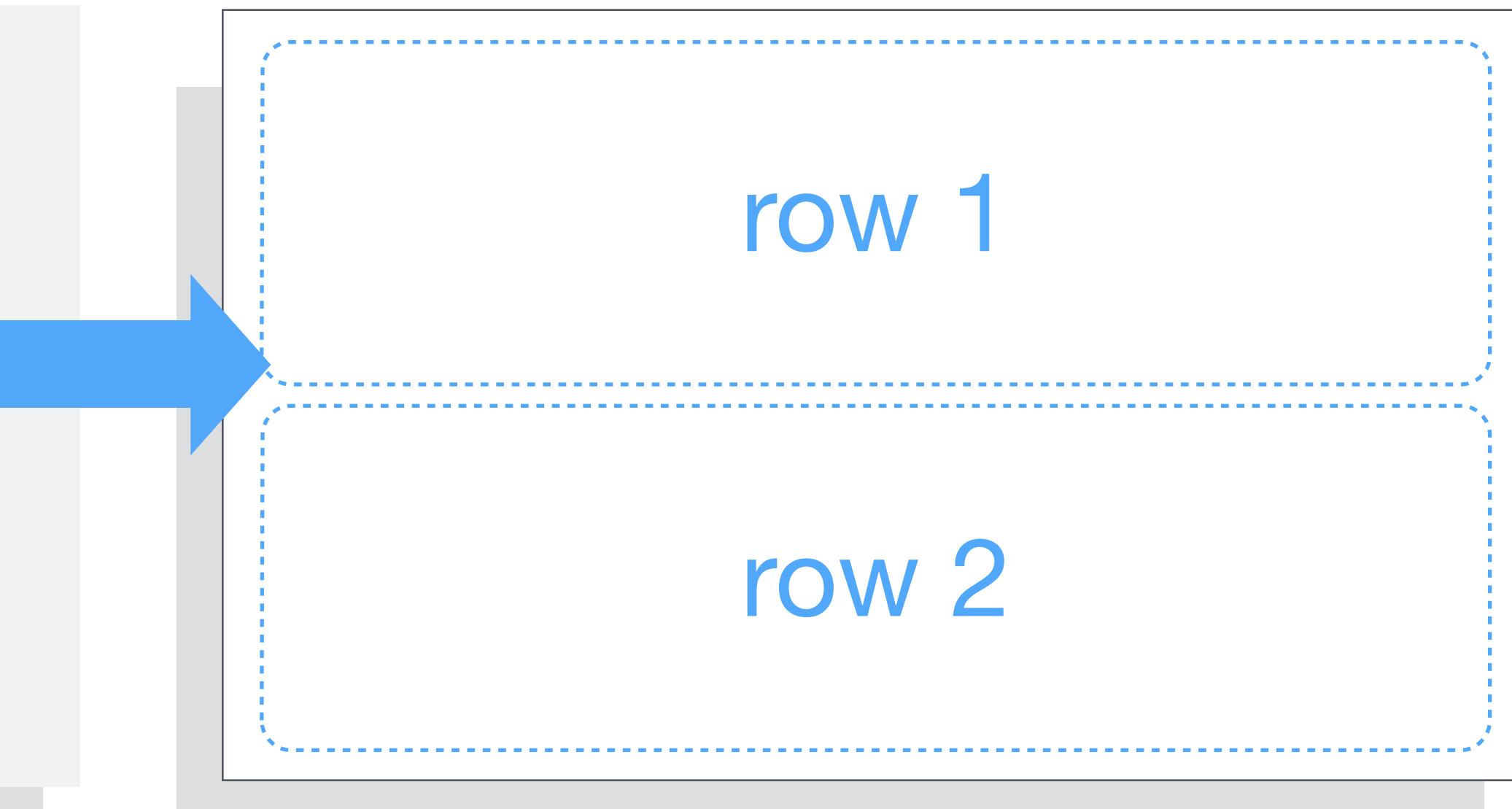
`column(width = 2)`

```
<div class="col-sm-2"></div>
```

fluidRow()

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

```
ui <- fluidPage(  
  fluidRow(),  
  fluidRow()  
)
```

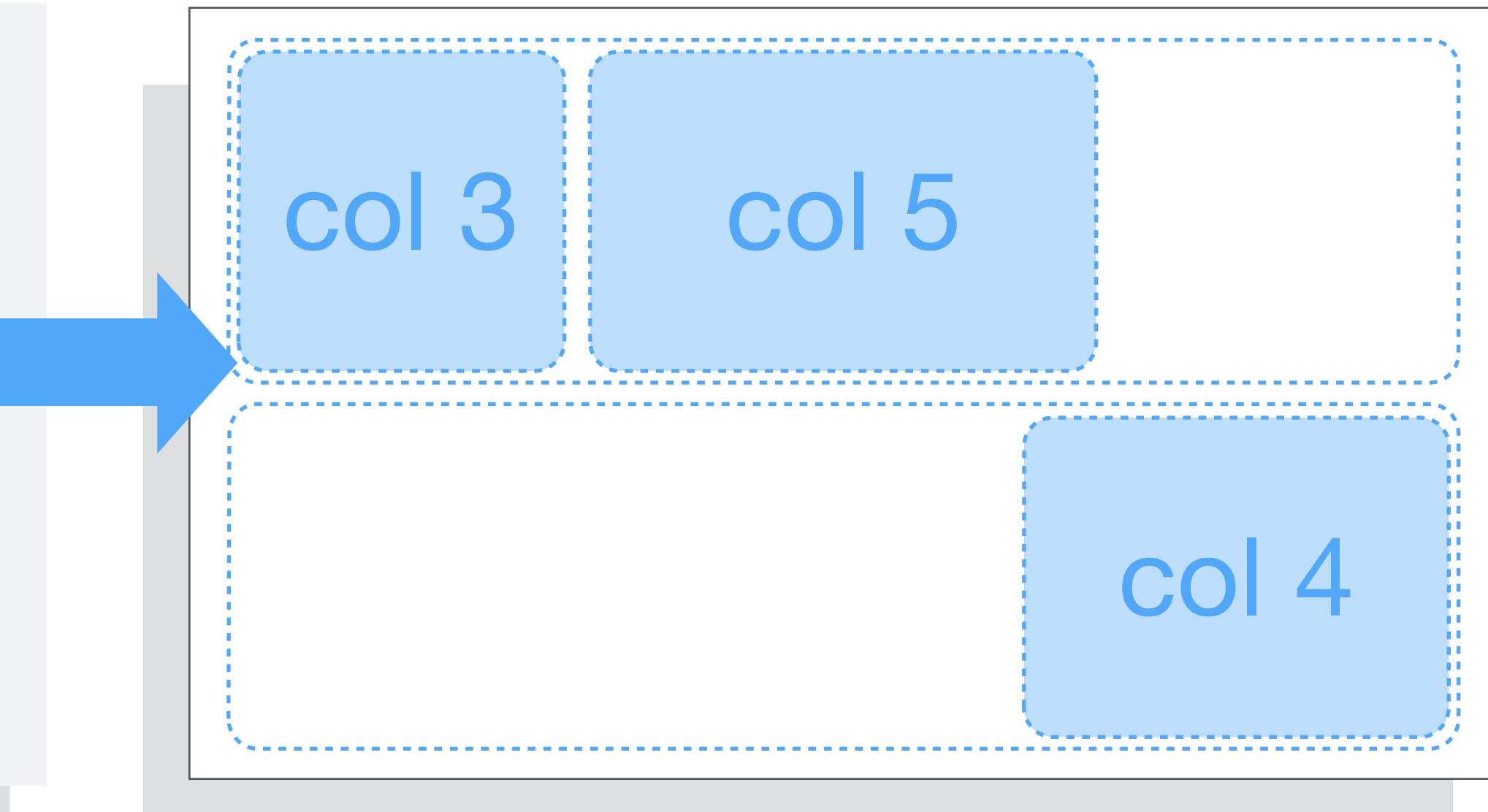


column()

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

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



To place an element in the grid, call it as an argument of a layout function

`fluidRow()`

```
<div class="row"></div>
```

`column(2)`

```
<div class="col-sm-2">  
</div>
```

To place an element in the grid, call it as an argument of a layout function

```
fluidRow("In the row")
```

```
<div class="row"></div>
```

```
column(2)
```

```
<div class="col-sm-2">  
</div>
```

To place an element in the grid, call it as an argument of a layout function

```
fluidRow("In the row")
```

```
<div class="row">In the row</div>
```

```
column(2)
```

```
<div class="col-sm-2">  
</div>
```

To place an element in the grid, call it as an argument of a layout function

```
fluidRow("In the row")
```

```
<div class="row">In the row</div>
```

```
column(2, plotOutput("hist"))
```

```
<div class="col-sm-2">  
</div>
```

To place an element in the grid, call it as an argument of a layout function

```
fluidRow("In the row")
```

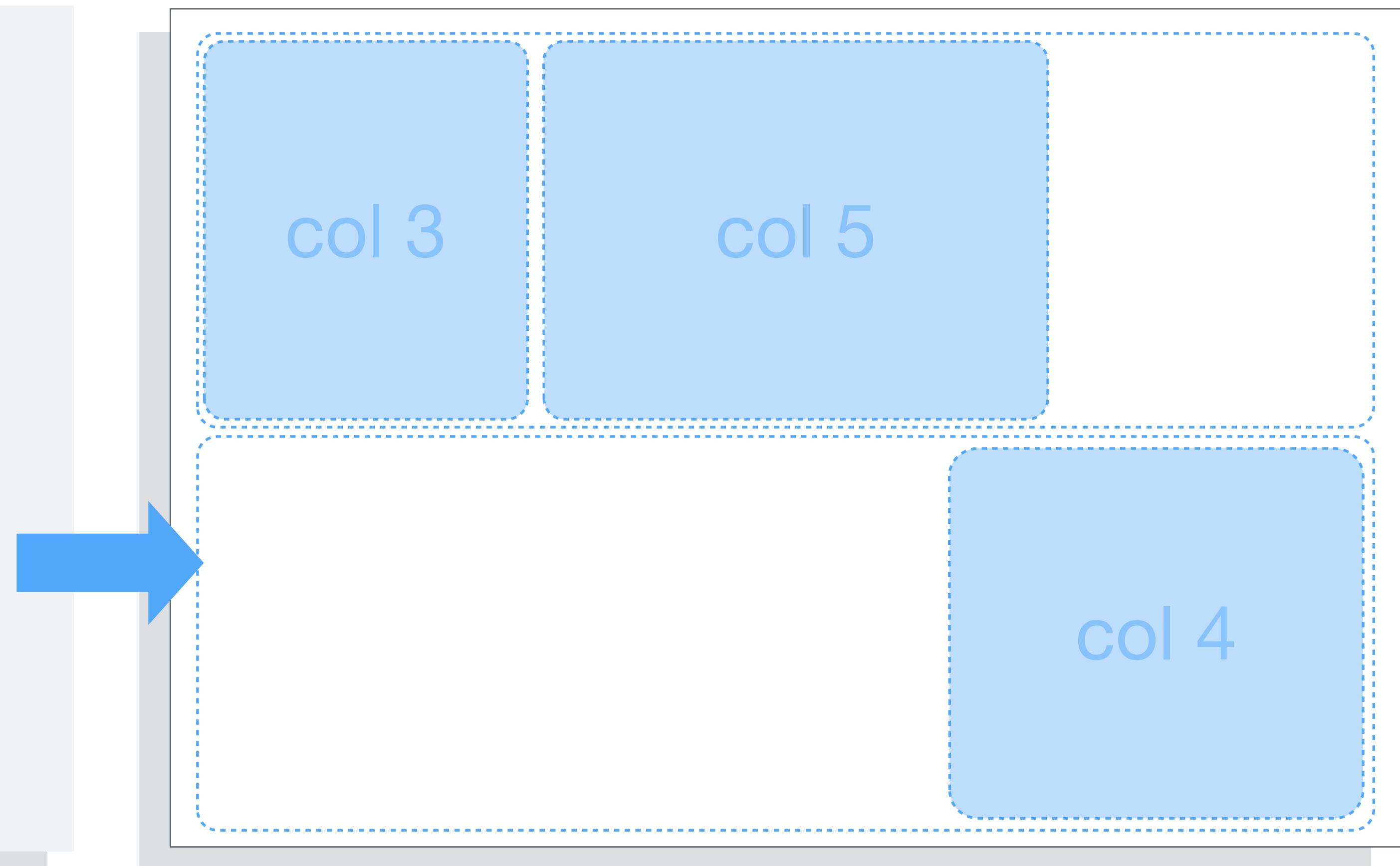
```
<div class="row">In the row</div>
```

```
column(2, plotOutput("hist"))
```

```
<div class="col-sm-2">  
  <div id="hist" class="shiny-plot-output"  
    style="width: 100% ; height: 400px"></div>  
</div>
```

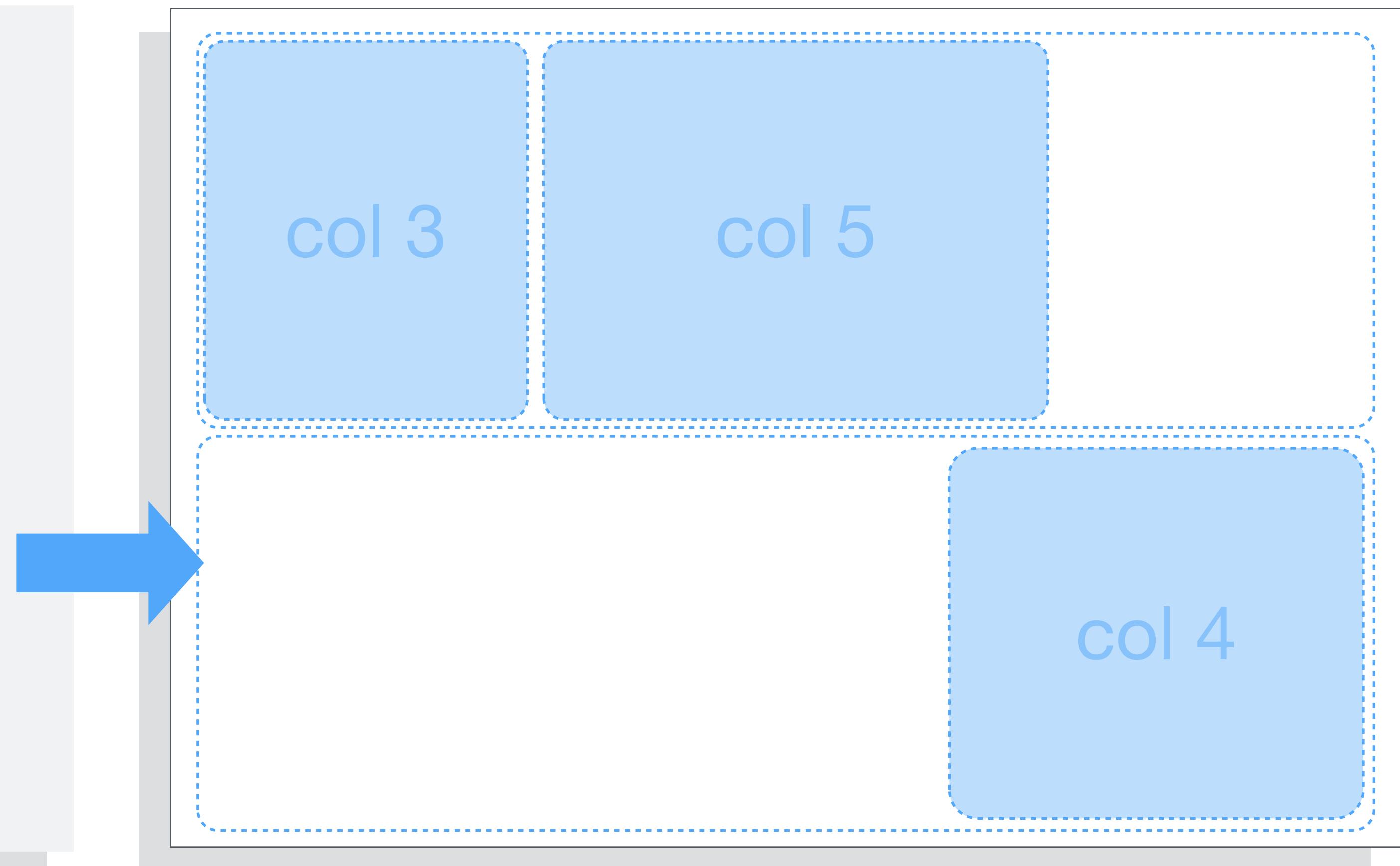
To place an element in the grid, call it as an argument of a layout function

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



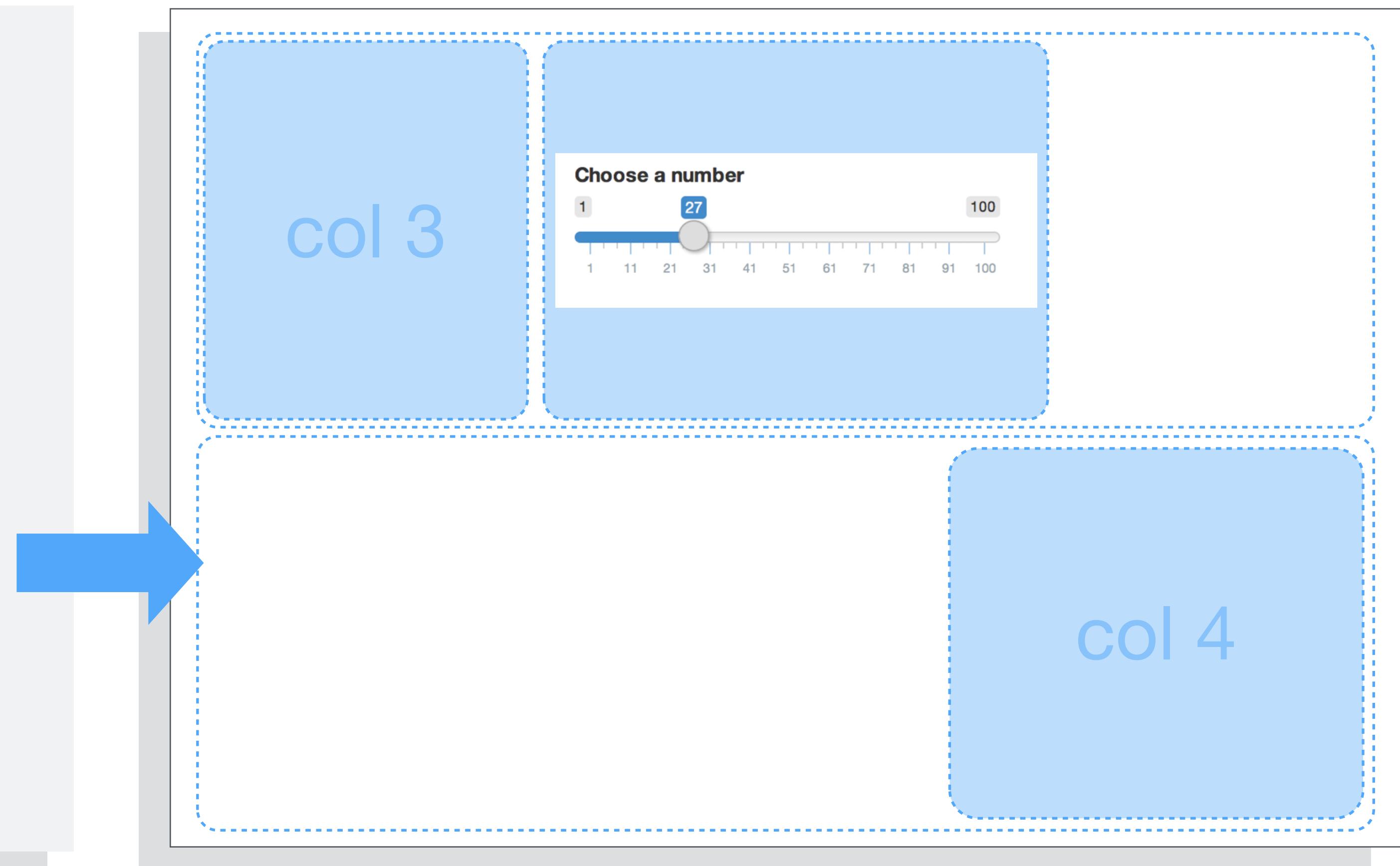
To place an element in the grid, call it as an argument of a layout function

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



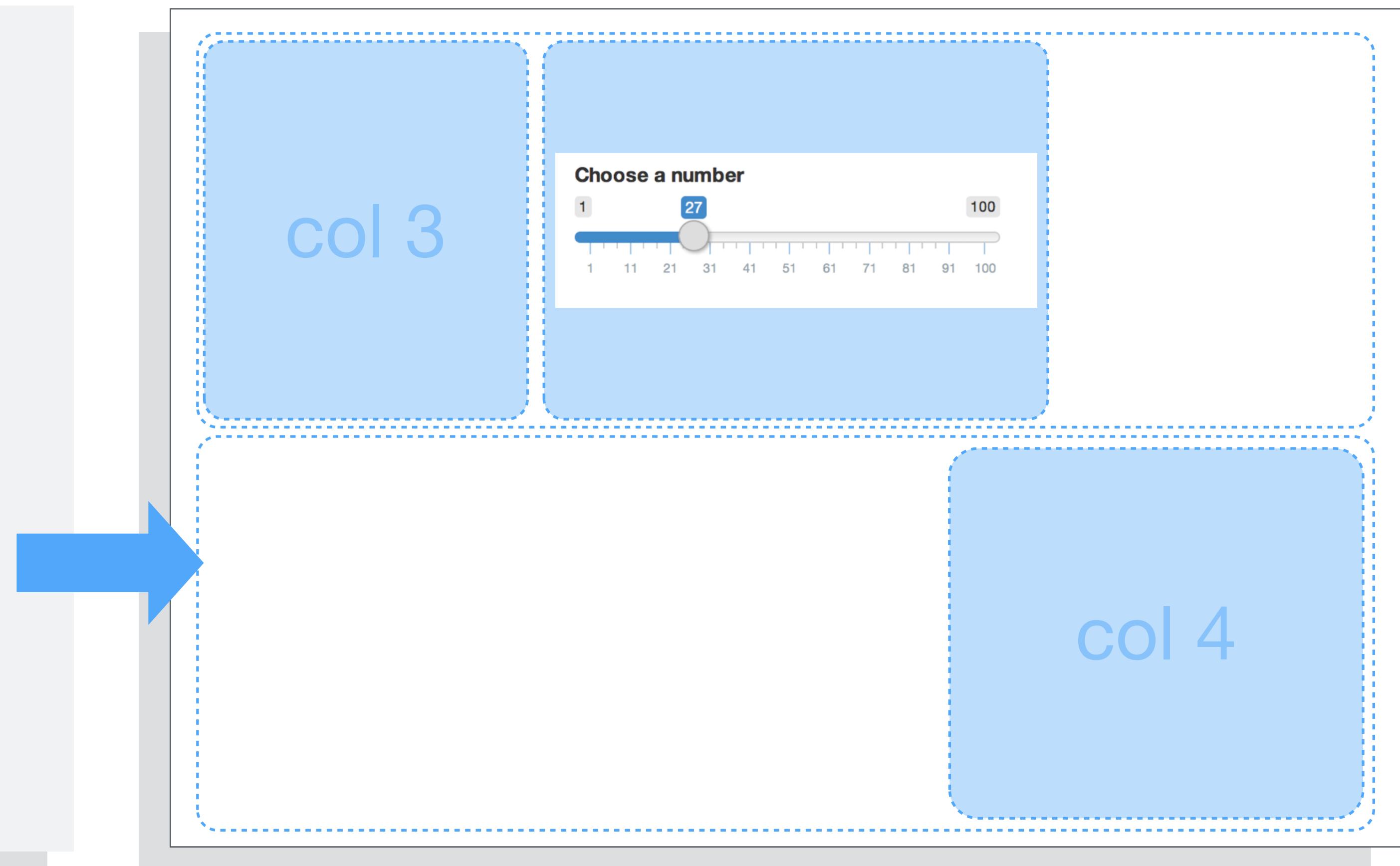
To place an element in the grid, call it as an argument of a layout function

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



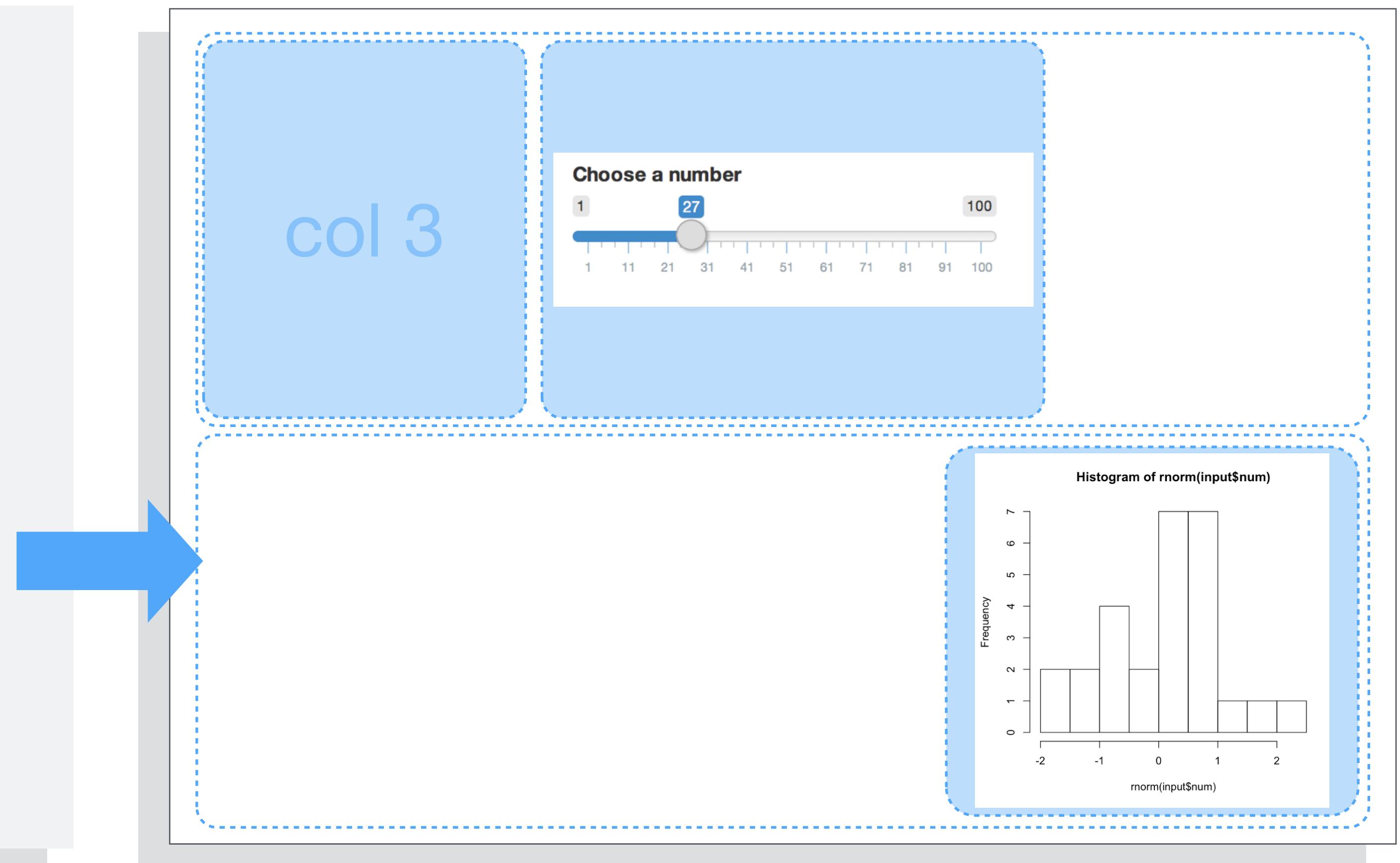
To place an element in the grid, call it as an argument of a layout function

```
fluidPage(  
  fluidRow(  
    column(3),  
    column(5, sliderInput(...))  
  ),  
  fluidRow(  
    column(4, offset = 8,  
      plotOutput("hist"))  
  )  
)  
)
```



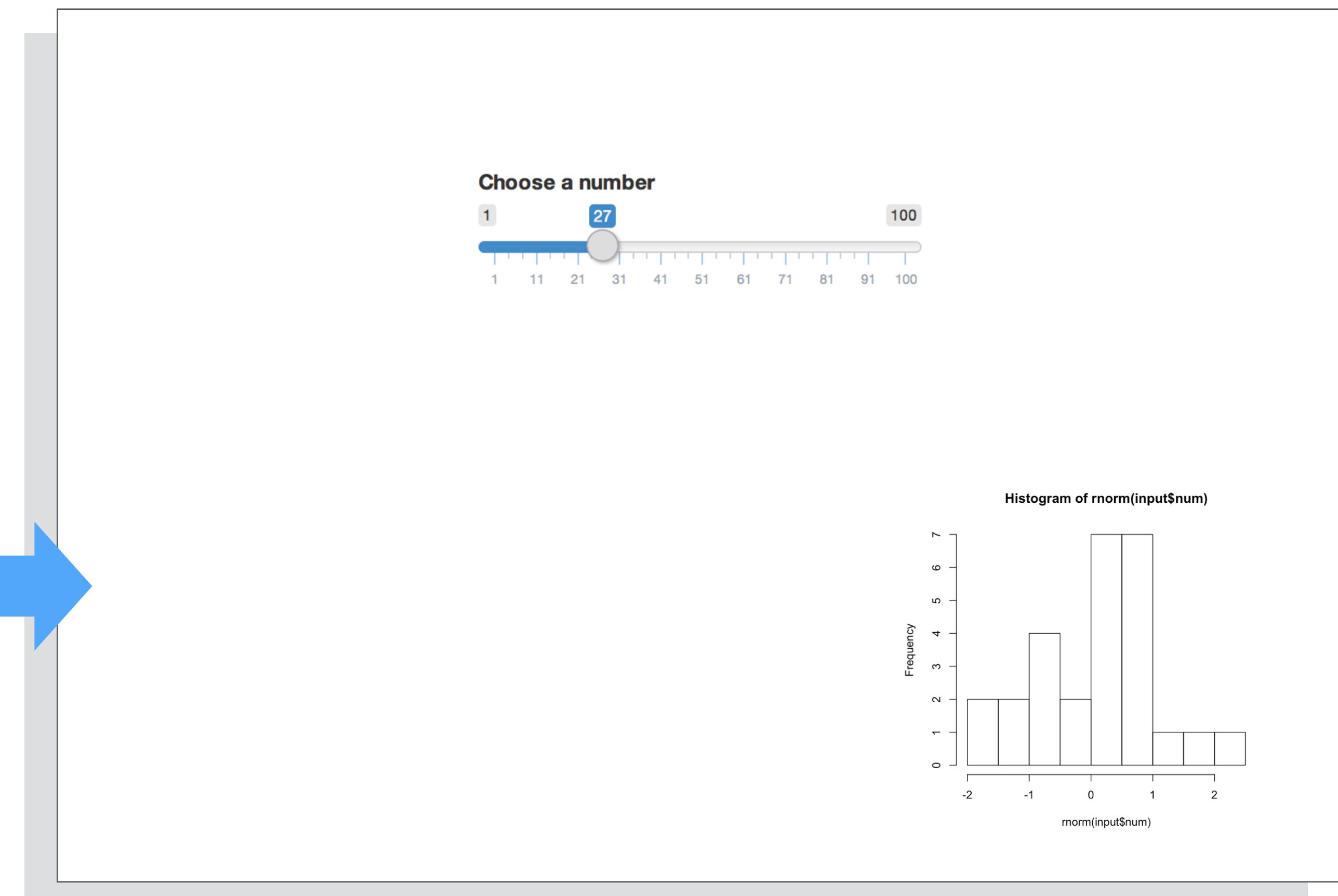
To place an element in the grid, call it as an argument of a layout function

```
fluidPage(  
  fluidRow(  
    column(3),  
    column(5, sliderInput(...))  
  ),  
  fluidRow(  
    column(4, offset = 8,  
      plotOutput("hist"))  
  )  
)
```

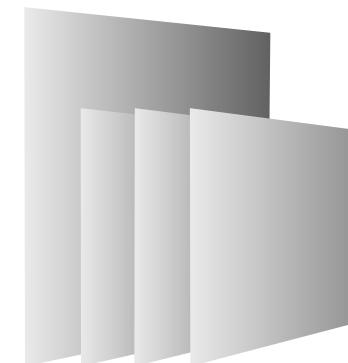


To place an element in the grid, call it as an argument of a layout function

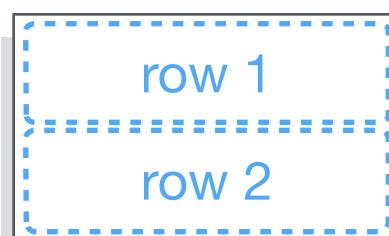
```
fluidPage(  
  fluidRow(  
    column(3),  
    column(5, sliderInput(...))  
  ),  
  fluidRow(  
    column(4, offset = 8,  
          plotOutput("hist"))  
  )  
)  
)
```



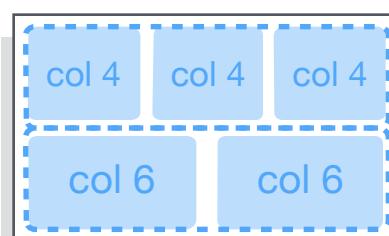
Recap: Layout functions



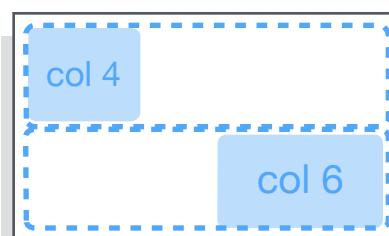
Position elements in a grid, or stack them in layers.



Use **fluidRow()** to arrange elements in rows



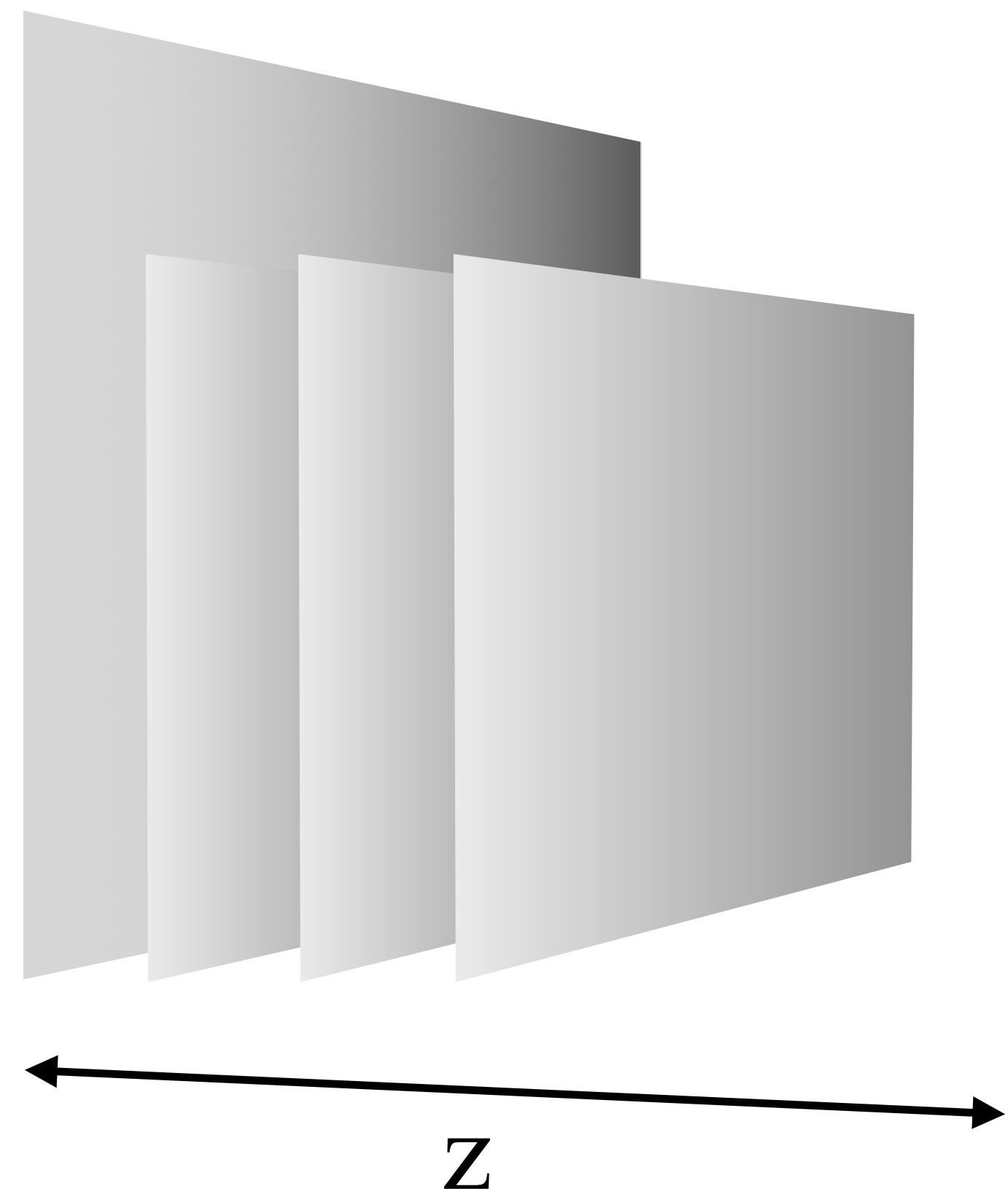
Use **column()** to arrange elements in columns



Column takes **width** and **offset** arguments

**Assemble
layers
of panels**

Use layout functions to position elements within your app



Panels

Panels to group multiple elements into a single unit with its own properties.

Action button

Action

Current Value:

```
[1] 0  
attr(,"class")  
[1] "integer"  
"shinyActionButtonValue"
```

See Code

The Shiny Widgets Gallery interface is shown on the right, featuring a header with 'Shiny by RStudio' and 'Shiny Widgets Gallery'. Below the header, a note states: '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.' The gallery contains several examples of different input types, each with a 'Current Value:' box and a 'See Code' button. The 'Action button' example is highlighted with a blue arrow pointing from the large panel above.

Action button

Action

Current Value:

```
[1] 0  
attr(,"class")  
[1] "integer"  
"shinyActionButtonValue"
```

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:

Date range

2015-06-02 to 2015-06-02

Current Values:

File input

Choose File No file chosen

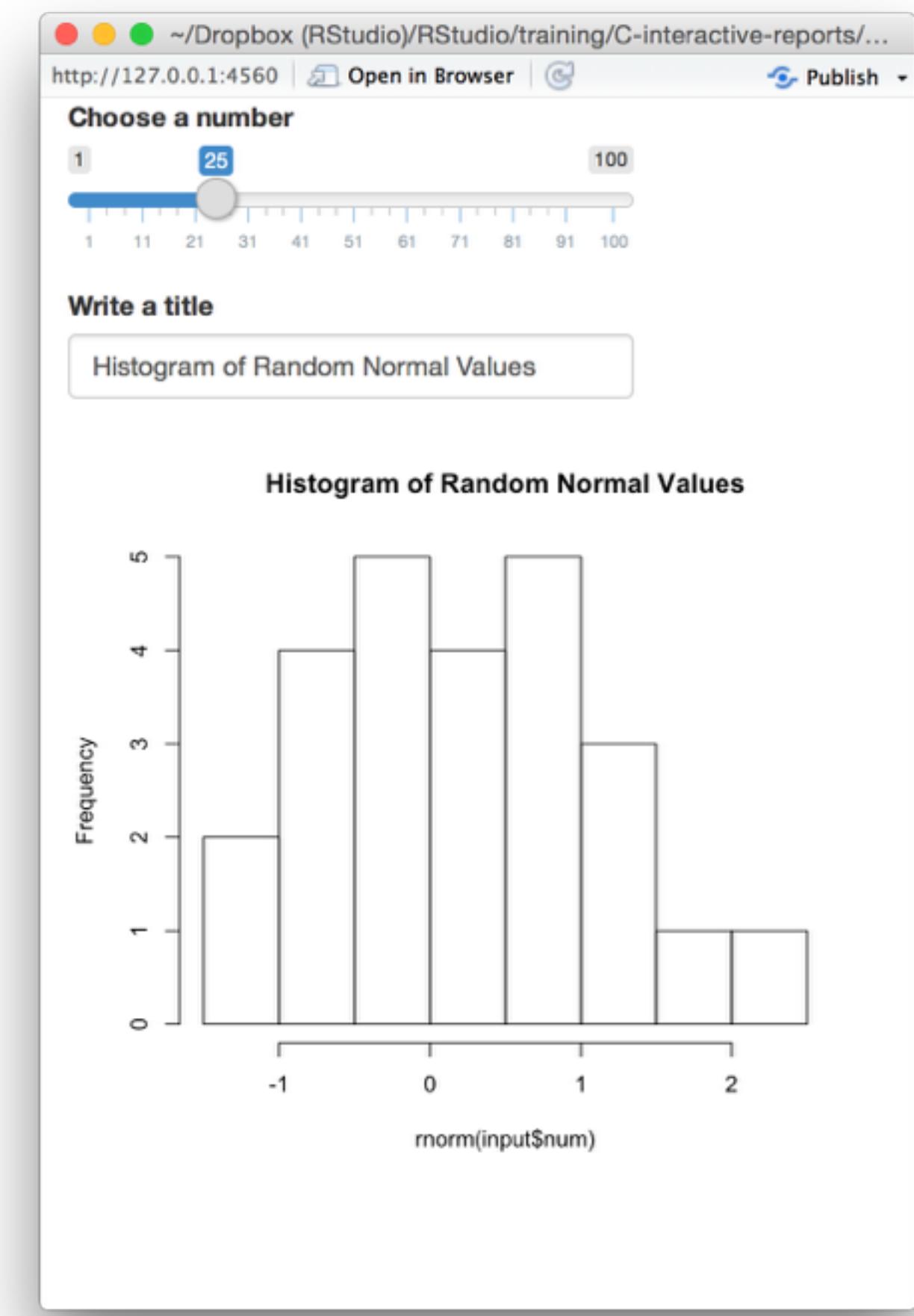
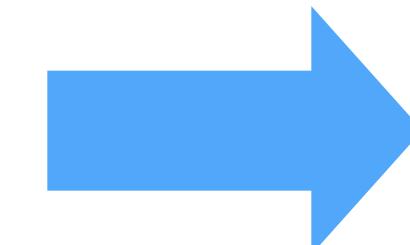
Current Value:

<http://shiny.rstudio.com/gallery/widget-gallery.html>

wellPanel()

Groups elements into a grey "well"

```
# 04-well.R  
  
ui <- fluidPage(  
  
  sliderInput("num", "Choose a number",  
    value = 25, min = 1, max = 100),  
  textInput("title", value = "Histogram",  
    label = "Write a title"),  
  
  plotOutput("hist")  
)
```

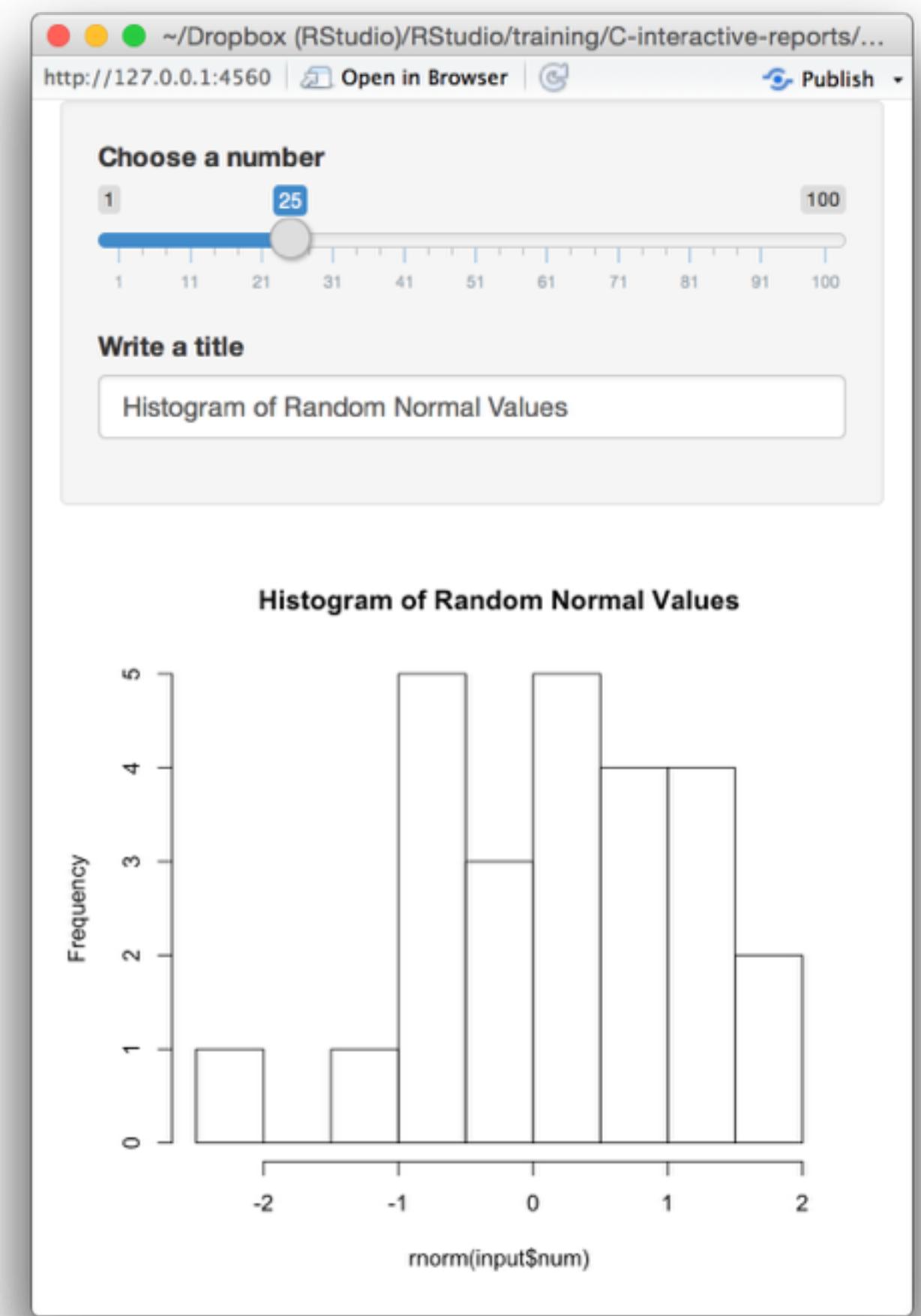
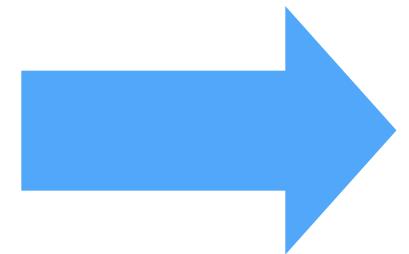


wellPanel()

Groups elements into a grey "well"

```
# 04-well.R

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



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.

tabPanel()

`tabPanel()` creates a stackable layer of elements.
Each tab is like a small UI of its own.

`tabPanel("Tab 1", ...)`

A title
(for navigation)

elements to
appear in the tab

Combine `tabPanel()`'s with one of:

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

tabsetPanel()

`tabsetPanel()` combines tabs into a single *panel*.
Use *tabs* to navigate between tabs.

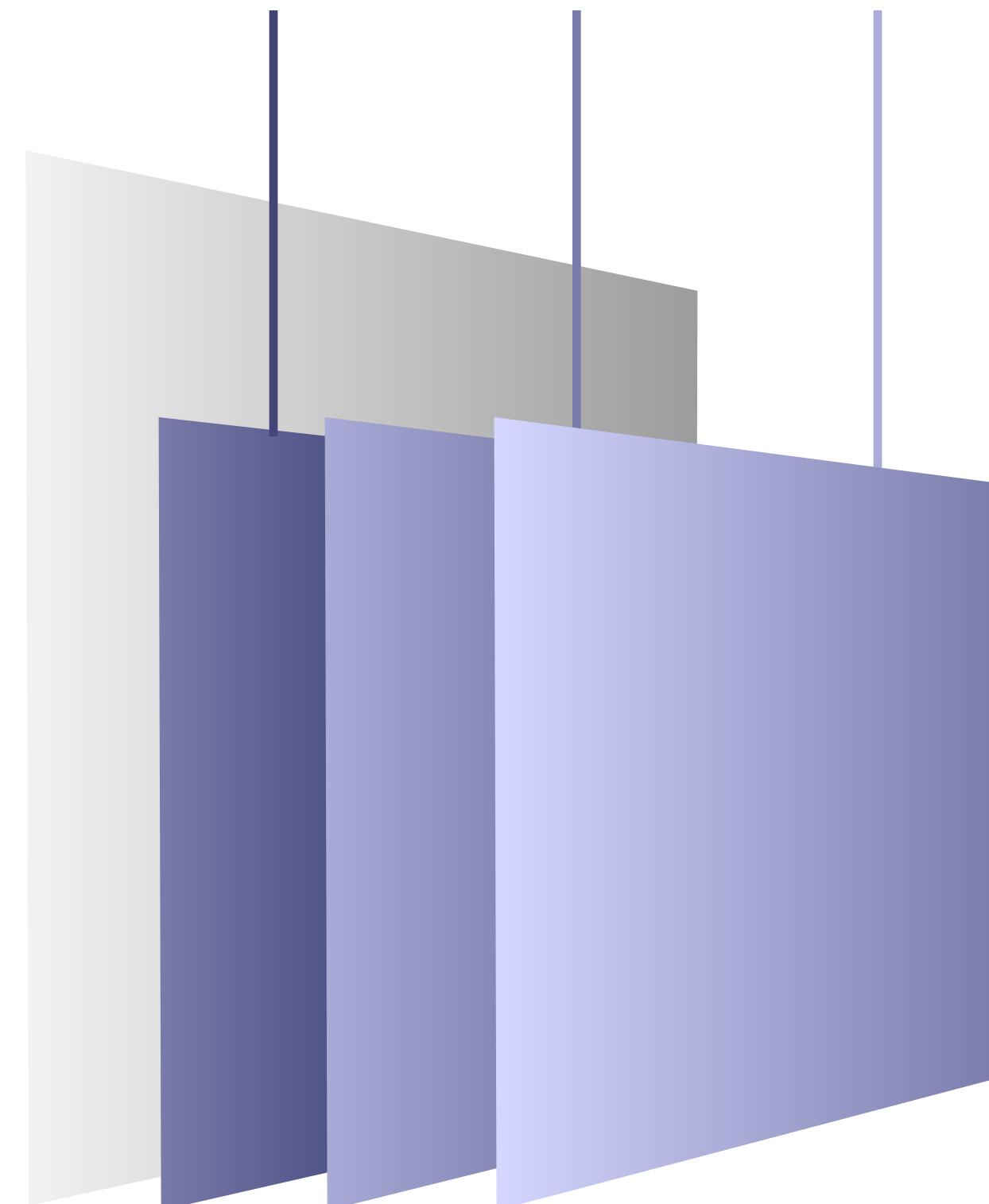
```
fluidPage(  
  tabsetPanel(  
    tabPanel("tab 1", "contents"),  
    tabPanel("tab 2", "contents"),  
    tabPanel("tab 3", "contents")  
  )  
)
```



Navigation

Content

tabPanel **tabPanel** **tabPanel**

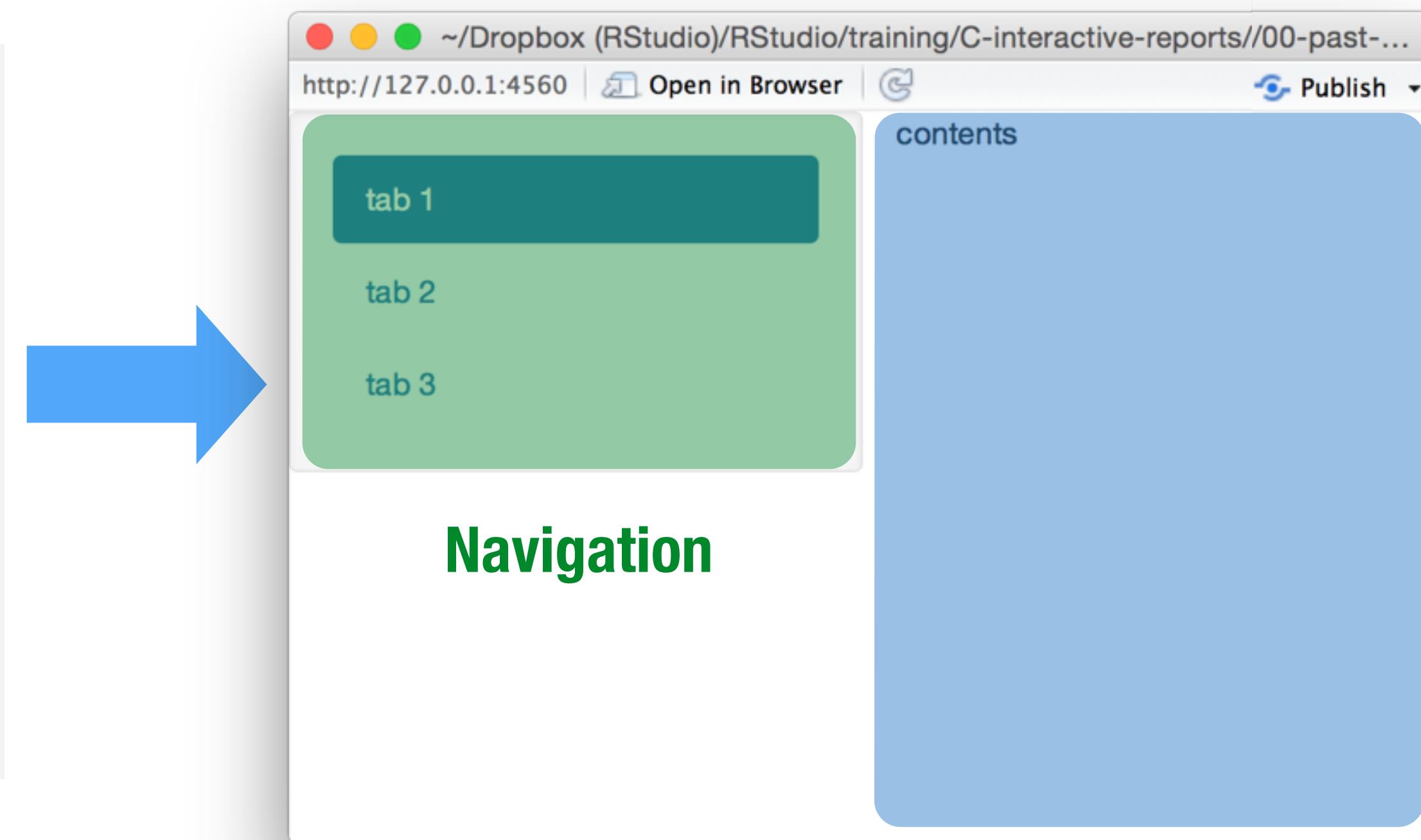


tabsetPanel

navlistPanel()

`navlistPanel()` combines tabs into a single *panel*.
Use *links* to navigate between tabs.

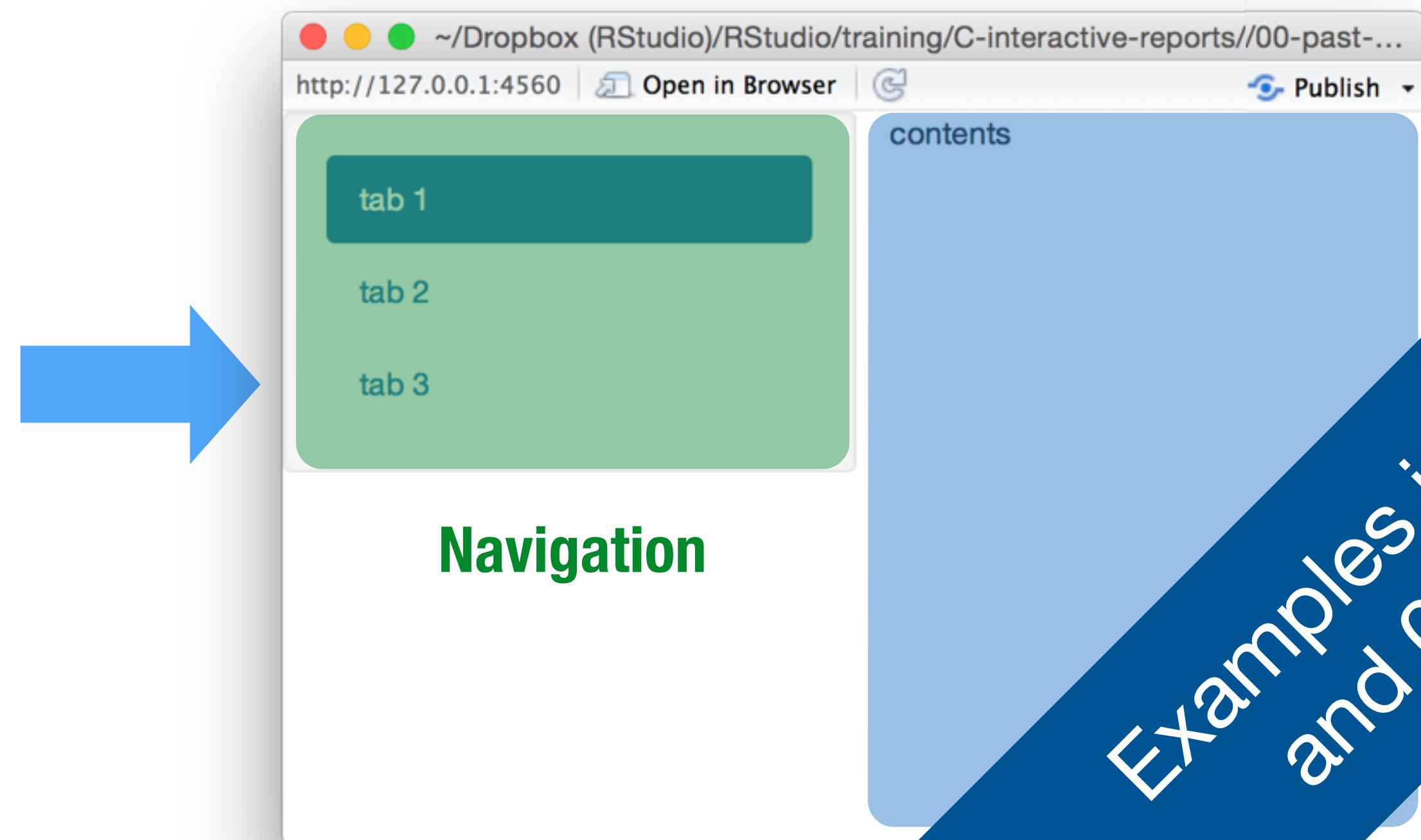
```
fluidPage(  
  navlistPanel(  
    tabPanel("tab 1", "contents"),  
    tabPanel("tab 2", "contents"),  
    tabPanel("tab 3", "contents")  
  )  
)
```



navlistPanel()

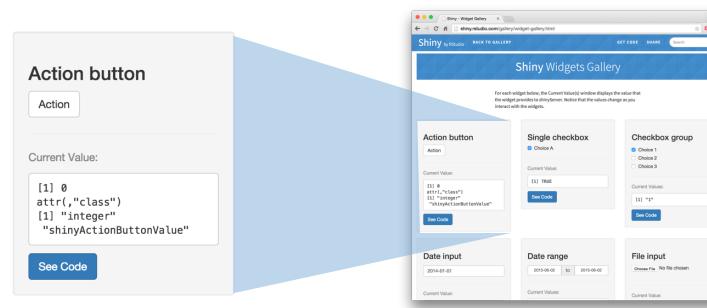
`navlistPanel()` combines tabs into a single *panel*.
Use *links* to navigate between tabs.

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

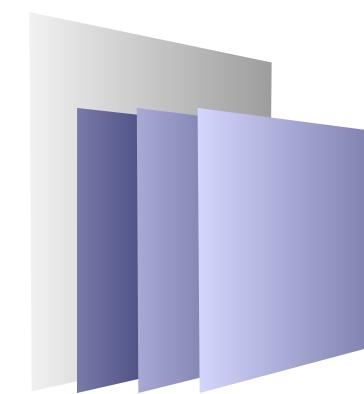


Examples in 05-tabs.R
and 06-navlist.R

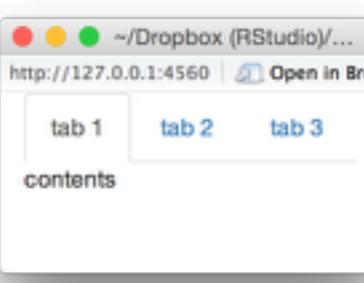
Recap: Panels



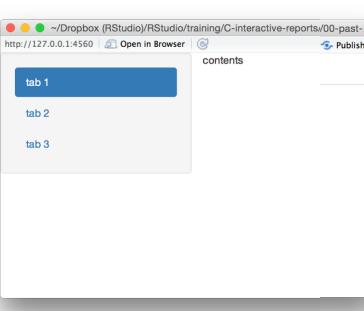
Panels group elements into a single unit for aesthetic or functional reasons



Use **tabPanel()** to create a stackable panel



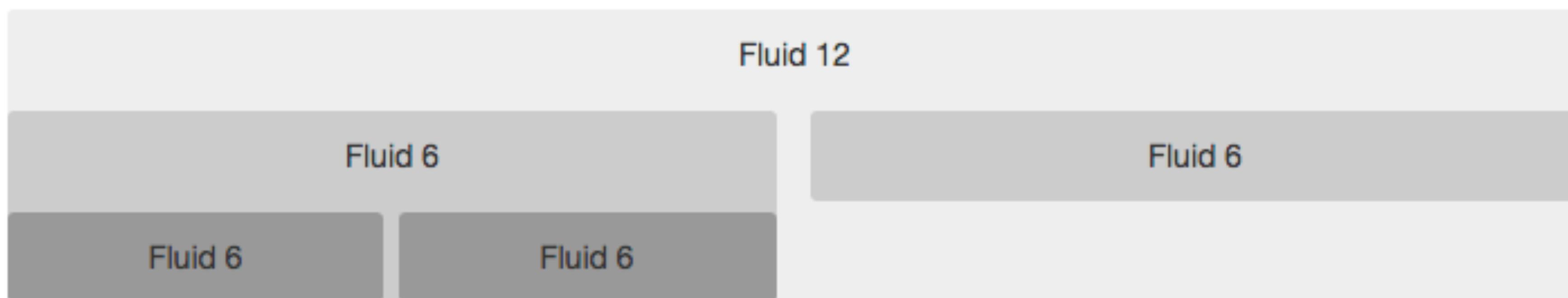
Use **tabsetPanel()** to arrange tab panels into a stack with tab navigation



Use **navlistPanel()** to arrange tab panels into a stack with sidebar navigation

The Shiny Layout Guide

<http://shiny.rstudio.com/articles/layout-guide.html>



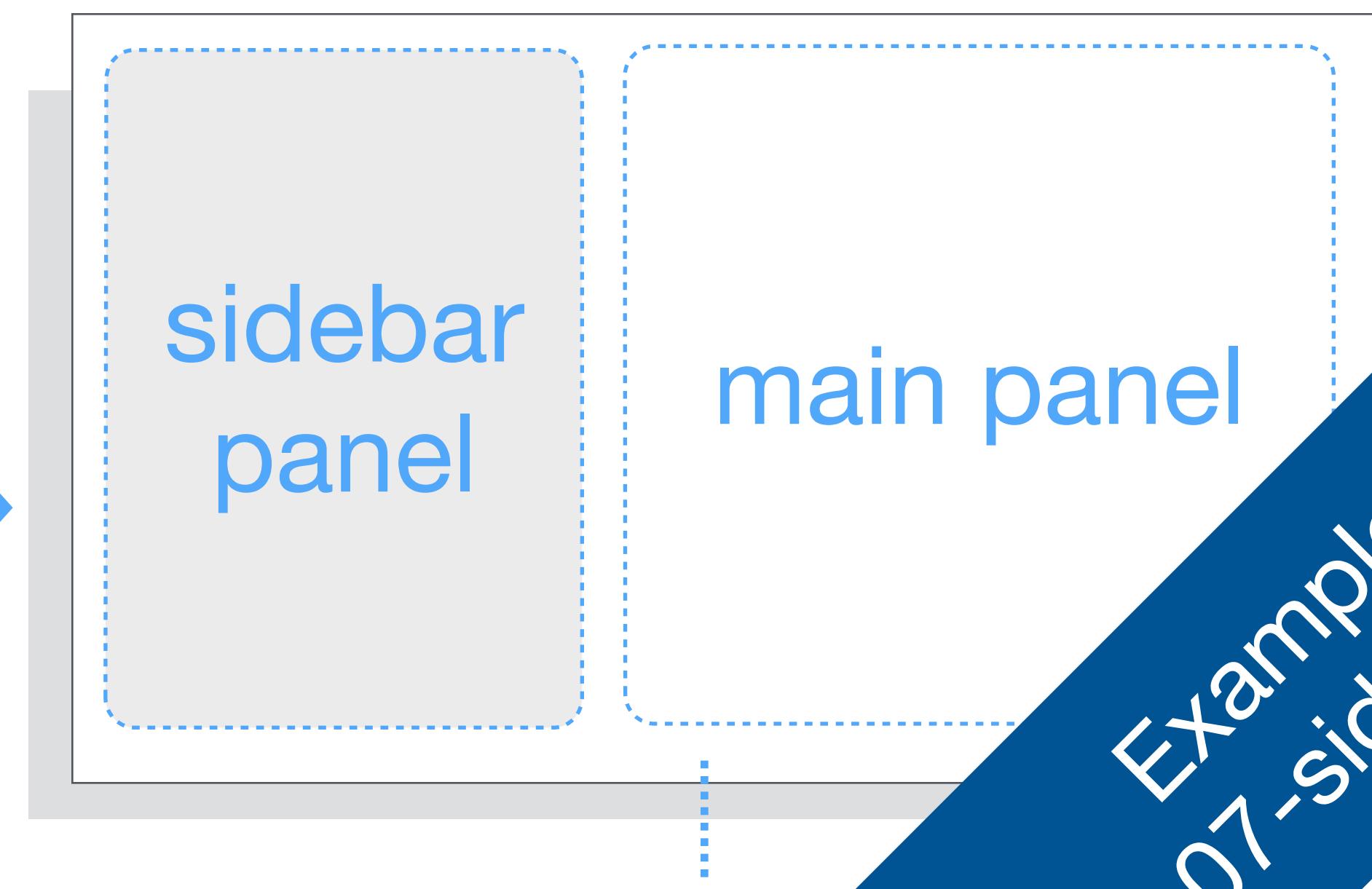
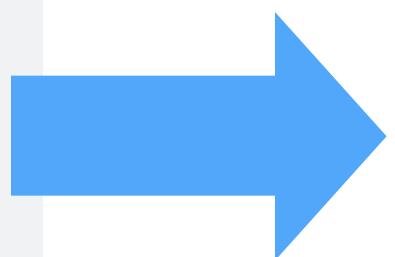
You can build sophisticated, customized layouts with Shiny's grid system.

Use a
prepackaged
layout

sidebarLayout()

Use with `sidebarPanel()` and `mainPanel()` to divide app into two sections.

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(),  
    mainPanel()  
)  
)
```



sidebar la

Example in
07-sidebar.R



fixedPage()

Creates a page that defaults to a width of 724, 940, or 1170 pixels (depending on browser window)

```
ui <- fixedPage(  
  fixedRow(  
    column(5, # etc.)  
  )  
)
```

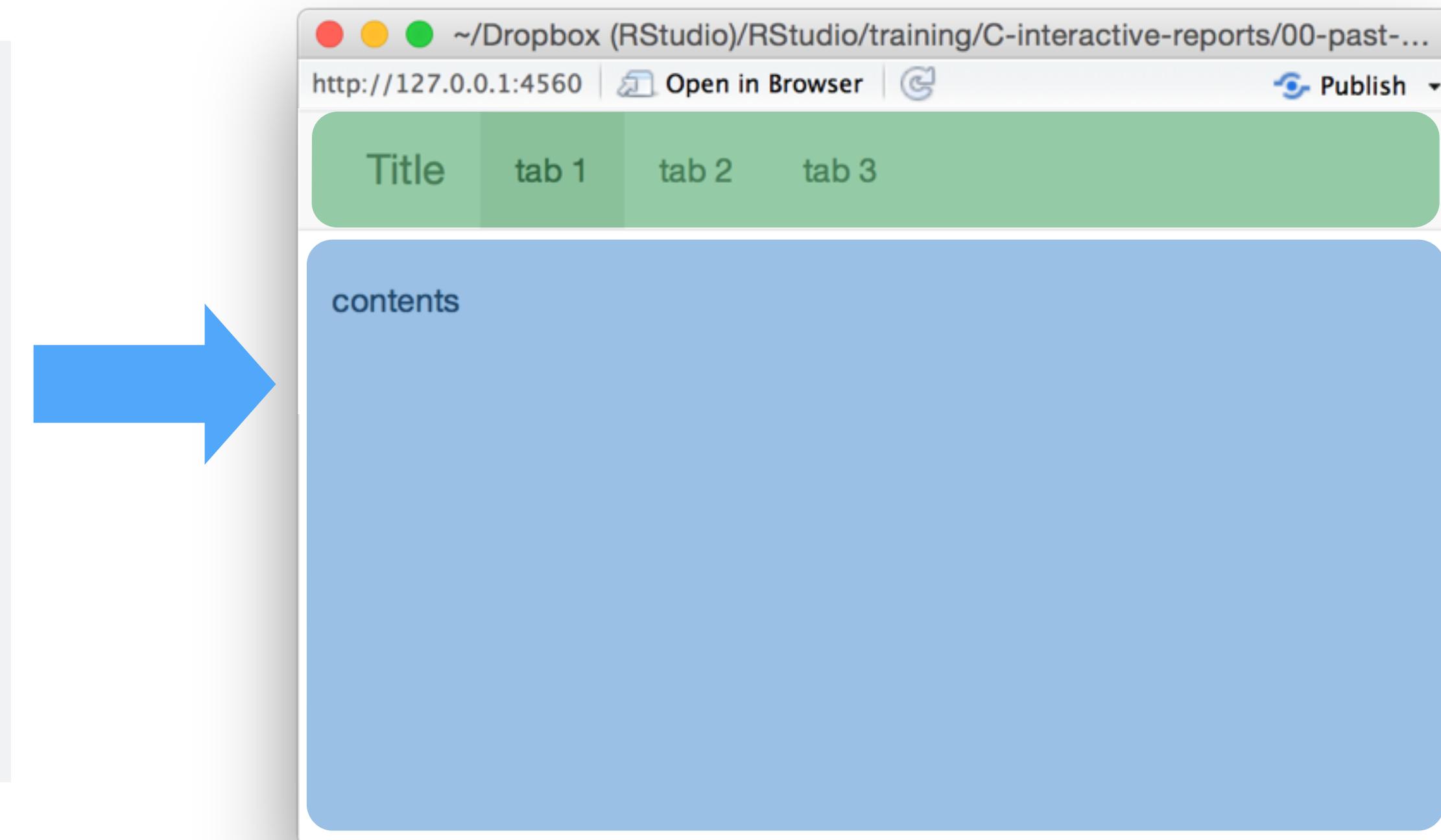
Use with **fixedRow()**

Compare to `fluidPage()` and `fluidRow()` which adjust to browser window

navbarPage()

`navbarPage()` combines tabs into a single *page*.
navbarPage() replaces *fluidPage()*. Requires *title*.

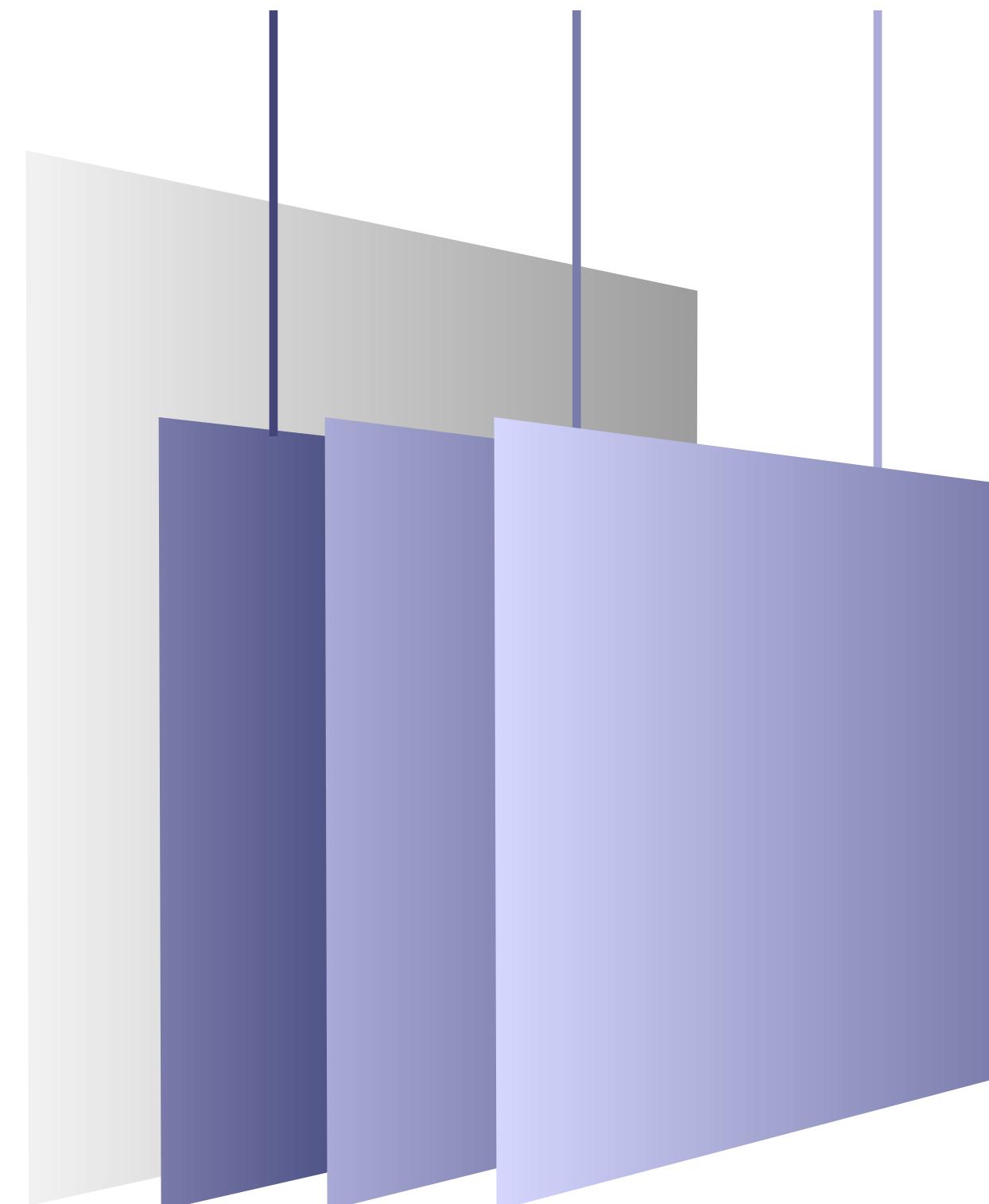
```
navbarPage(title = "Title",  
           tabPanel("tab 1", "contents"),  
           tabPanel("tab 2", "contents"),  
           tabPanel("tab 3", "contents"))
```



Navigation

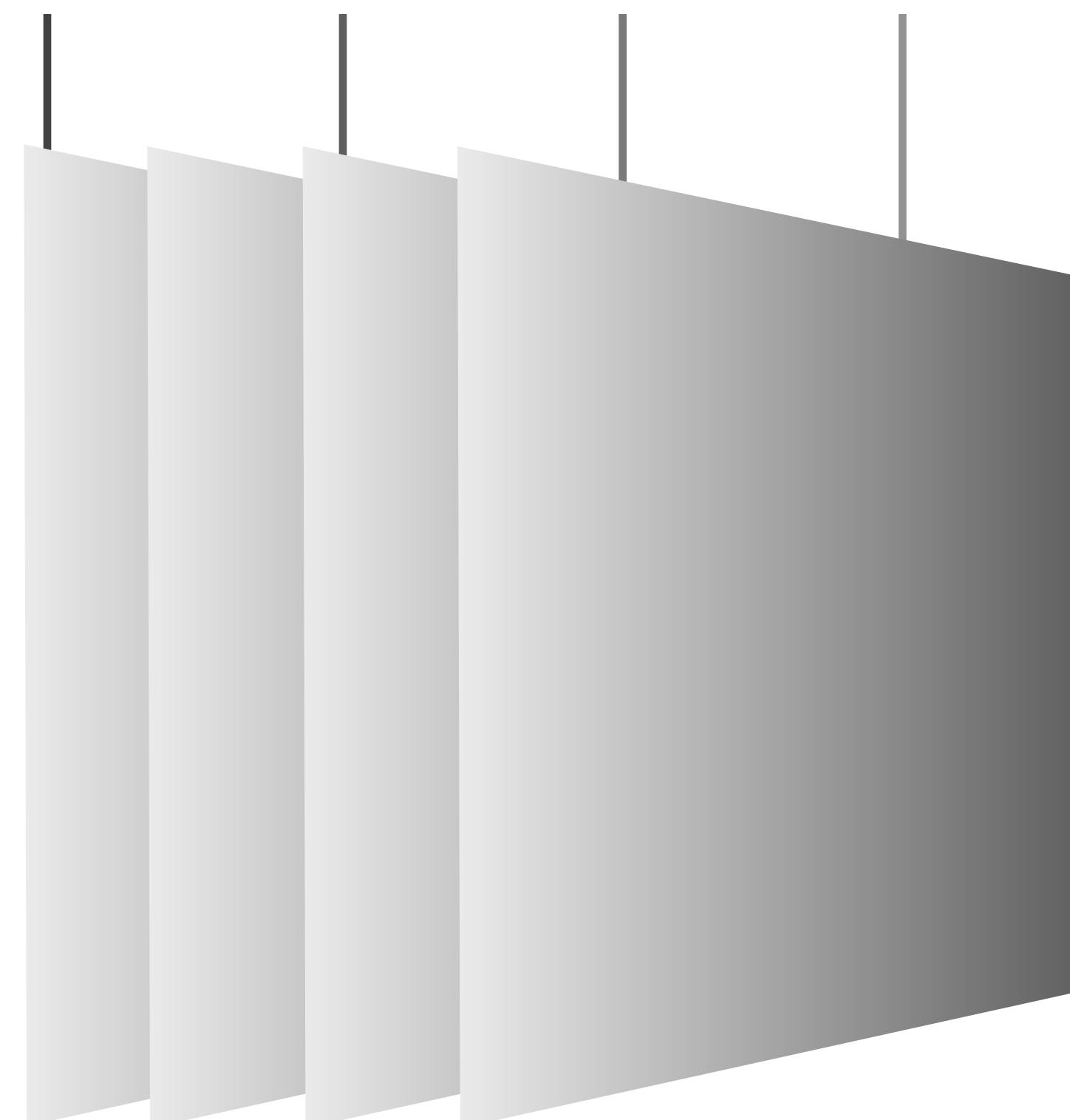
Content

tabPanel **tabPanel** **tabPanel**



tabsetPanel

tabPanel **tabPanel** **tabPanel** **tabPanel**

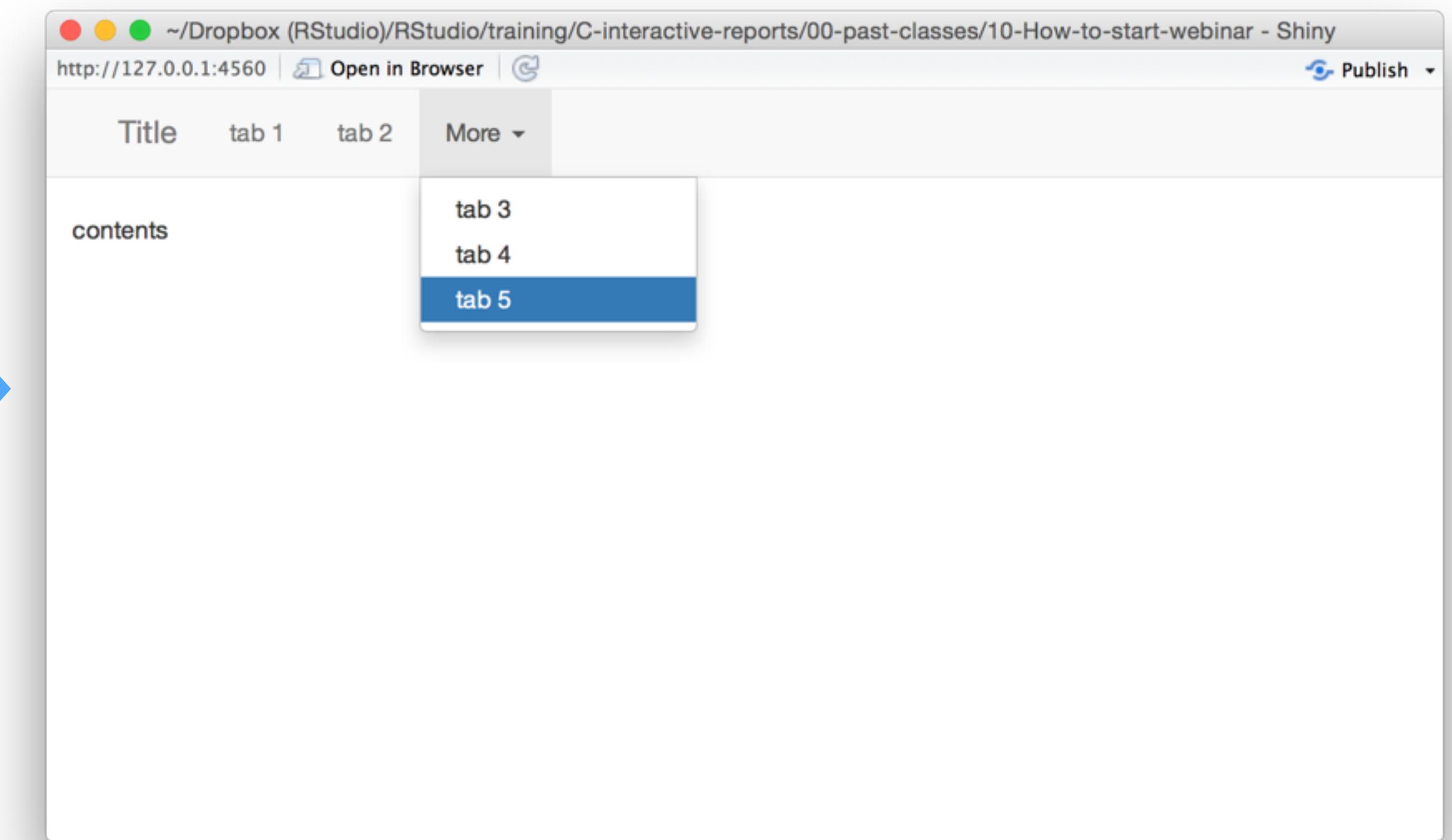
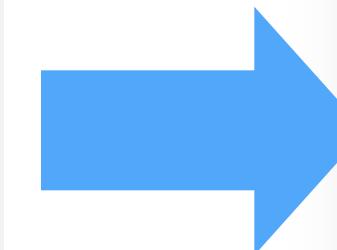


navbarPage

navbarMenu()

`navbarMenu()` combines tab links into a dropdown menu for `navbarPage()`

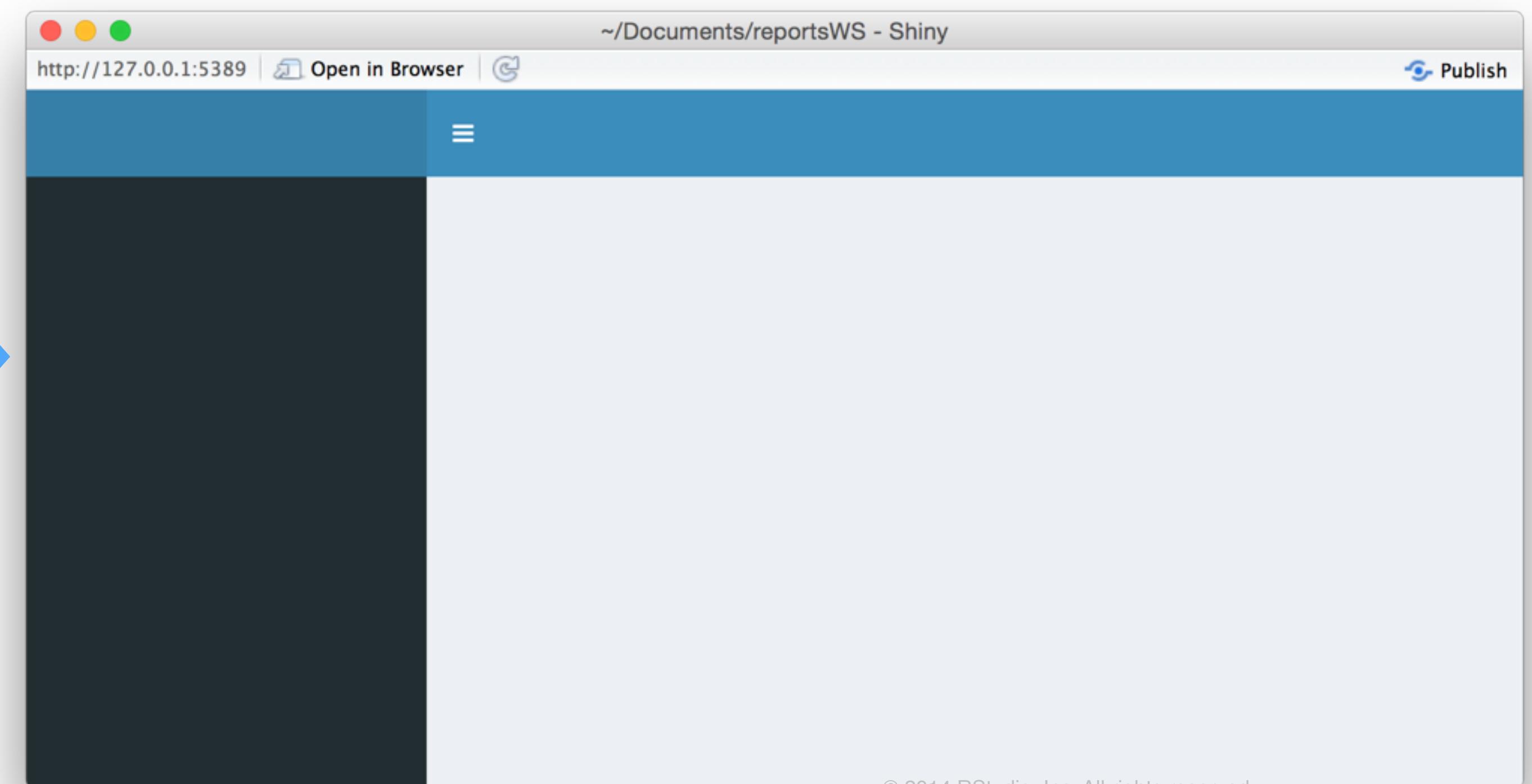
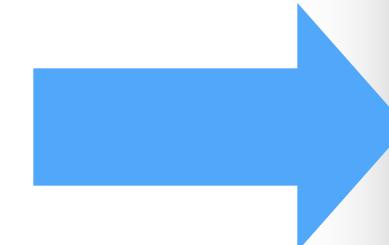
```
navbarPage(title = "Title",
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  navbarMenu(title = "More",
    tabPanel("tab 3", "contents"),
    tabPanel("tab 4", "contents"),
    tabPanel("tab 5", "contents"))
)
```



dashboardPage()

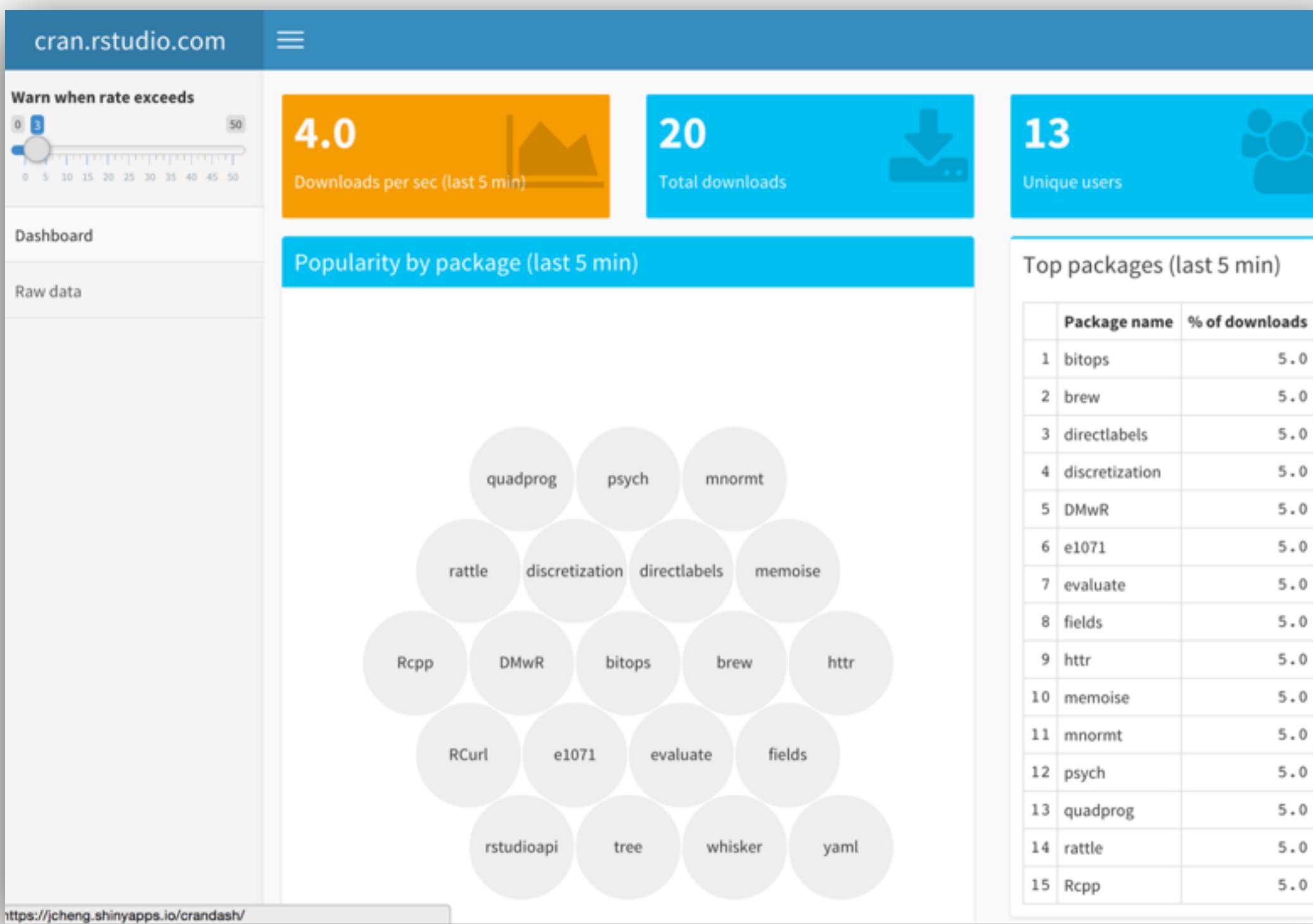
`dashboardPage()` comes in the shinydashboard package

```
library(shinydashboard)  
ui <- dashboardPage(  
  dashboardHeader(),  
  dashboardSidebar(),  
  dashboardBody())  
)
```



shinydashboard

<http://rstudio.github.io/shinydashboard/>

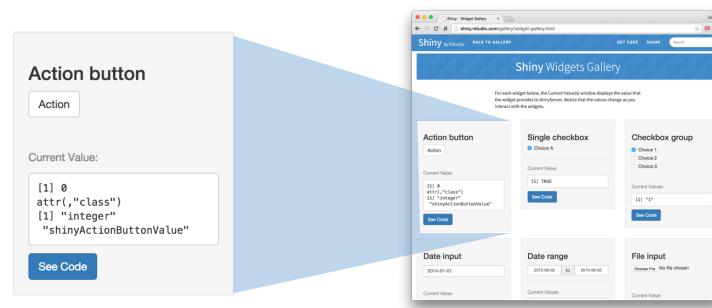


A package of layout functions for building administrative dashboards with Shiny

Dynamic Dashboards with Shiny Webinar:

www.rstudio.com/resources/webinars/

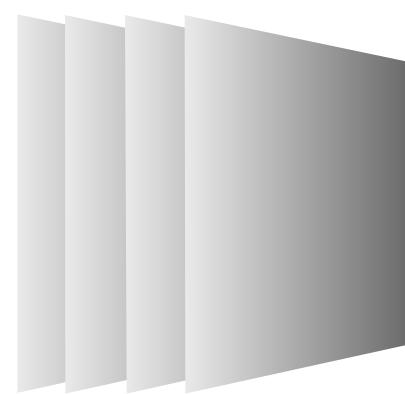
Recap: Prepackaged Layouts



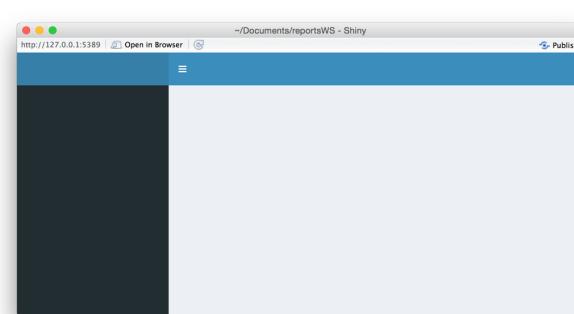
Use **sidebarLayout()** with **sidebarPanel()** and **mainPanel()** to quickly create a sidebar design.



Use **fixedPanel()** with **fixedRow()** to create a fixed (non-fluid) design



Use **navbarPage()** with **navbarMenu()** to create "multipage" app

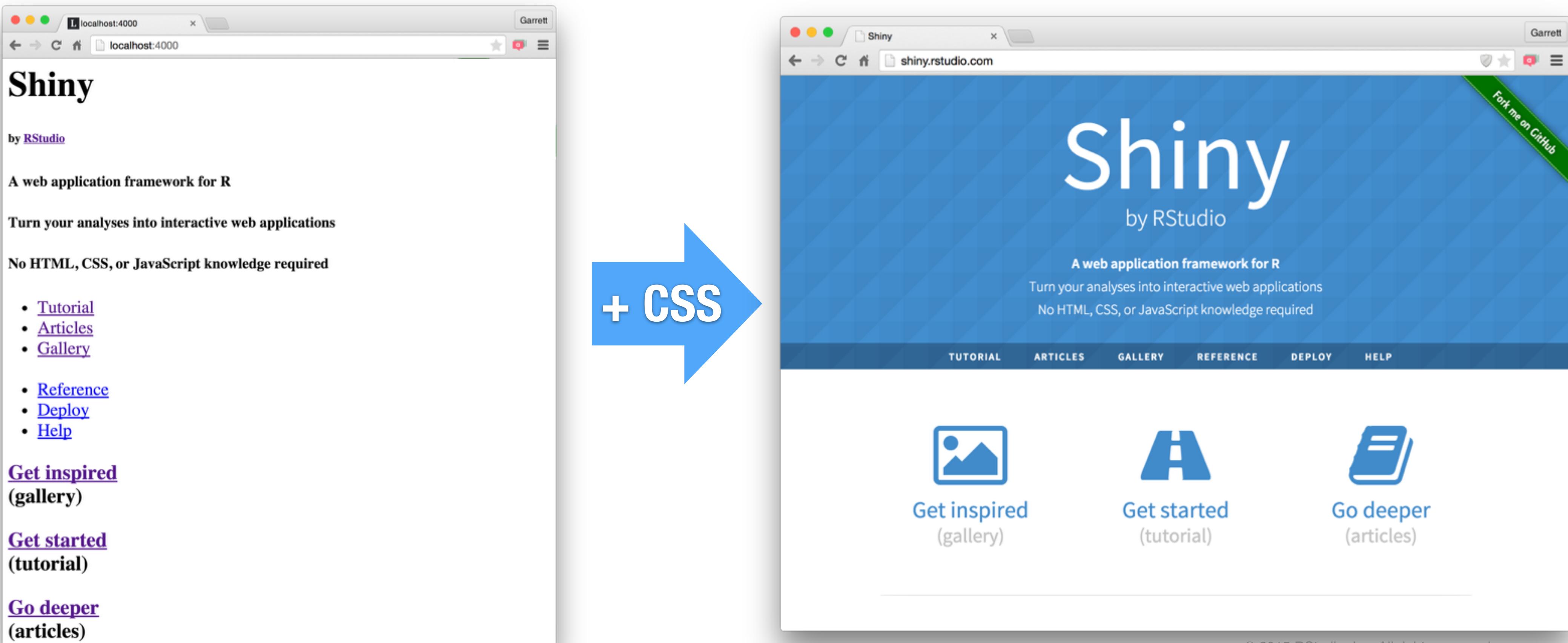


Use the **shinyDashboard** package to create dashboard layouts

**Style with
css**

What is CSS?

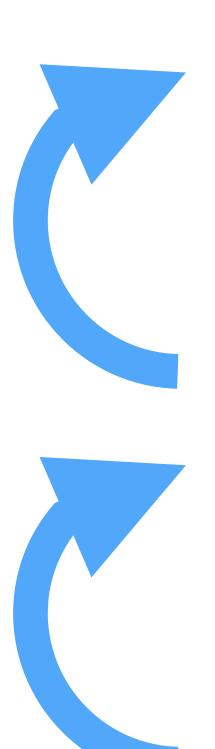
Cascading Style Sheets (CSS) are a framework for customizing the appearance of elements in a web page.



Style a web page in three ways:

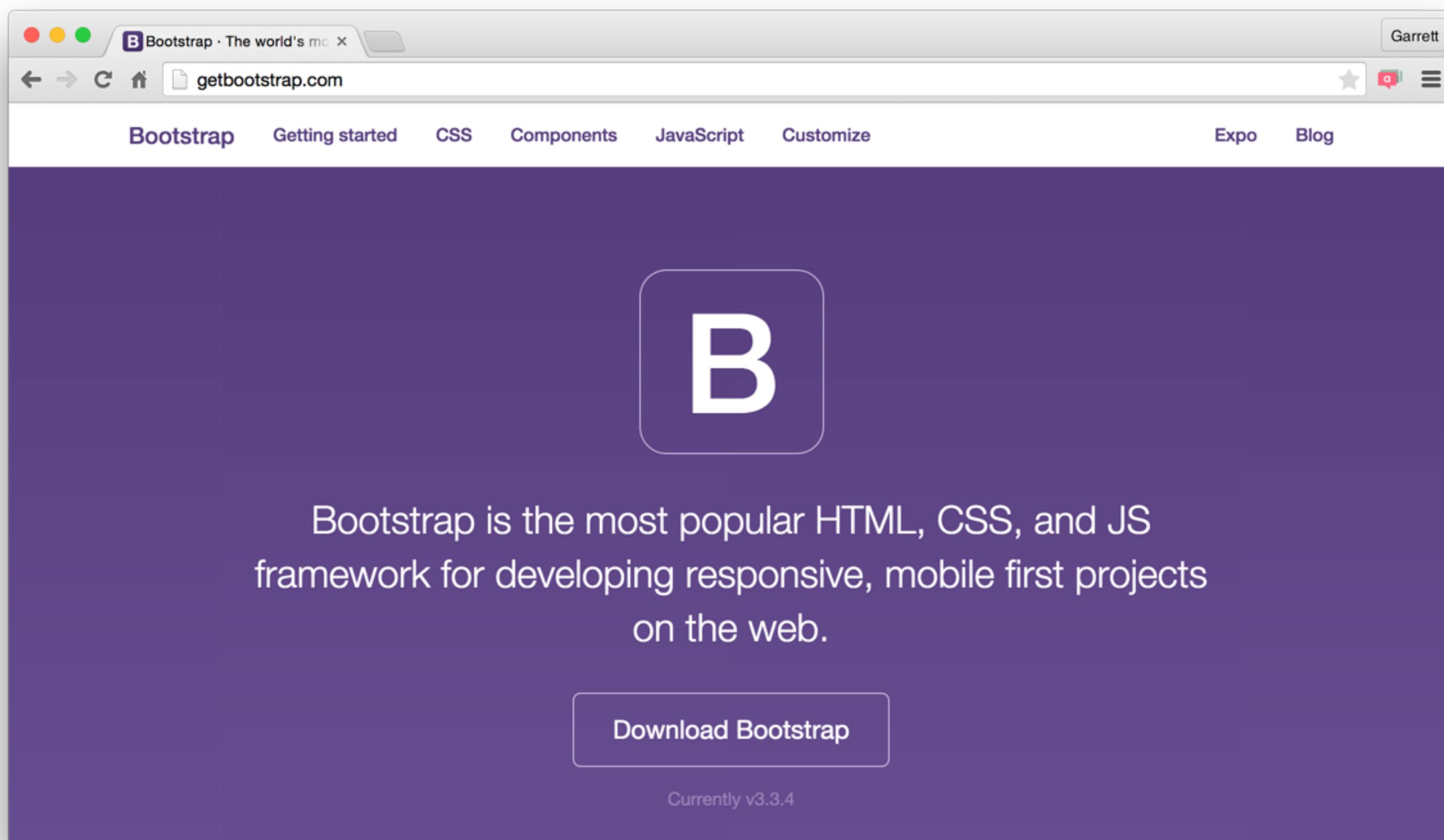
- 
- 1** Link to an external CSS file
 - 2** Write global CSS in header
 - 3** Write individual CSS in a tag's style attribute
- Overrides**

Match styling to:

- 
- 1** Tag
 - 2** Class
 - 3** id
- Overrides**

Bootstrap

Shiny uses the Bootstrap 3 CSS framework, getbootstrap.com



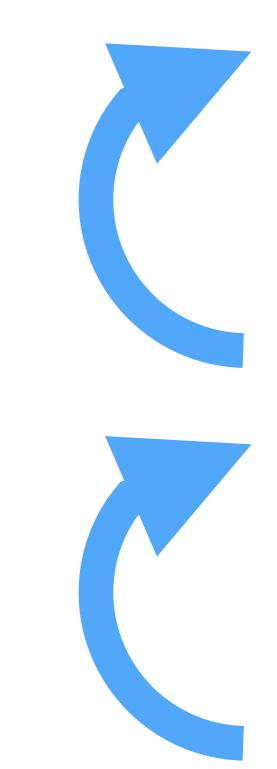
Bootstrap

Shiny uses the Bootstrap 3 CSS framework, getbootstrap.com

```
fluidPage()
```

```
<div class="container-fluid"></div>
```

Style a **Shiny app** in three ways:



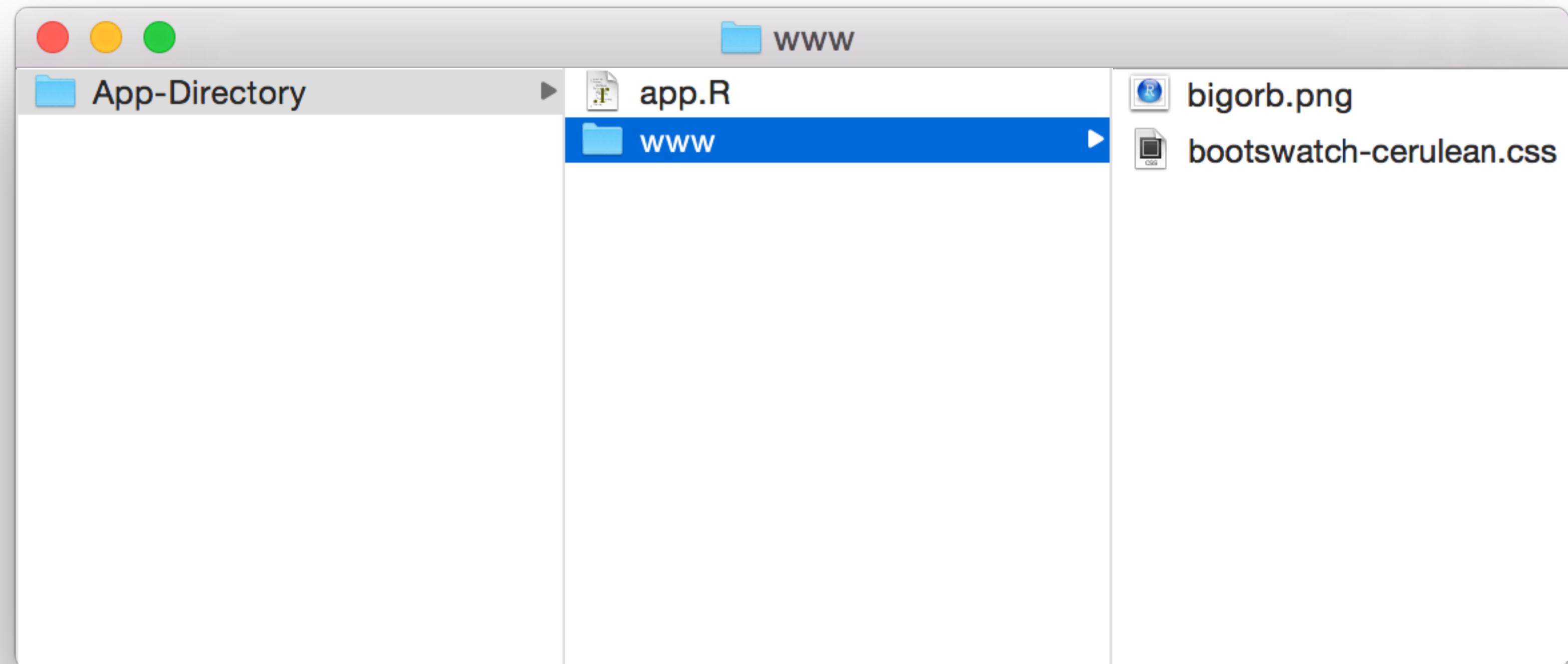
- 1** Link to an external CSS file
- 2** Write global CSS in header
- 3** Write individual CSS in a tag's style attribute

Overrides

*CSS designed to work with Bootstrap 3 will work best with Shiny.

1 Link to an external CSS file

Place .css files in the **www** folder of your app directory



Shiny will share a file with your user's browser if the file appears in www. Shiny will not share files that you do not place in www.

1 Link to an external CSS file

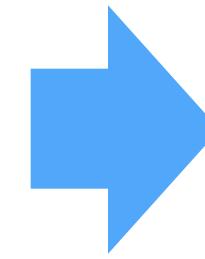
Set the theme argument of fluidPage() to the .css filename,
or...

```
ui <- fluidPage(  
  theme = "bootswatch-cerulean.css",  
  sidebarLayout(  
    sidebarPanel(),  
    mainPanel()  
  )  
)
```

1 Link to an external CSS file

Or place a link in the app's header with to the file with
tags\$head() and **tags\$link()**

```
ui <- fluidPage(  
  tags$head(  
    ))  
)
```

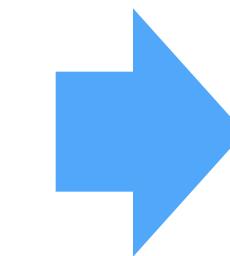


```
<head>  
  </head>  
<body>  
  <div class="container-fluid">  
  </div>  
</body>
```

1 Link to an external CSS file

Or place a link in the app's header with to the file with
tags\$head() and **tags\$link()**

```
ui <- fluidPage(  
  tags$head(  
    tags$link(  
      rel = "stylesheet",  
      type = "text/css",  
      href = "file.css"  
    )  
  )  
)
```

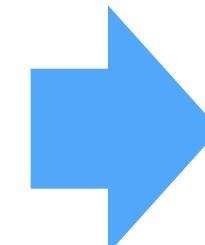


```
<head>  
  <link type="text/css" rel="stylesheet" href="file.css"/>  
</head>  
<body>  
  <div class="container-fluid">  
  </div>  
</body>
```

2 Write global CSS in header

Write global CSS with **tags\$head()** and **tags\$style()** and **HTML()**

```
ui <- fluidPage(  
  tags$head(  
    tags$style(HTML("'  
      p {  
        color:red;  
      }  
    "'))  
  )  
)
```

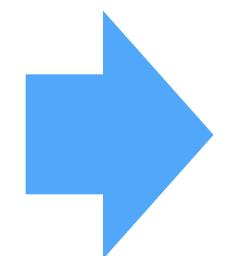


```
<head>  
  <style>  
    p {  
      color:red;  
    }  
  </style>  
</head>  
  
<body>  
  <div class="container-fluid">  
  </div>  
</body>
```

2 Write global CSS in header

Or save the CSS as a file in your app directory and include it with **includeCSS()**

```
ui <- fluidPage(  
  includeCSS("file.css")  
)
```



```
<head>  
  <style>  
    p {  
      color:red;  
    }  
  </style>  
</head>  
  
<body>  
  <div class="container-fluid">  
  </div>  
</body>
```

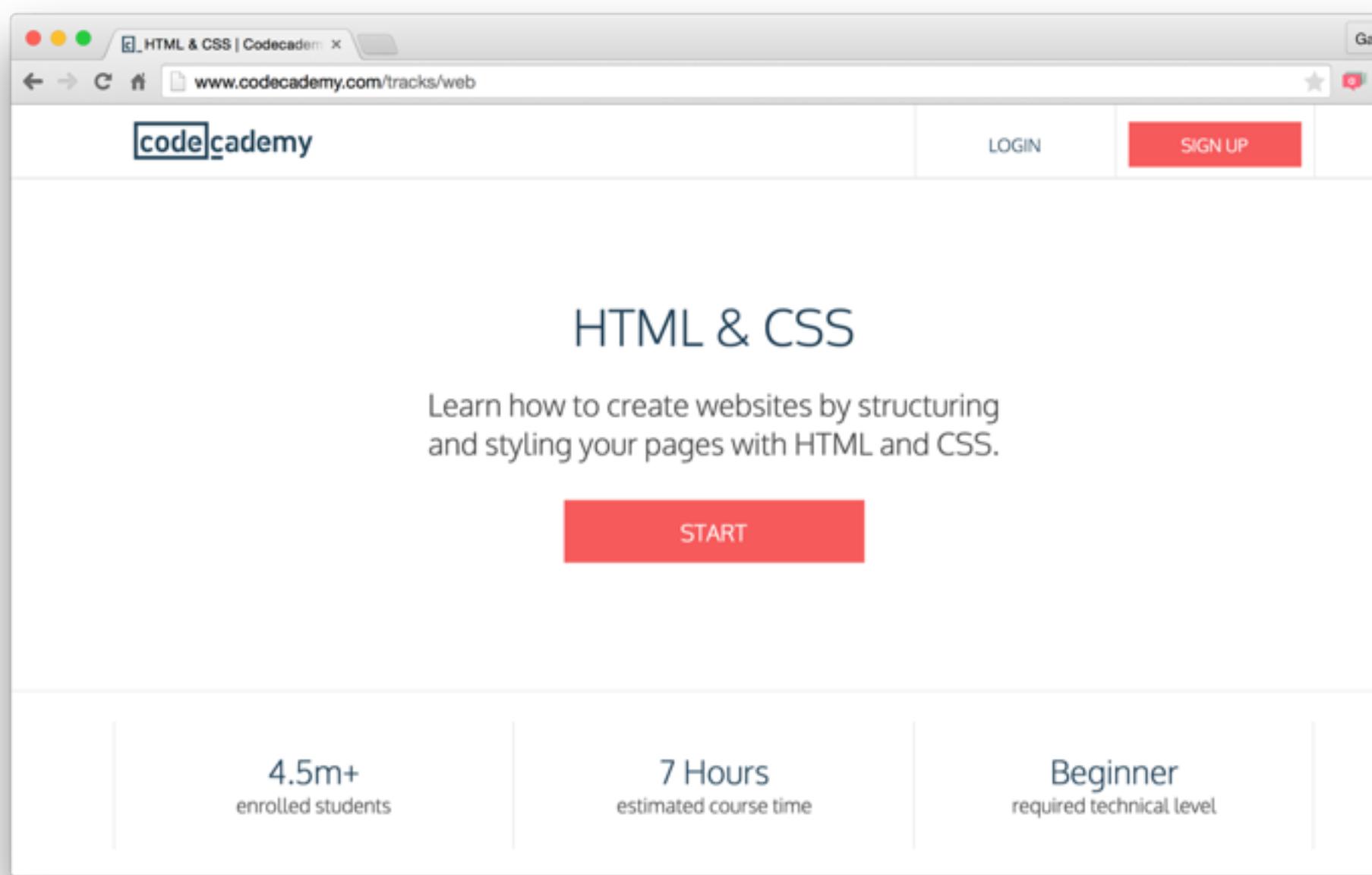
3 Write individual CSS in a tag's style attribute

Set the `style` argument in Shiny's tag functions

```
ui <- fluidPage(  
  tags$h1("Title", style = "color:red;")  
)
```

To learn more about CSS & HTML

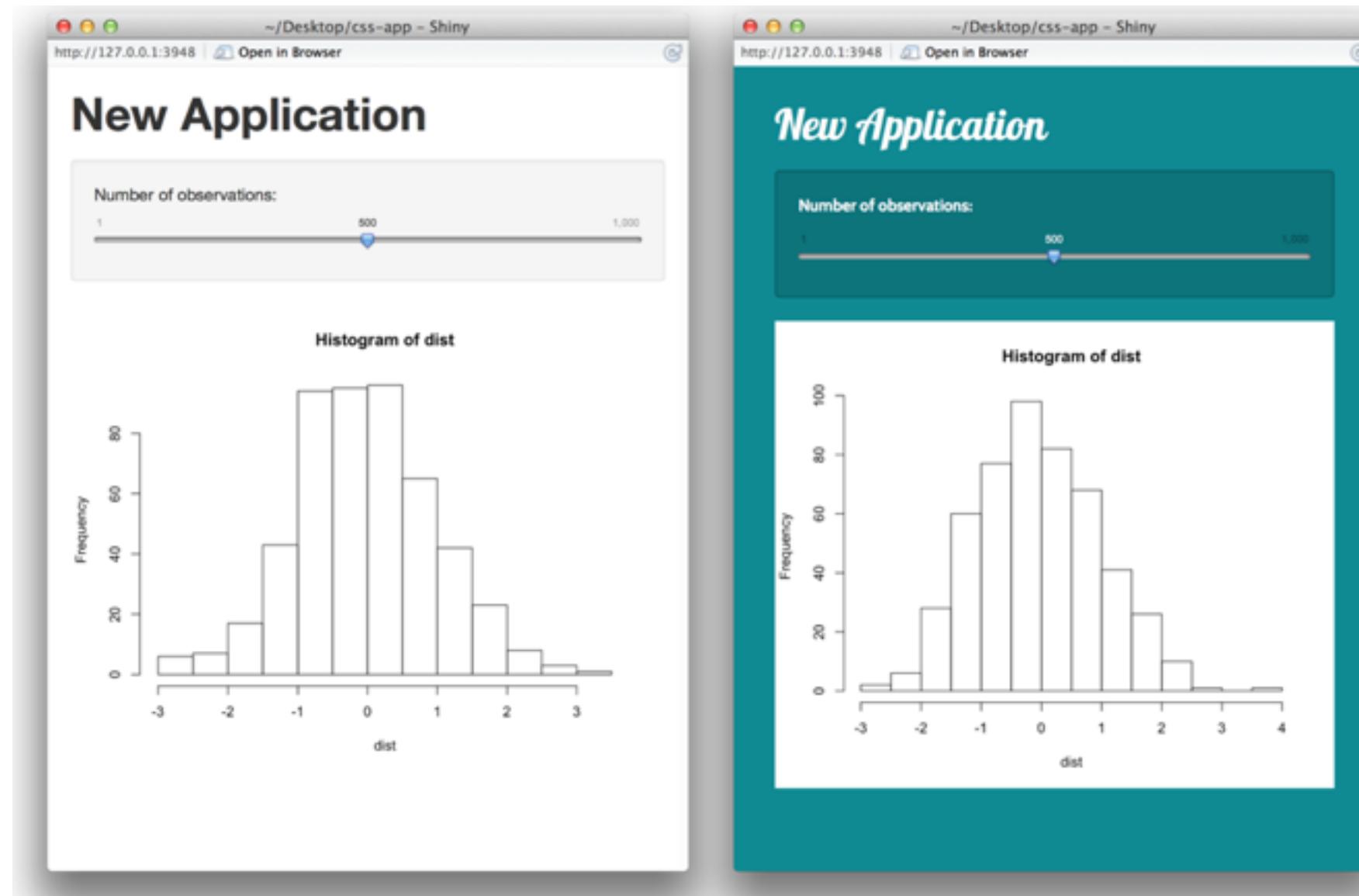
<http://www.codecademy.com/tracks/web>



I recommend the
free codecademy
tutorial

Customize your apps with HTML, CSS, and Javascript

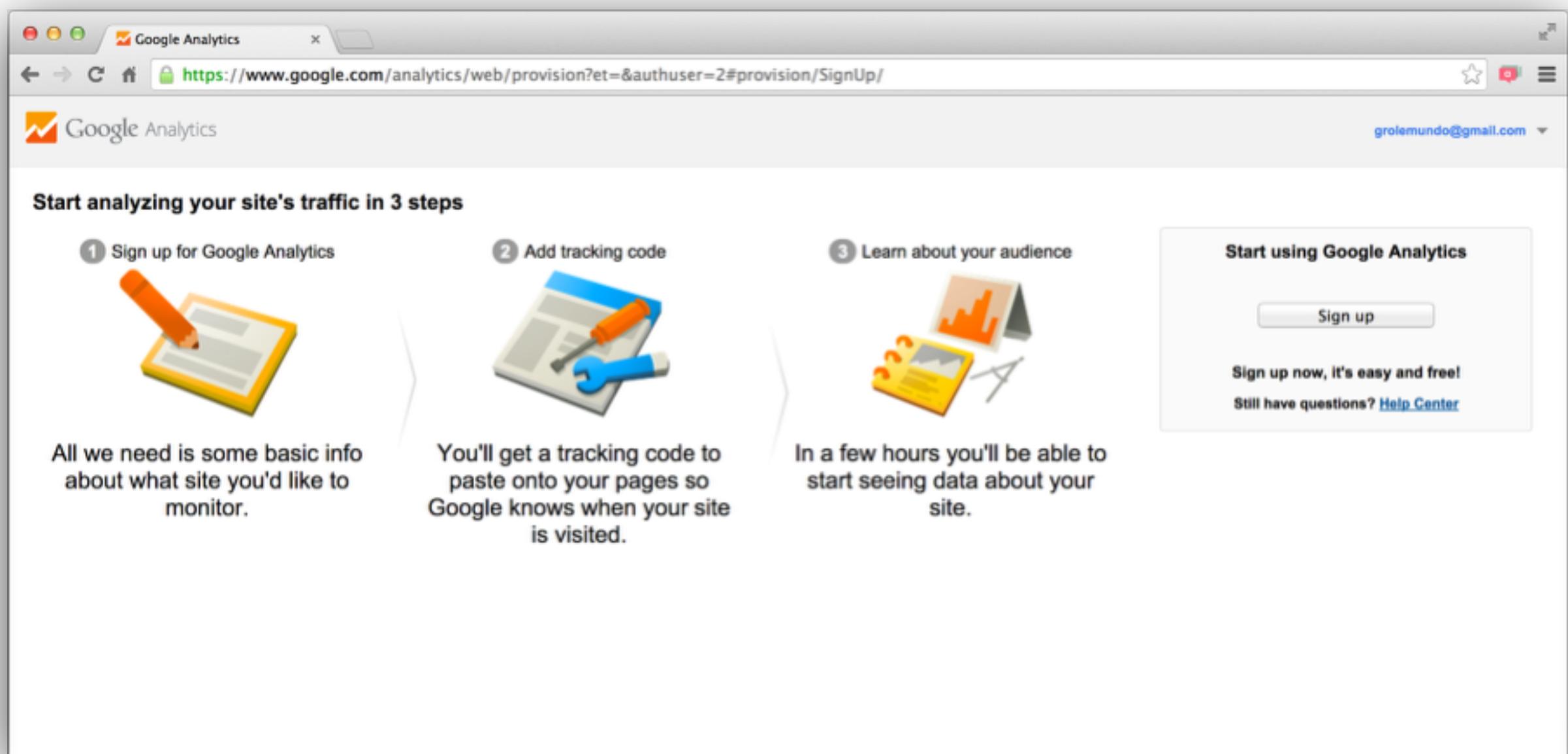
<http://shiny.rstudio.com/articles/css.html>



You can pair any app with whatever web technologies you wish. The above guide explains how to style your app with CSS.

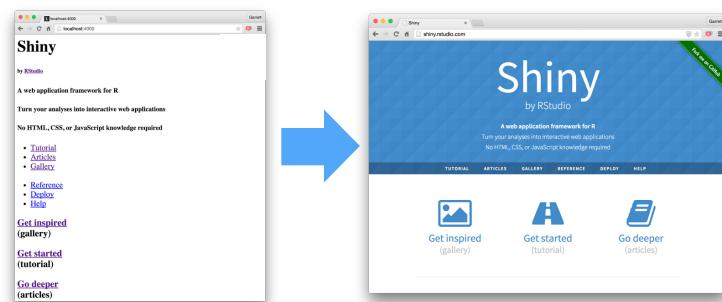
Add Google Analytics to a Shiny app

<http://shiny.rstudio.com/articles/google-analytics.html>

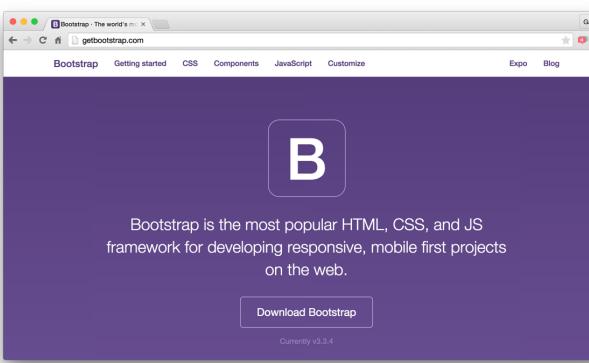


A case study in using jQuery to track visitor actions with Google Analytics

Recap: Style with CSS



Style Shiny apps like web pages: with CSS.



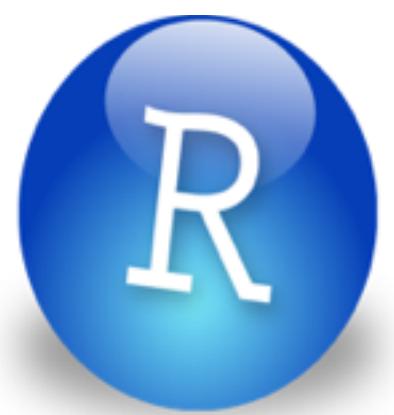
Shiny's general CSS classes come from the
bootstrap 3 framework

```
names(tags)
## [1] "a"      "abbr"   "address" "area"
## [5] "article" "aside"  "audio"   "audioio"
## [9] "b"      "bdi"    "bdo"    "blockquote"
## [13] "body"   "br"     "button"  "canvas"
## [17] "caption" "cite"   "code"    "col"
## [21] "colgroup" "command" "data"    "datalist"
## [25] "del"    "details" "dfn"    "em"
## [29] "div"    "dt"     "eventsource" "figcaption"
## [33] "embed"  "eventsource" "fieldset" "figmap"
## [37] "figure" "footer"  "form"    "h1"
## [41] "h2"    "h3"     "h4"     "h5"
## [45] "h6"    "head"   "header"  "hgroup"
```

You can recreate HTML methods for including
CSS with **tags\$head()**, **tags\$link()** and
tags\$style()

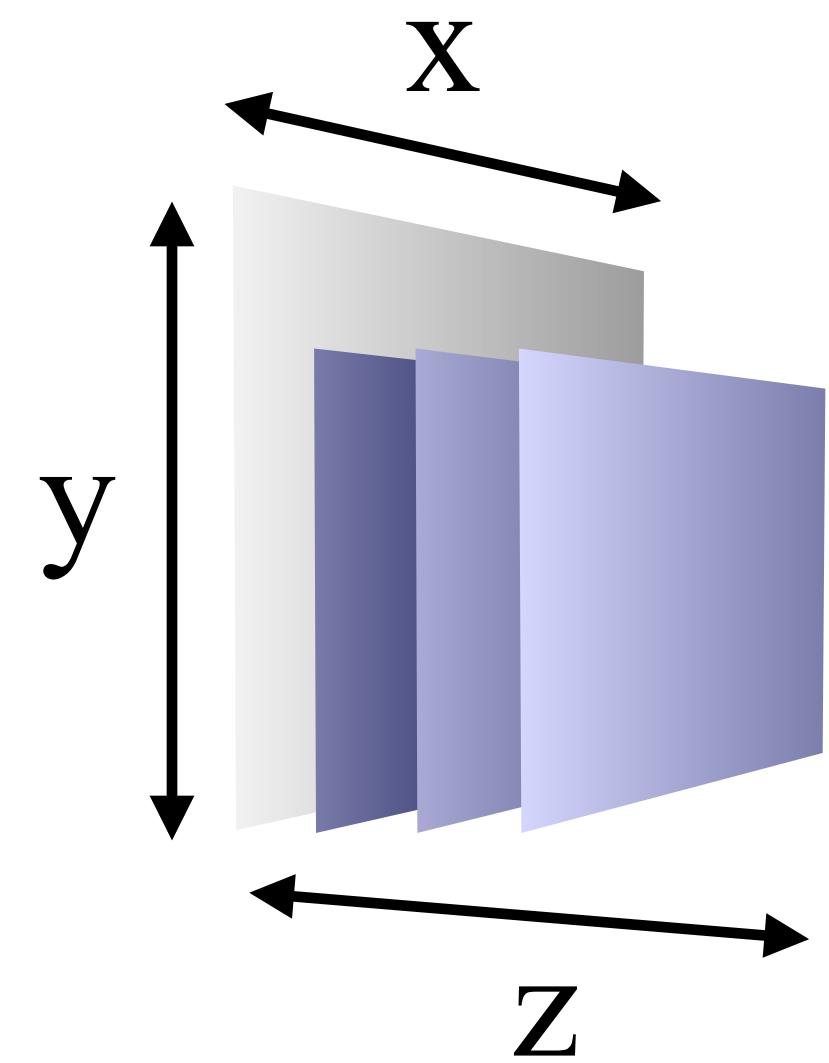
You now how to

My Shiny App

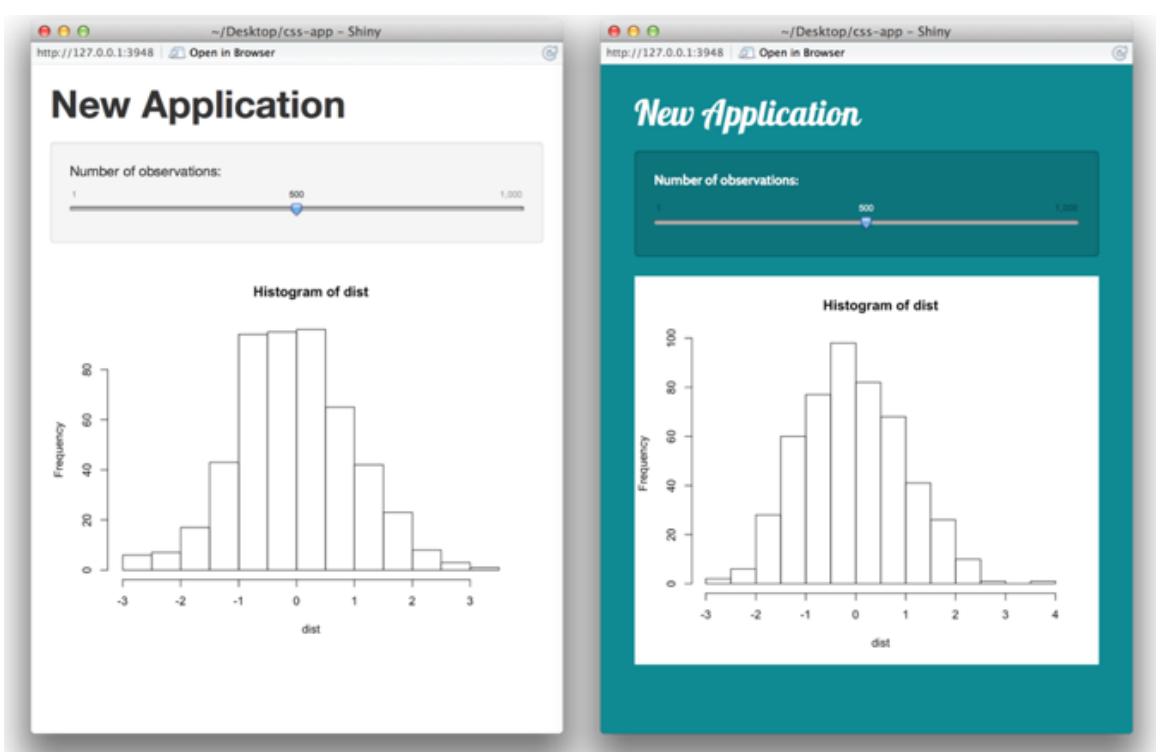


See other apps in
the [Shiny Showcase](#)

Add static
elements

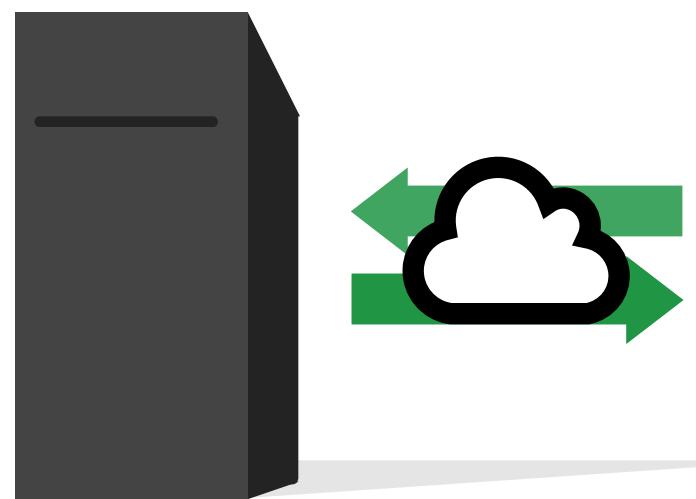


Lay out
elements

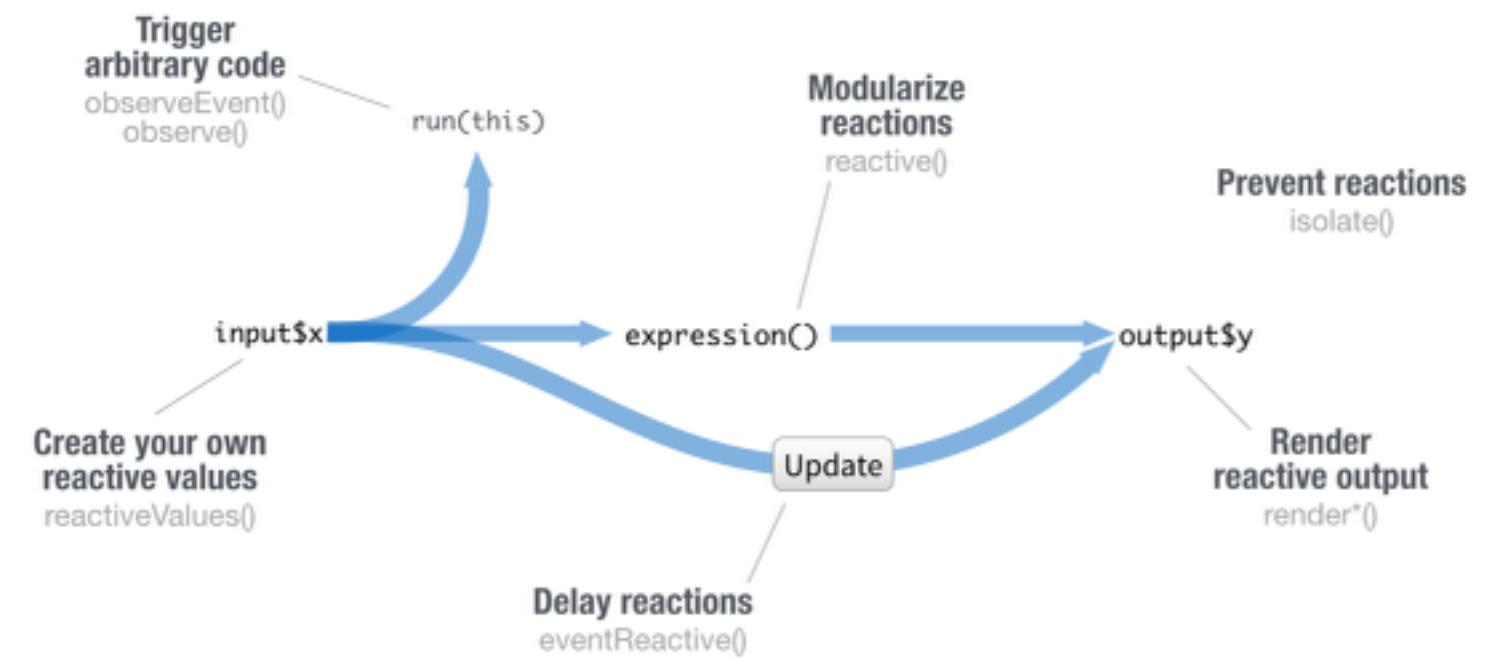
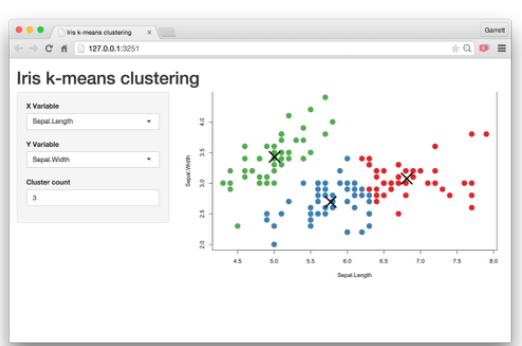


Style elements
with CSS

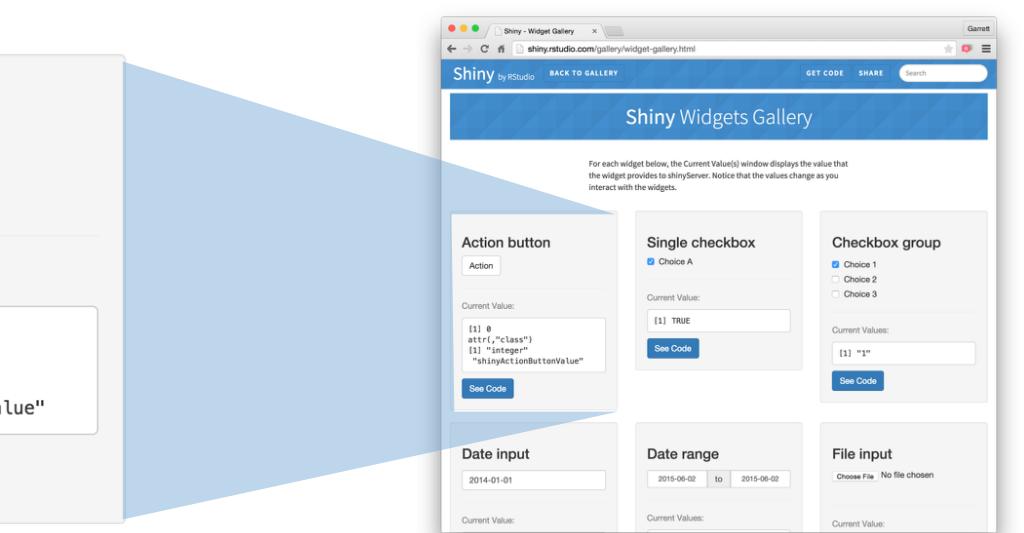
How to start with Shiny



Build and
share apps
(Part 1)



Control
reactions
(Part 2)



Customize
appearance
(Part 3)

**Where to go
from here**

**Experiment and practice
build your own
apps**

The Shiny Development Center

shiny.rstudio.com

