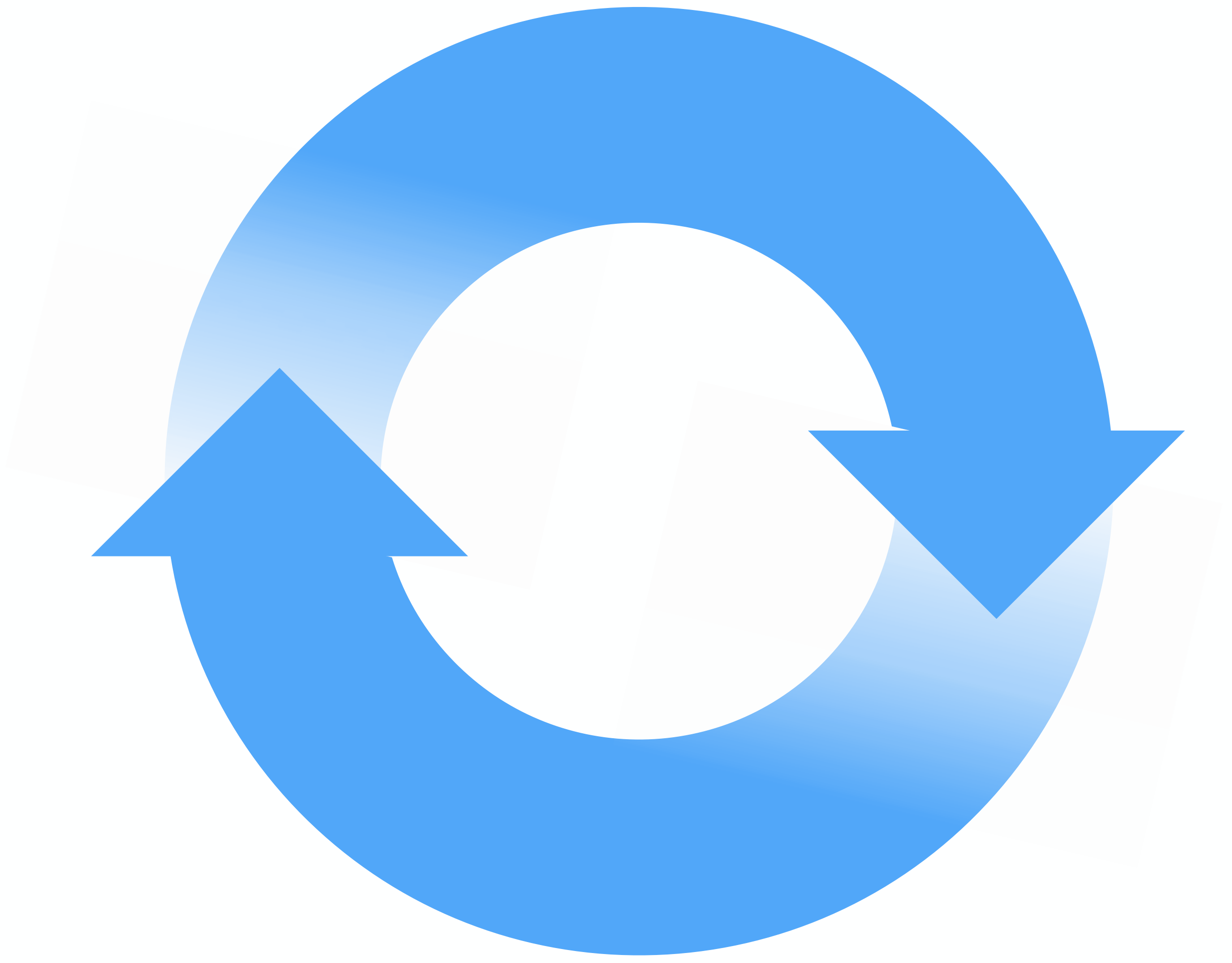
How to start with Shiny, Part 2

How to customize reactions



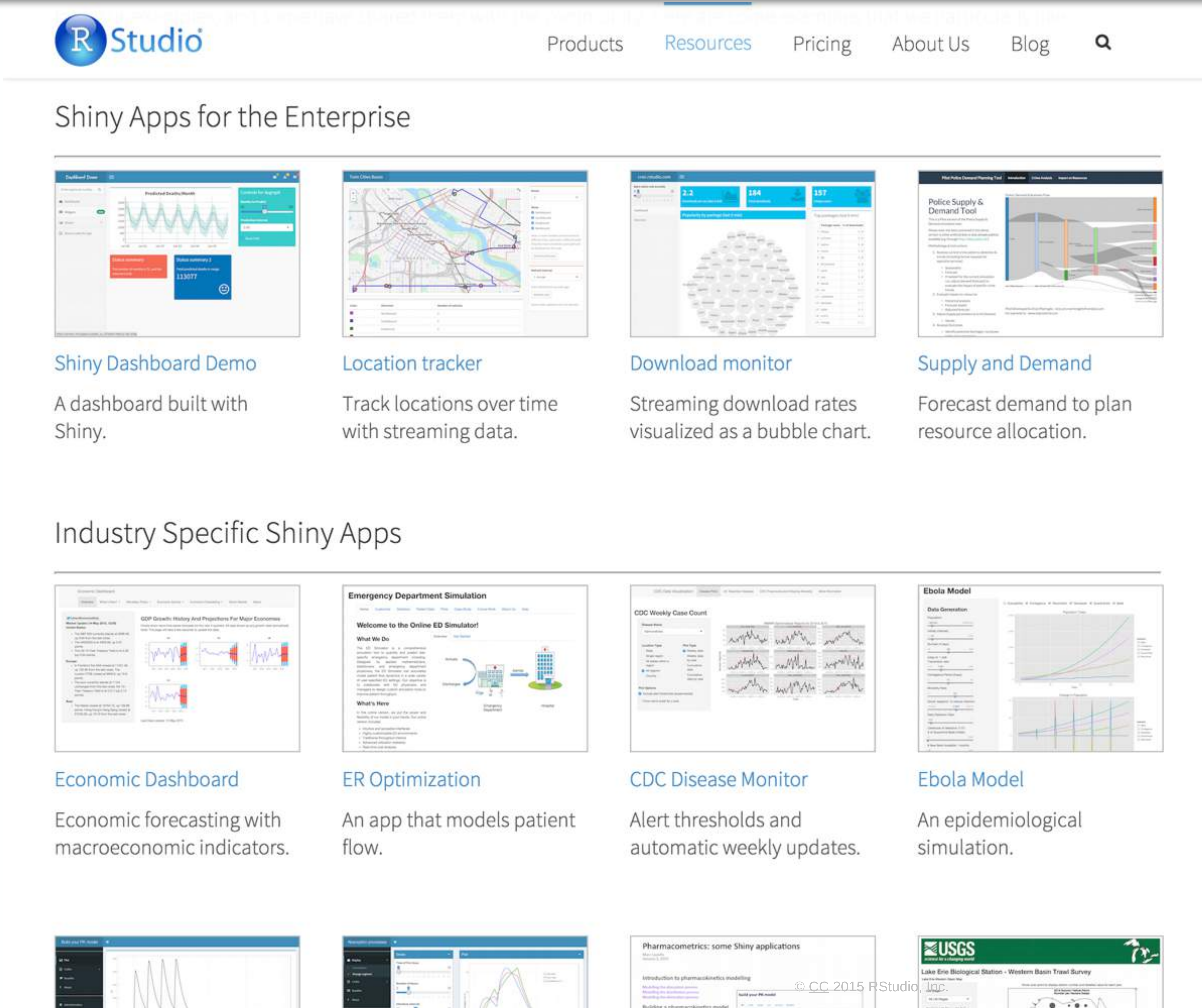
Garrett Grolemund
Data Scientist and Master Instructor
May 2015
Email: garrett@rstudio.com
Twitter: @StatGarrett

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.

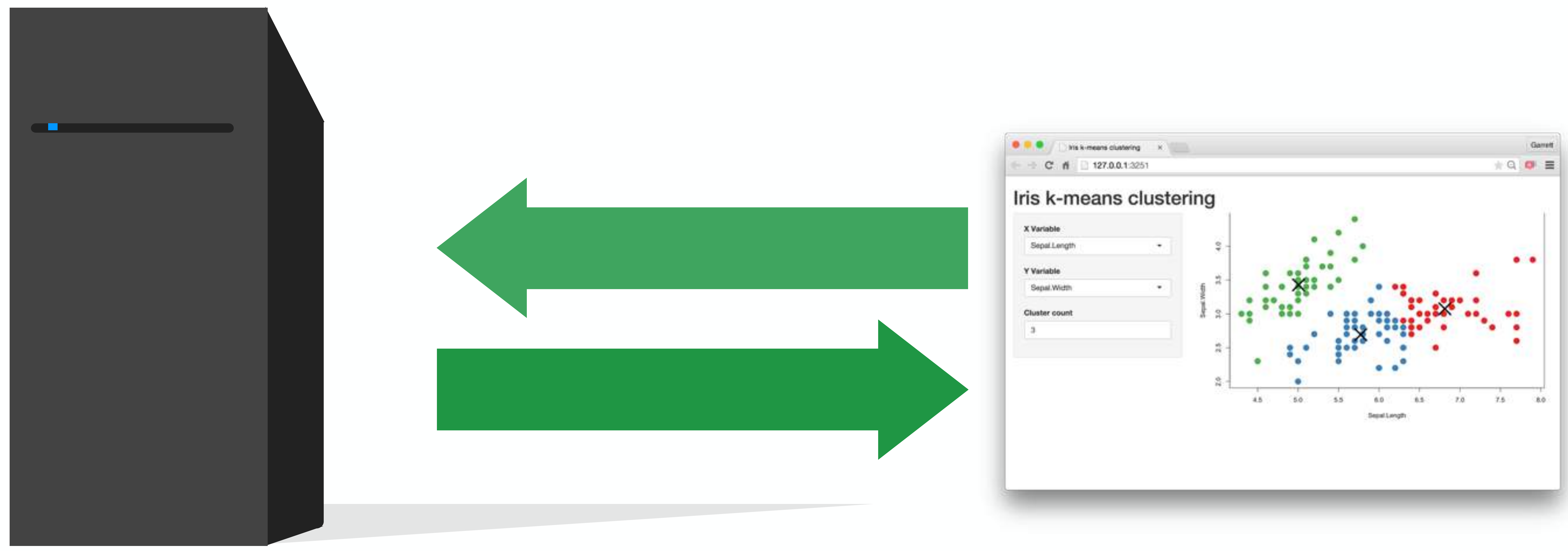
Ebola Model
An epidemiological simulation.

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

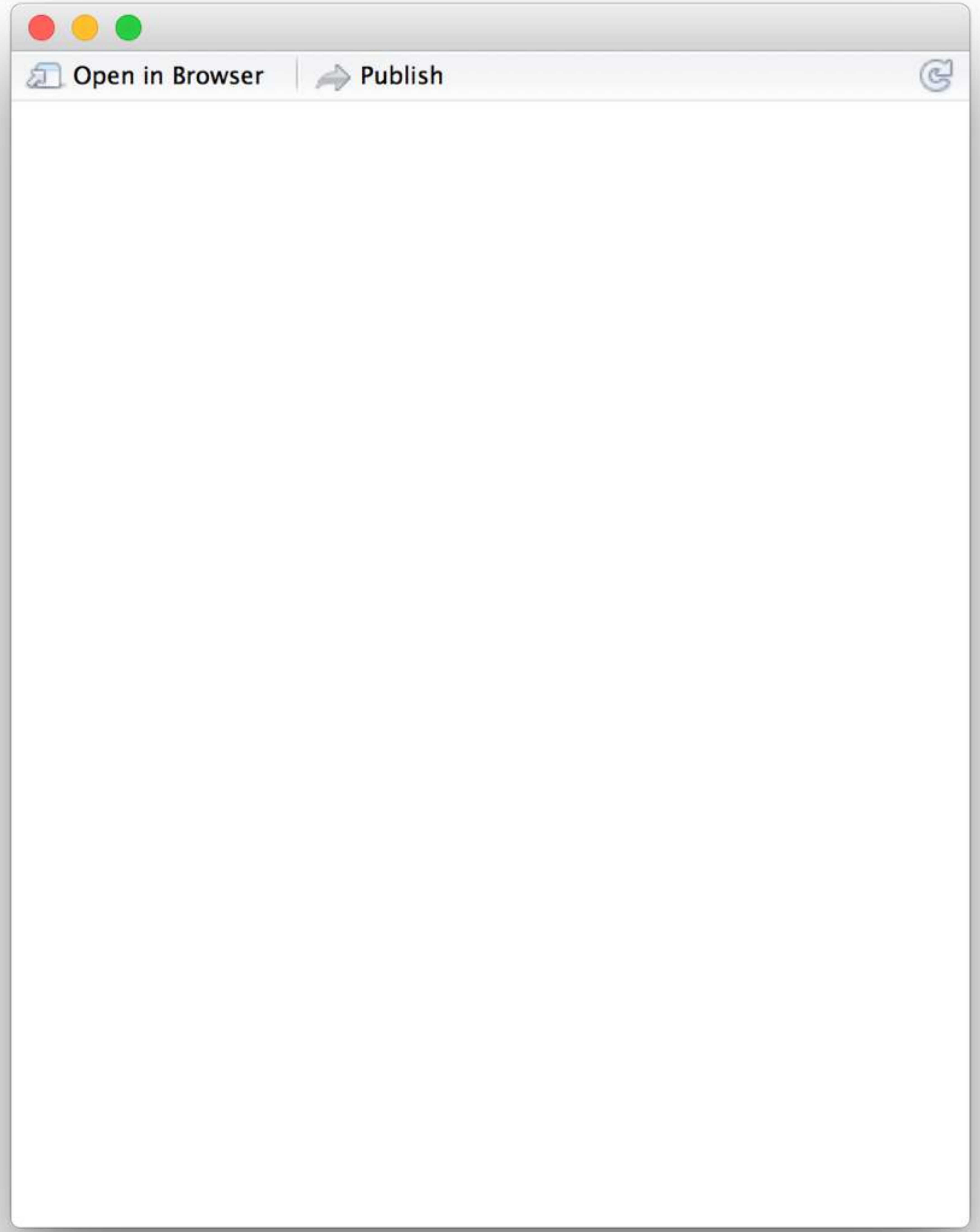


App template

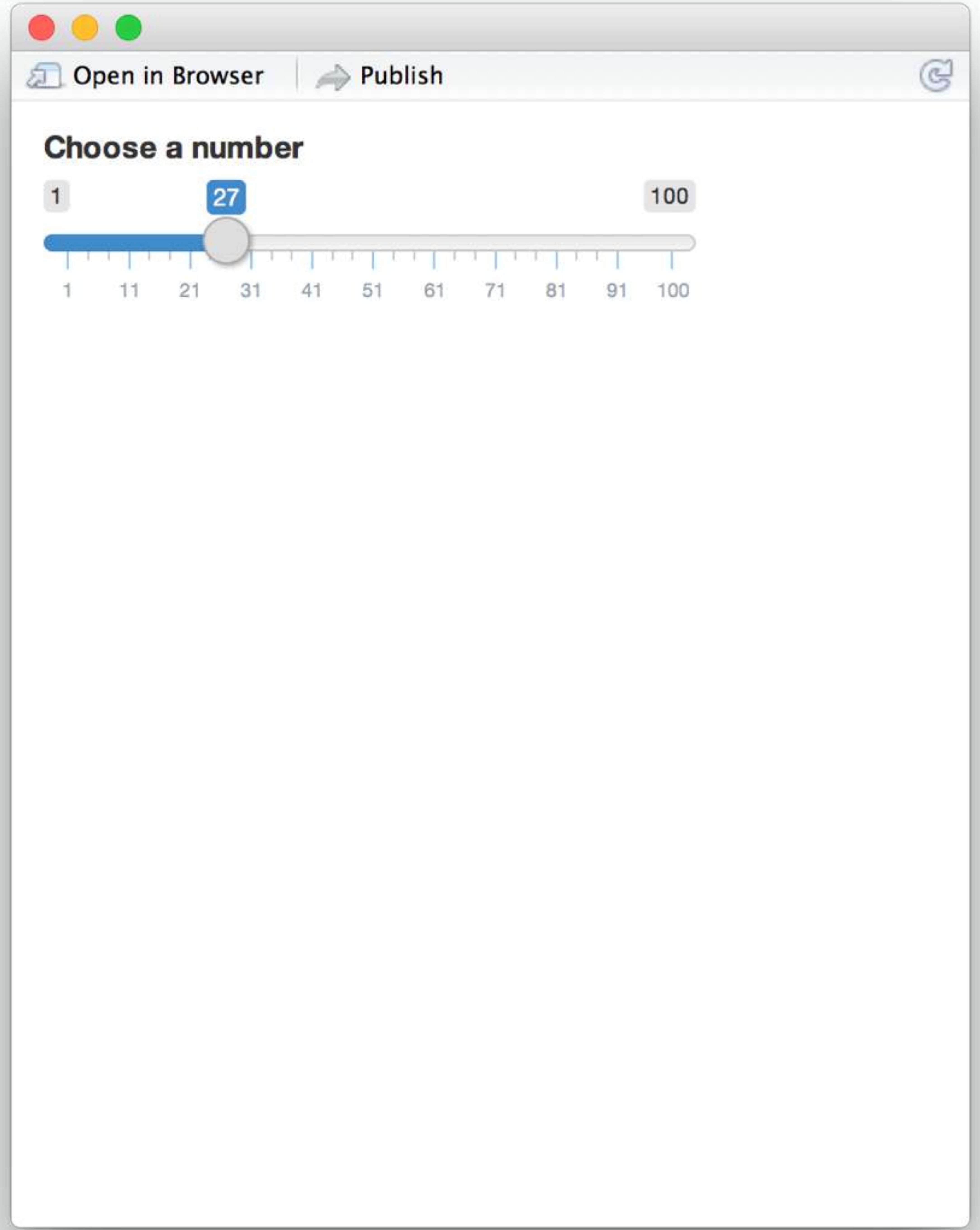
The shortest viable shiny app

```r
library(shiny)

ui <- fluidPage()


server <- function(input, output) {}


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

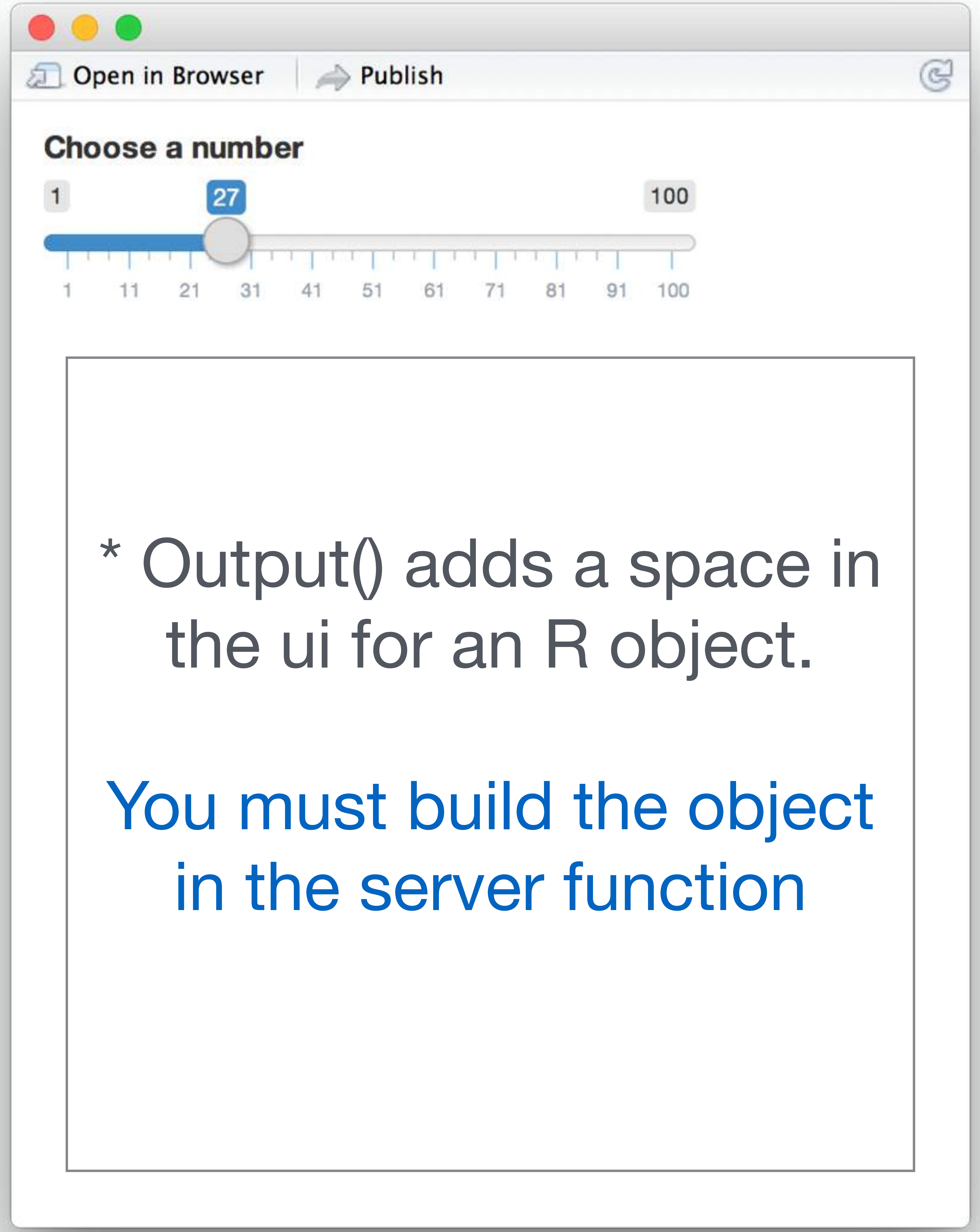
```
library(shiny)

ui <- fluidPage(




)

server <- function(input, output) {



}

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



```r
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)
```

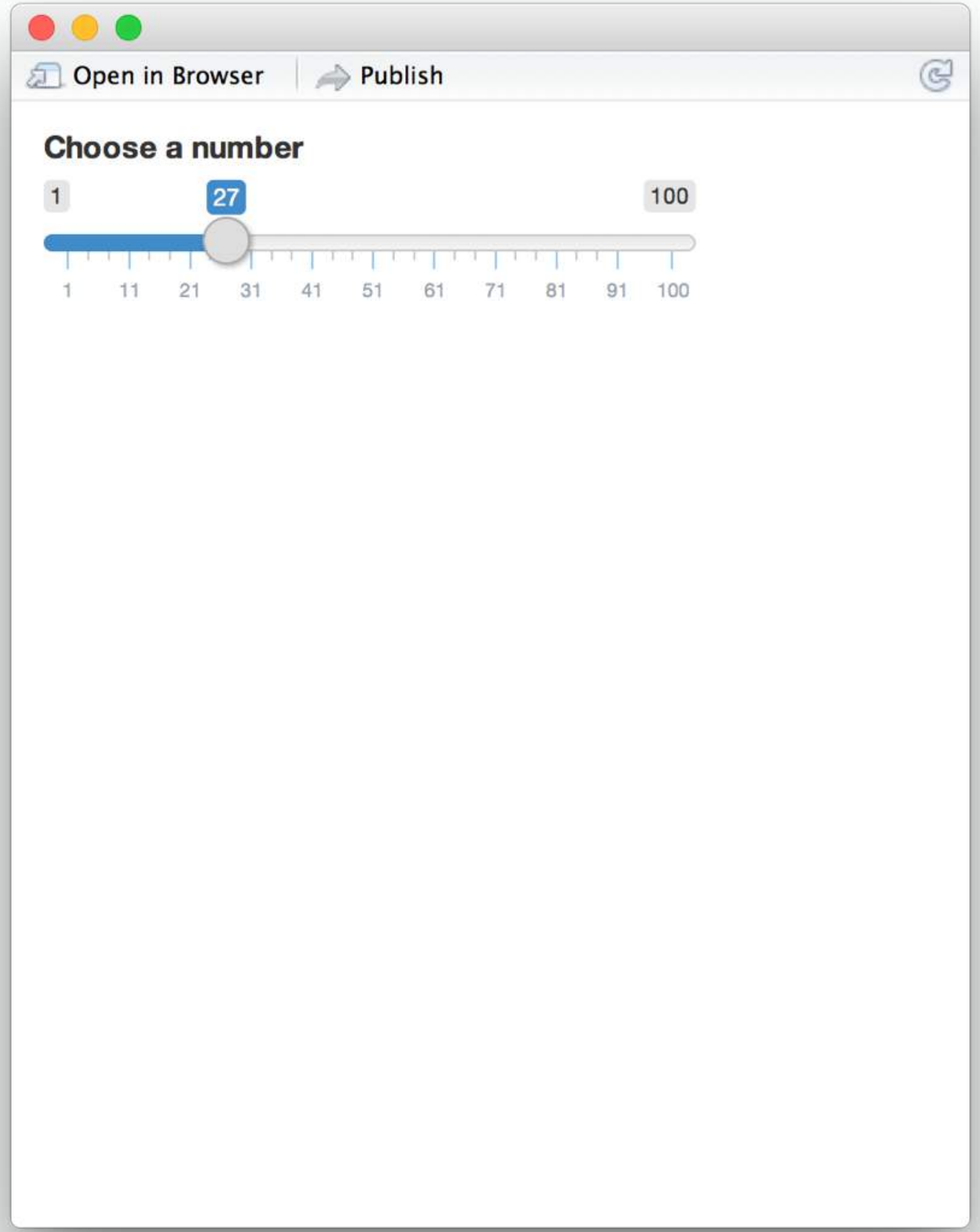


* Output() adds a space in the ui for an R object.

You must build the object in the server function

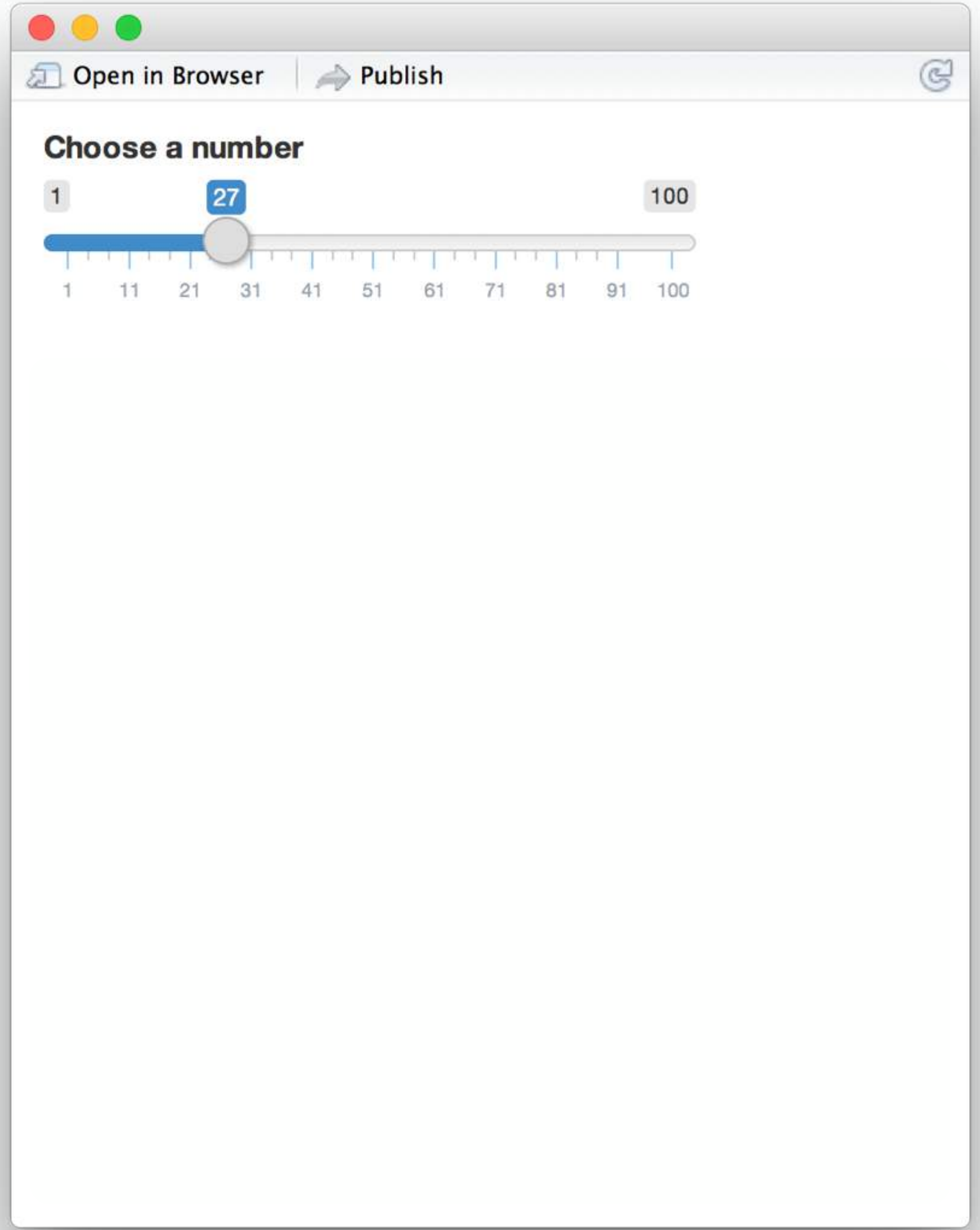
```
library(shiny)

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



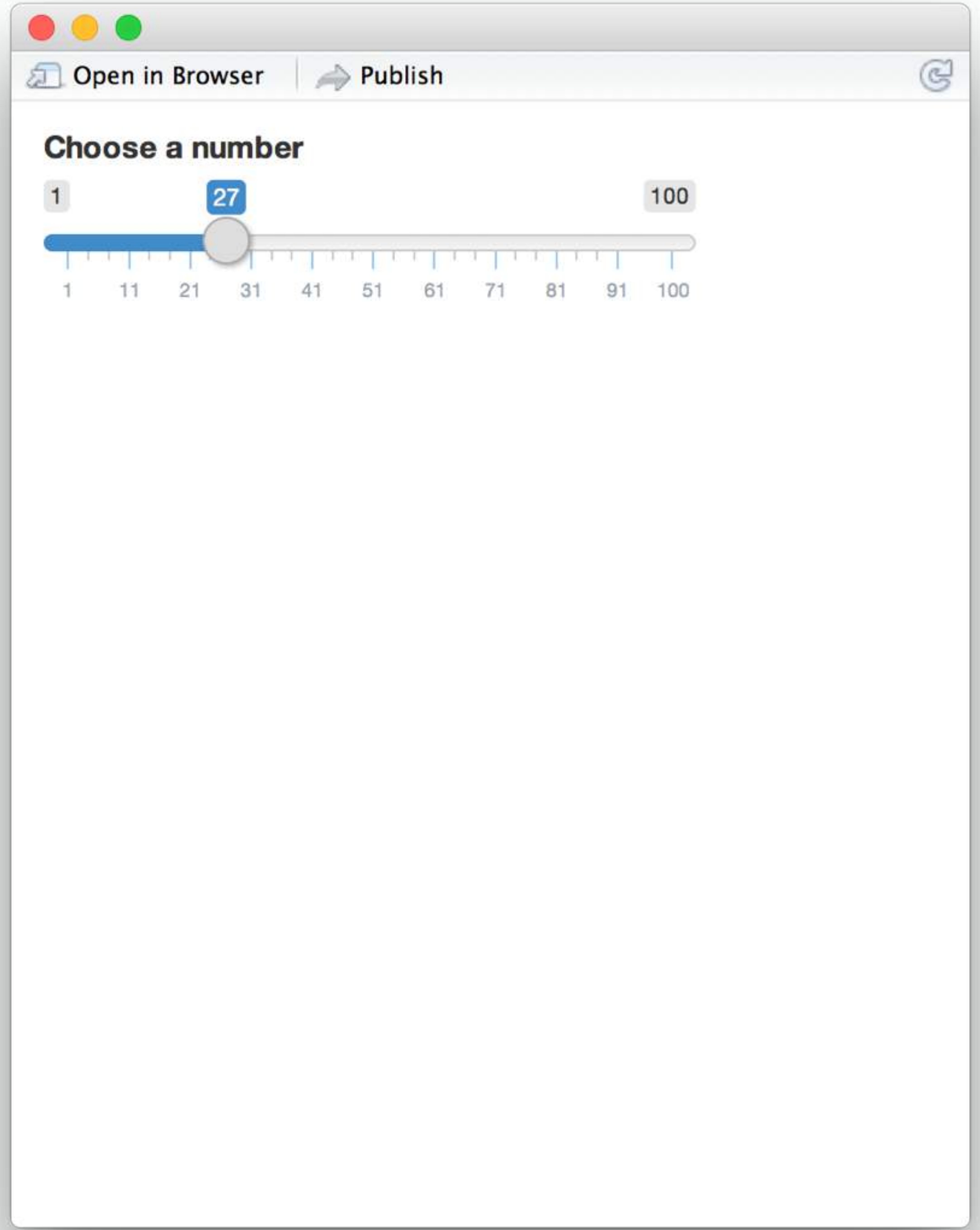
```r
library(shiny)

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

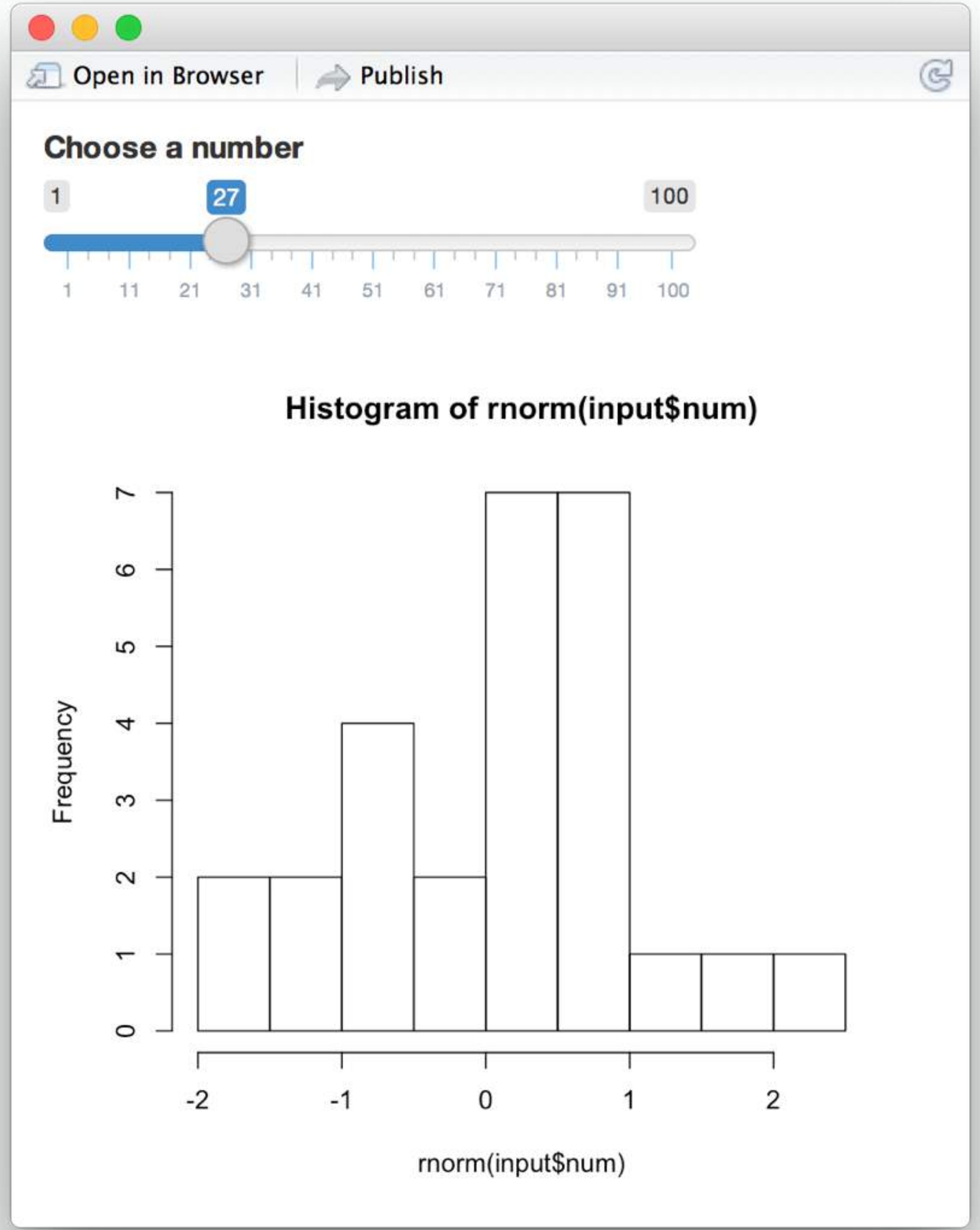


```
library(shiny)

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)
```



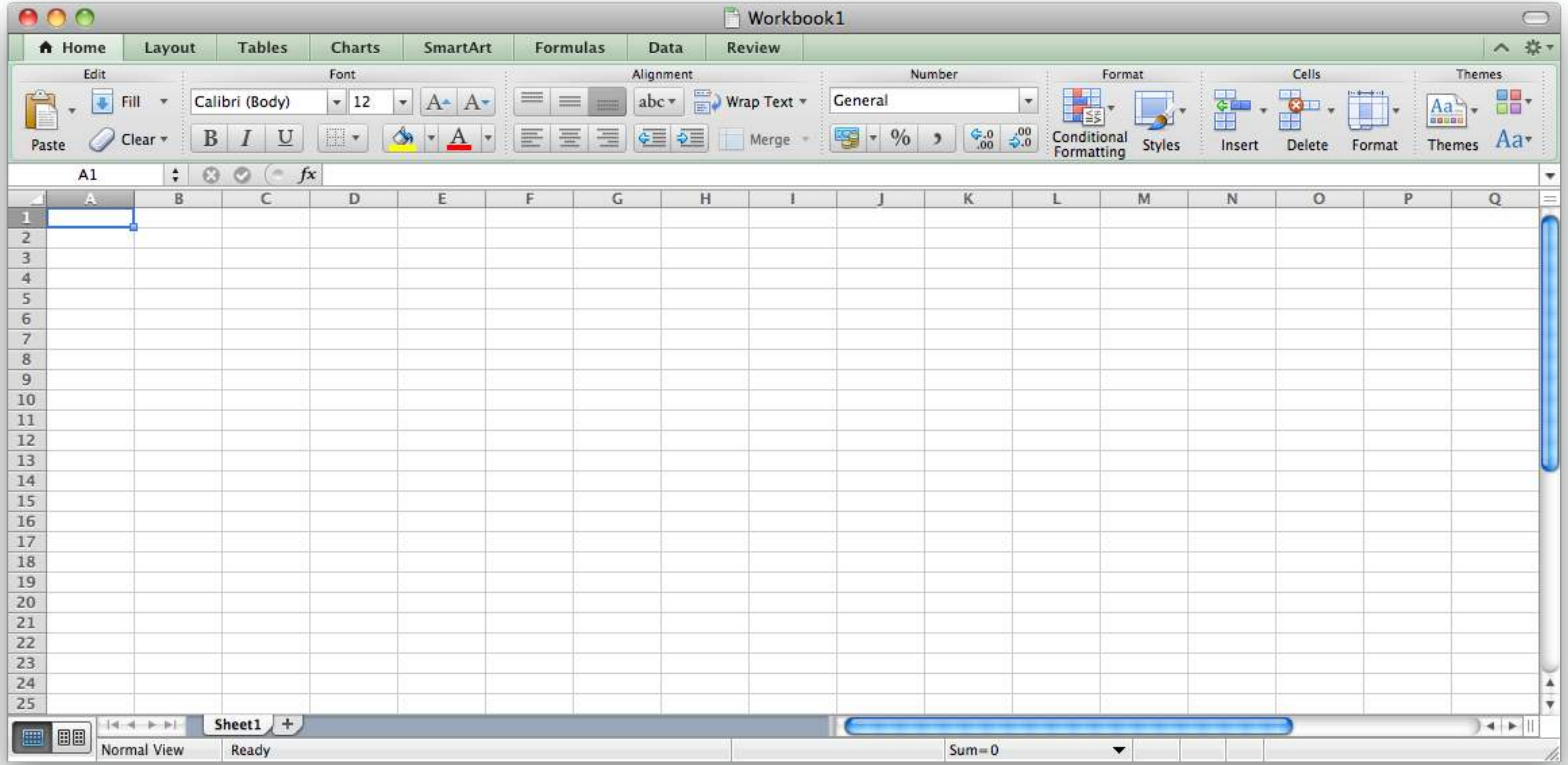
Sharing apps

Shiny Server (Pro)

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

What is Reactivity?

Think Excel.


Workbook1
Home
Layout
Tables
Charts
SmartArt
Formulas
Data
Review
Edit
Font
Alignment
Number
Format
Cells
Themes
Paste
Fill
Clear
Calibri (Body)
12
Wrap Text
Merge
General
Conditional Formatting
Styles
Insert
Delete
Format
Themes
A1
Sheet1
Normal View
Ready
Sum=0

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ri (Body)
12
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
fx
C
D
E
F
G
H
I
J
K
L
M
N
O

50				= F4 + 1

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete

50

51

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ri (Body)
12
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
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
ri (Body)
12
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
fx
C
D
E
F
G
H
I
J
K
L
M
N
O

999

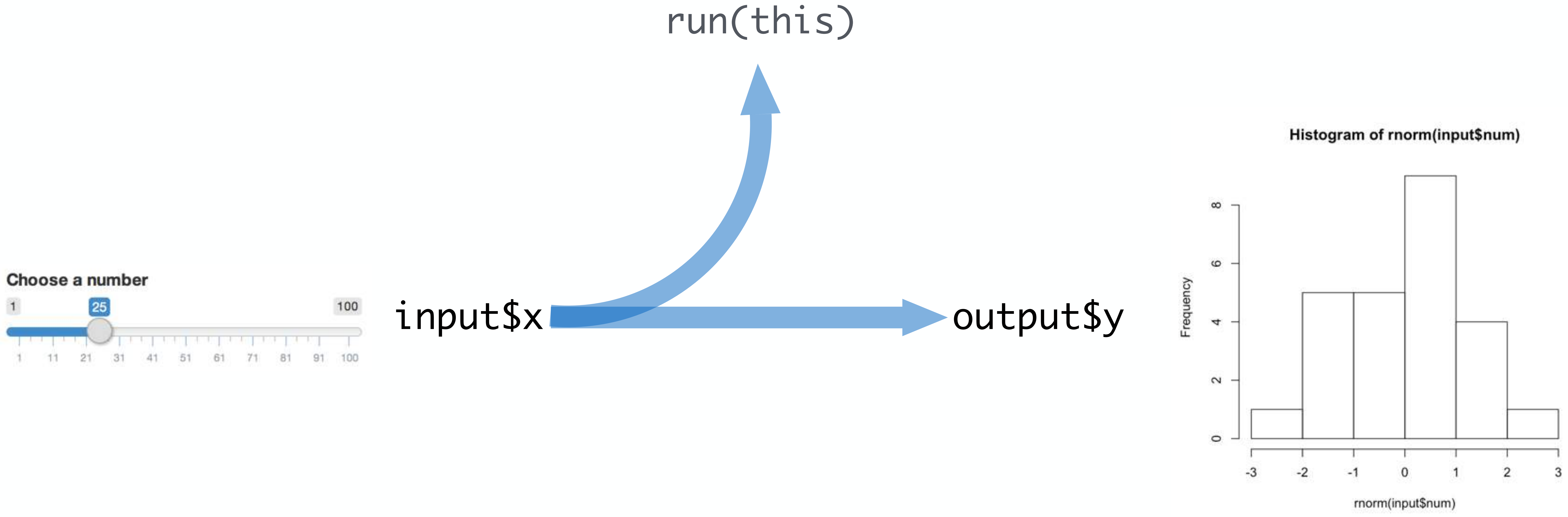
1000

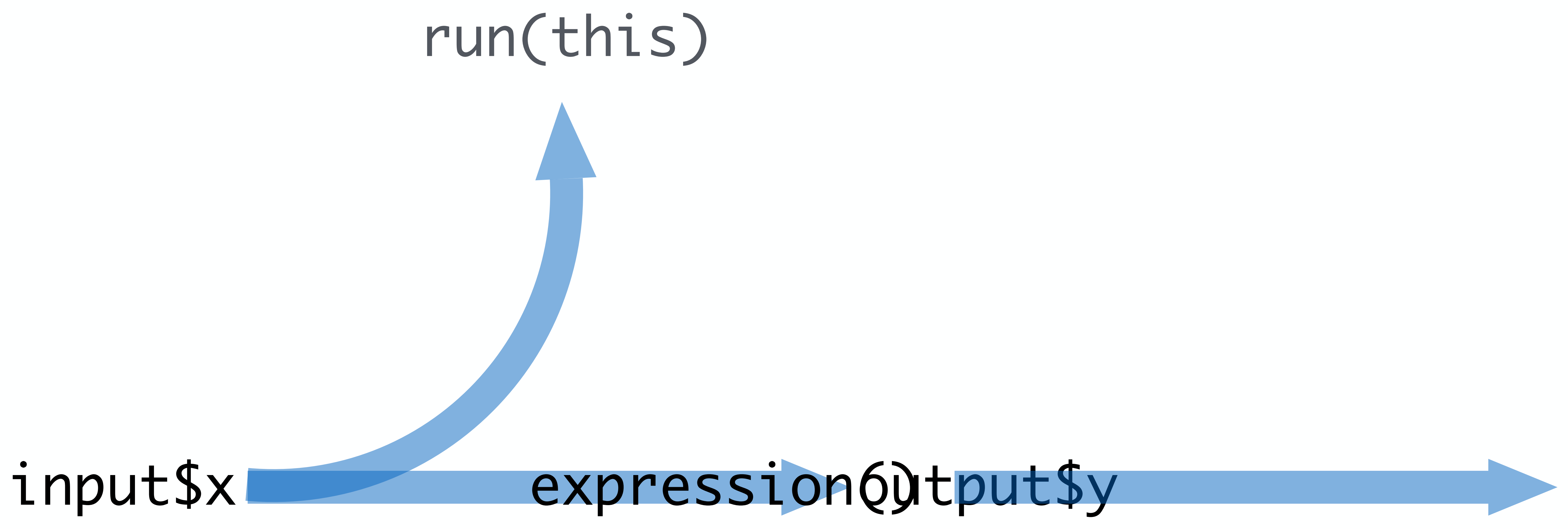
Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
ri (Body)
12
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
C
D
E
F
G
H
I
J
K
L
M
N
O

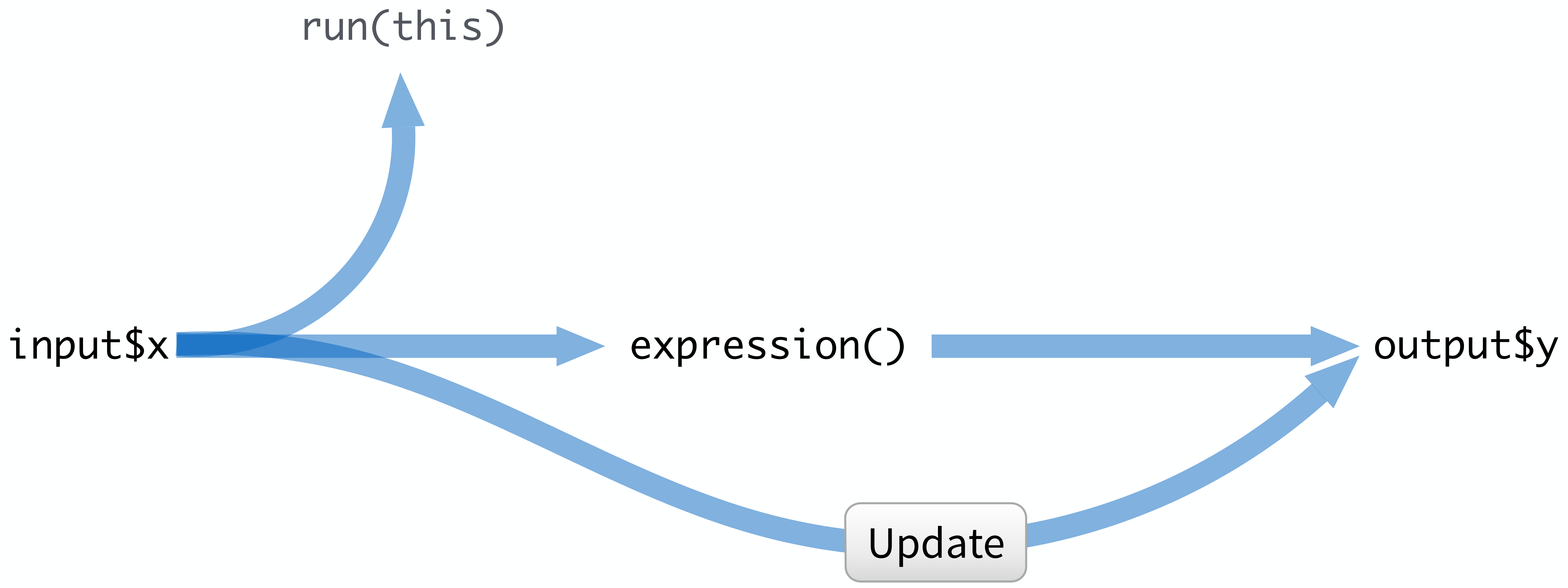
1000

1001

Workbook1
Tables
Charts
SmartArt
Formulas
Data
Review
Font
Alignment
Number
Format
Cells
Wrap Text
General
Merge
Conditional Formatting
Styles
Insert
Delete
1000
1001
input$x
output$y

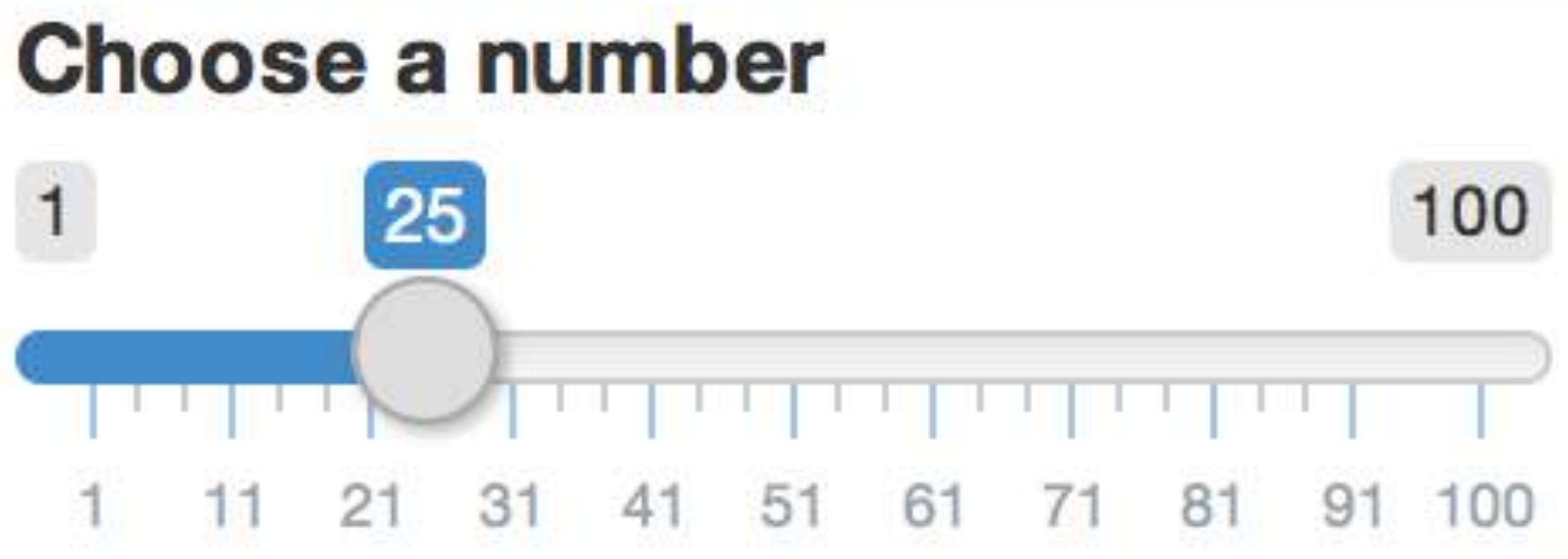

run(this)
Choose a number
1
25
100
1 11 21 31 41 51 61 71 81 91 100
input$x
output$y
Histogram of rnorm(input$num)
Frequency
0 2 4 6 8
-3 -2 -1 0 1 2 3
rnorm(input$num)


run(this)
input$x
expression()
output$y


run(this)
input$x
expression()
output$y
Update

Begin with
reactive values

Syntax

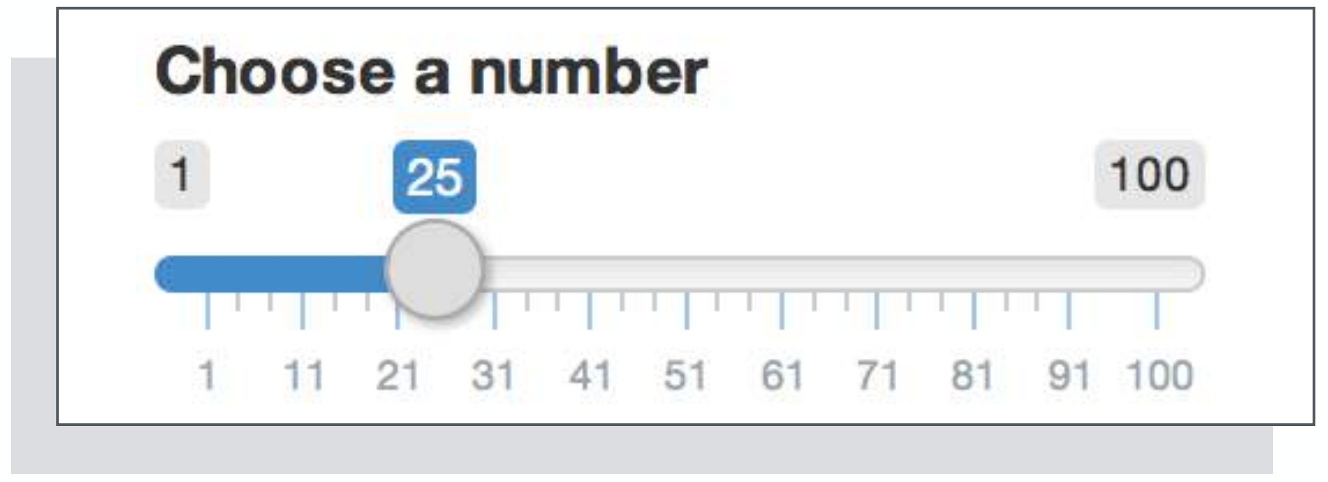


```
sliderInput(inputId = "num", label = "Choose a number", …)
```

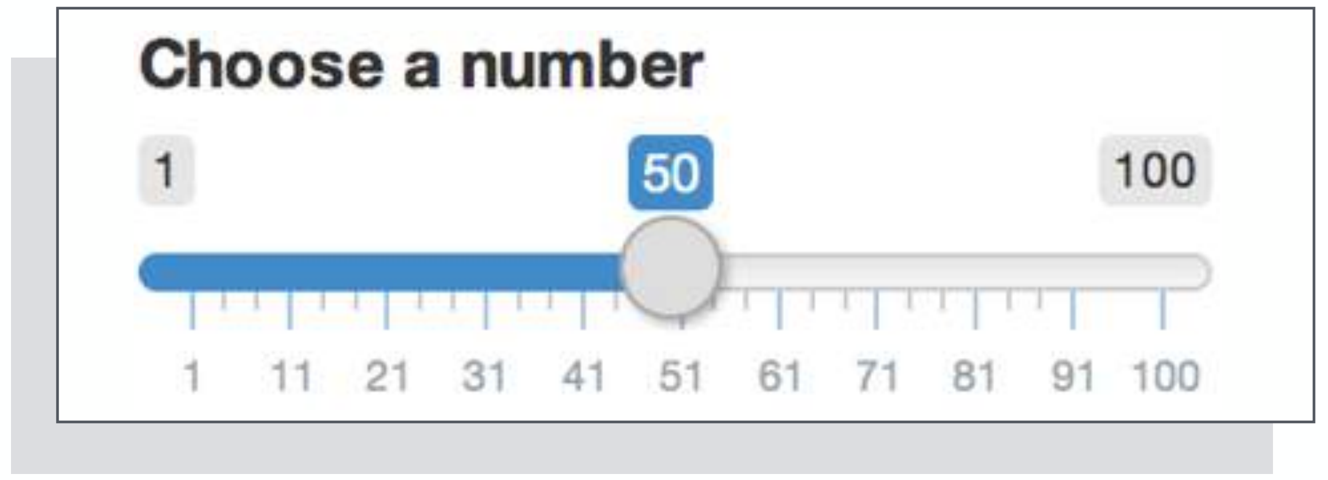
this input will provide a value saved as **input$num**

Input values

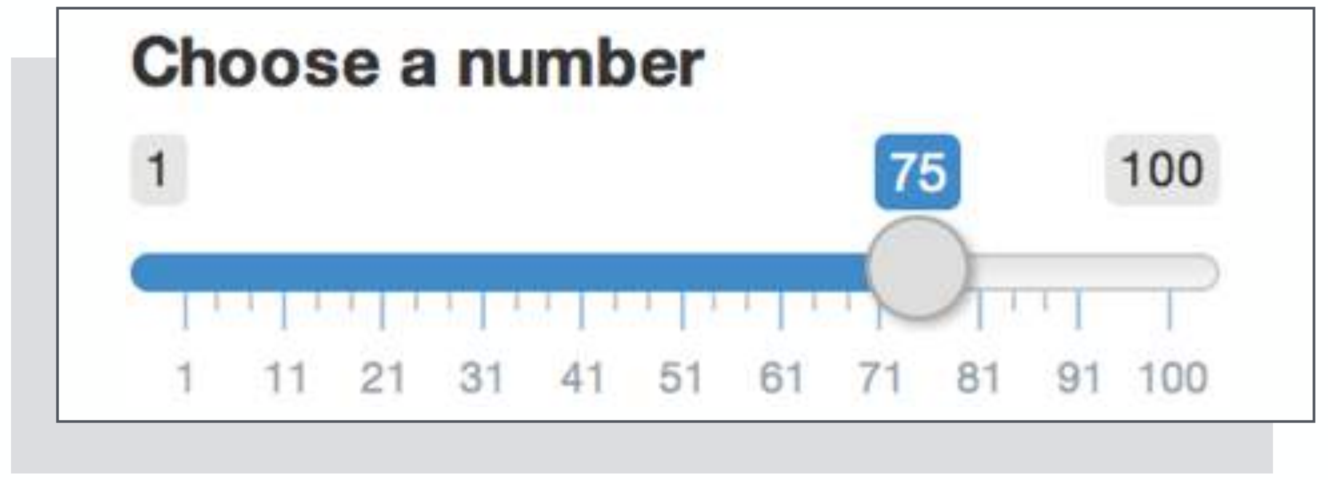
The input value changes whenever a user changes the input.



```
input$num = 25
```



```
input$num = 50
```



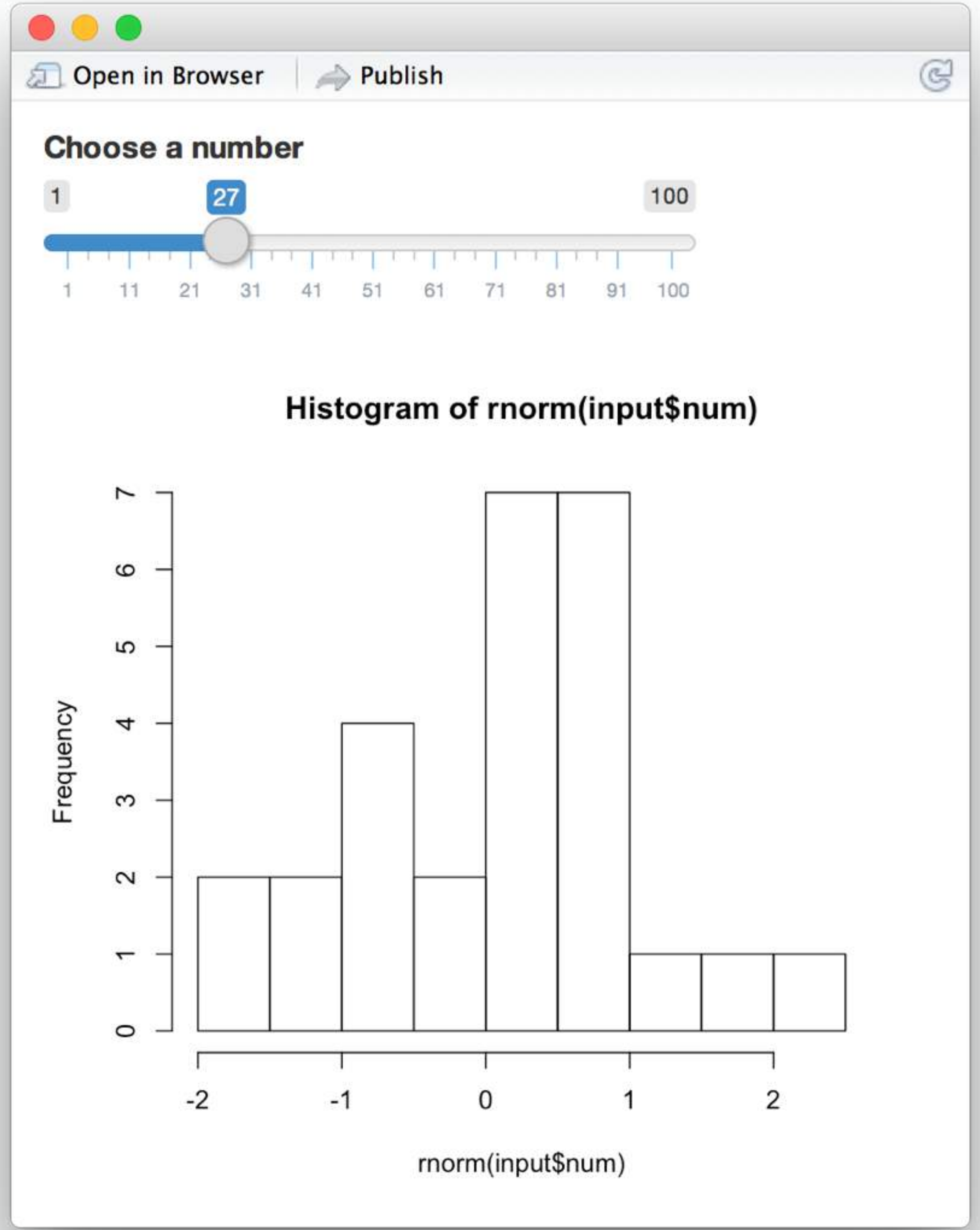
```
input$num = 75
```

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

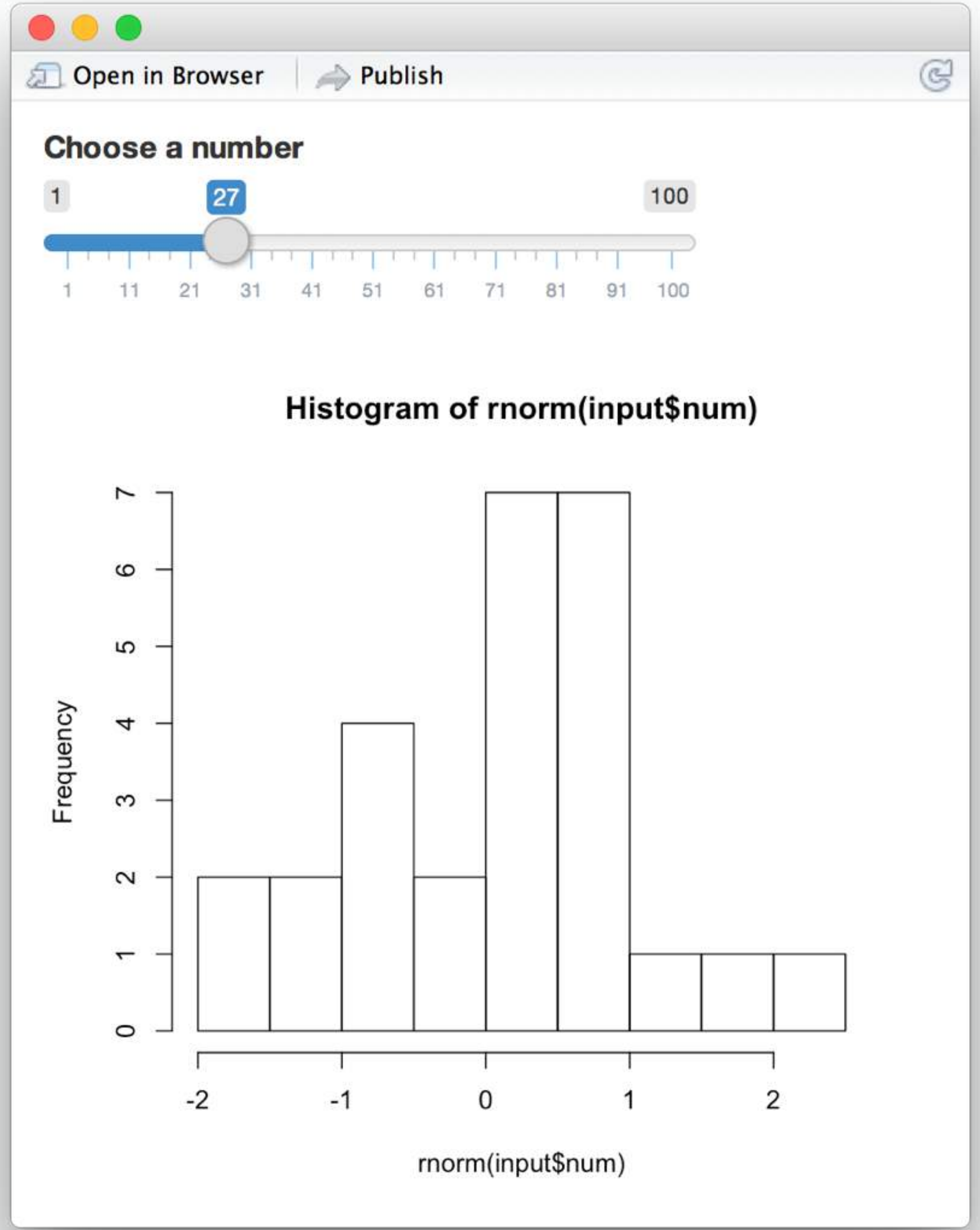
✓
```
renderPlot({ hist(rnorm(100, input$num)) })
```

⚠
```
hist(rnorm(100, input$num))
```

```r
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)
```



```r
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)
```

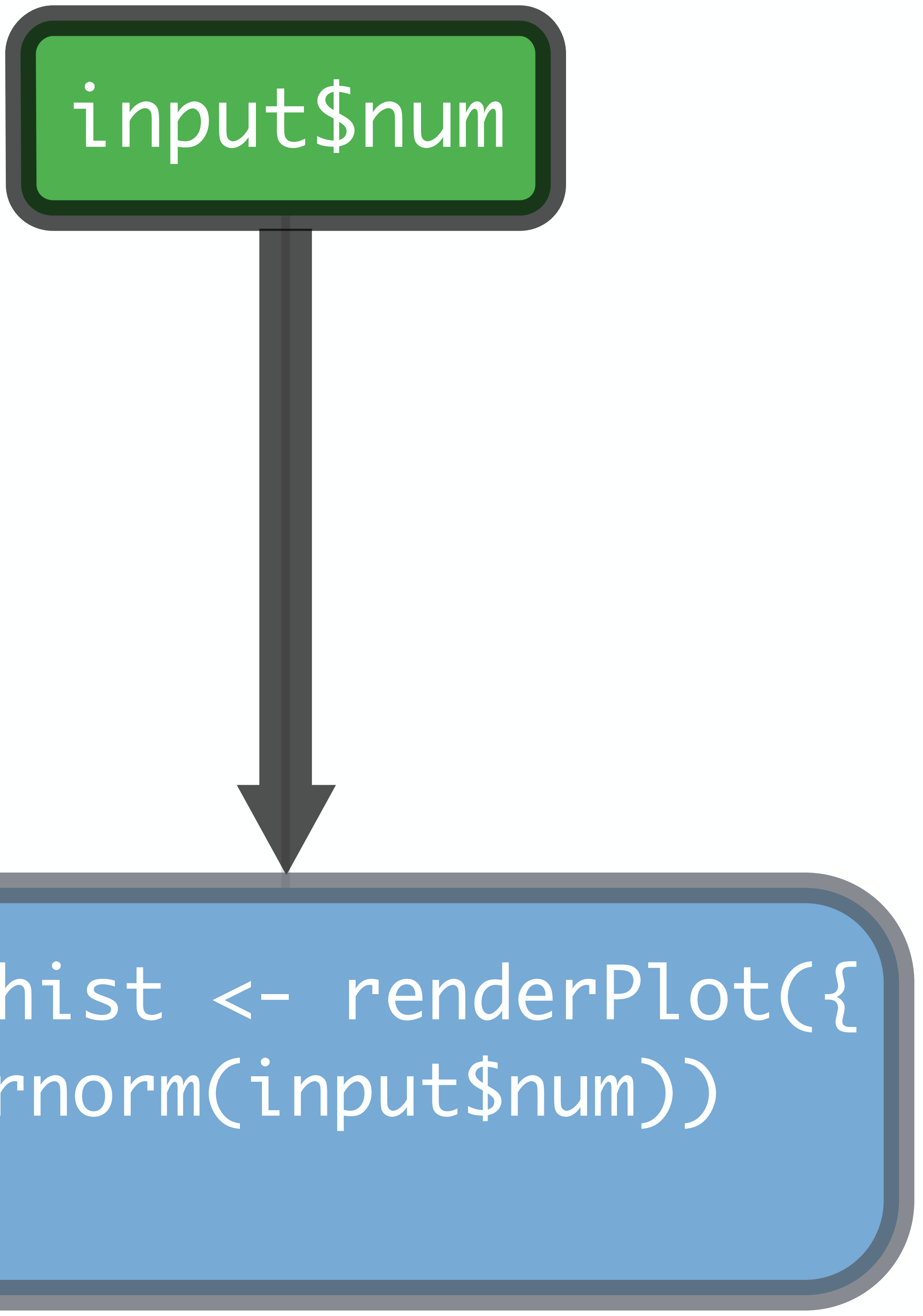


```r
library(shiny)

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)
```

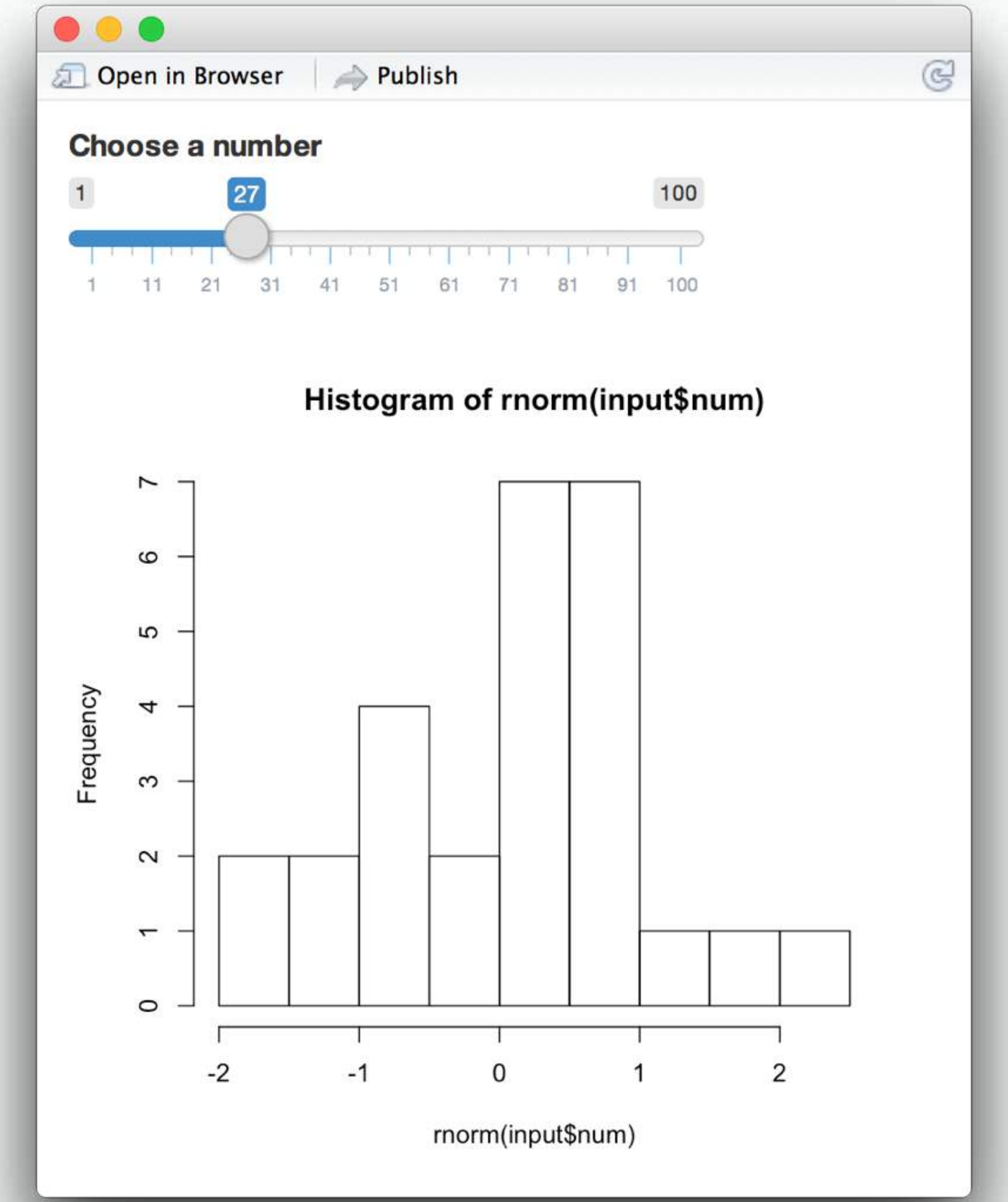
```
 Error in .getReactiveEnvironment()
$currentContext() :
  Operation not allowed without an
active reactive context. (You tried
to do something that can only be done
from inside a reactive expression or
observer.)
```

Think of reactivity in R as a two step process

1 Reactive values notify
the functions that use them when they become invalid



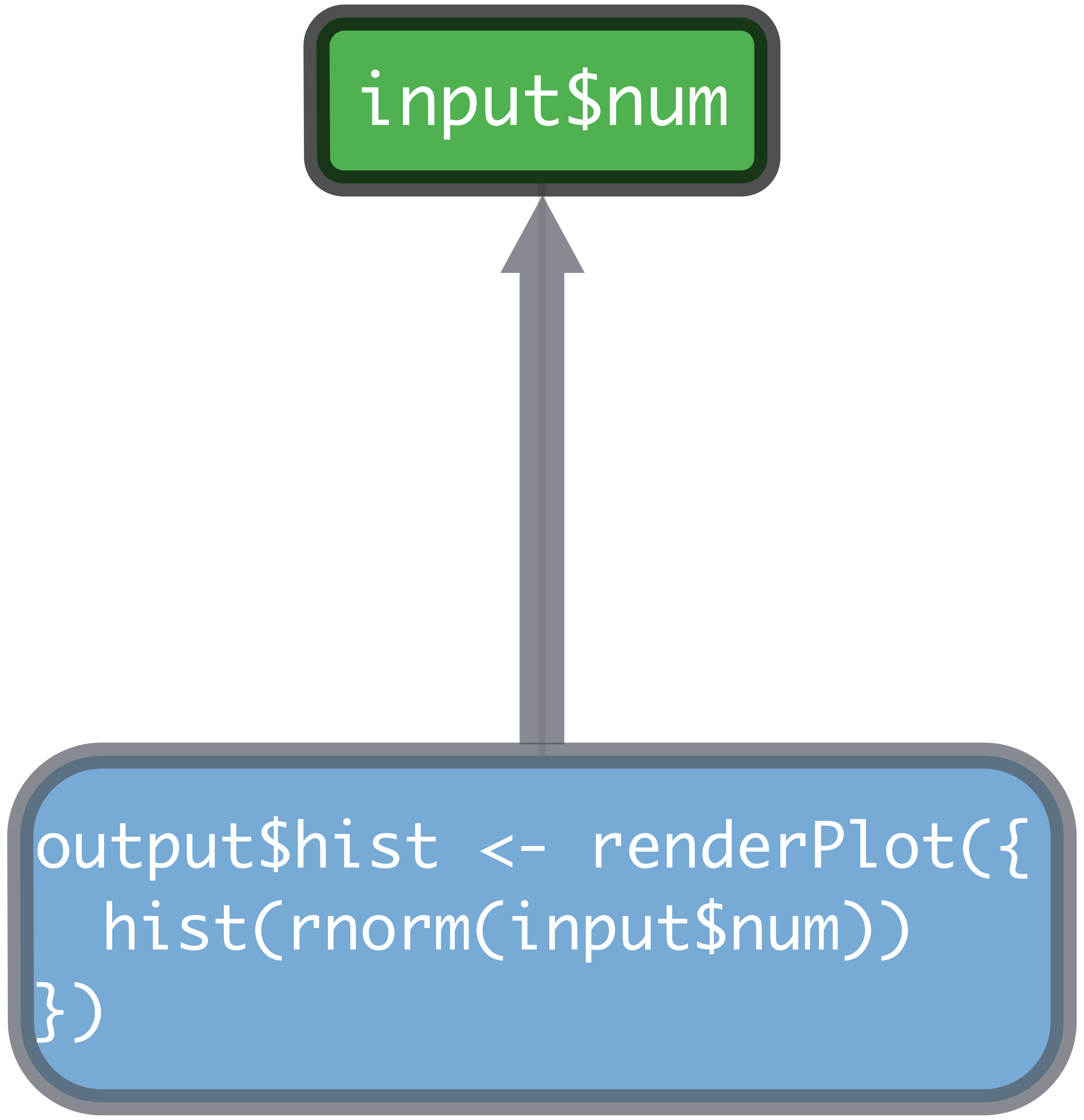
```
output$hist <- renderPlot({
  hist(rnorm(input$num))
})
```



Think of reactivity in R as a two step process

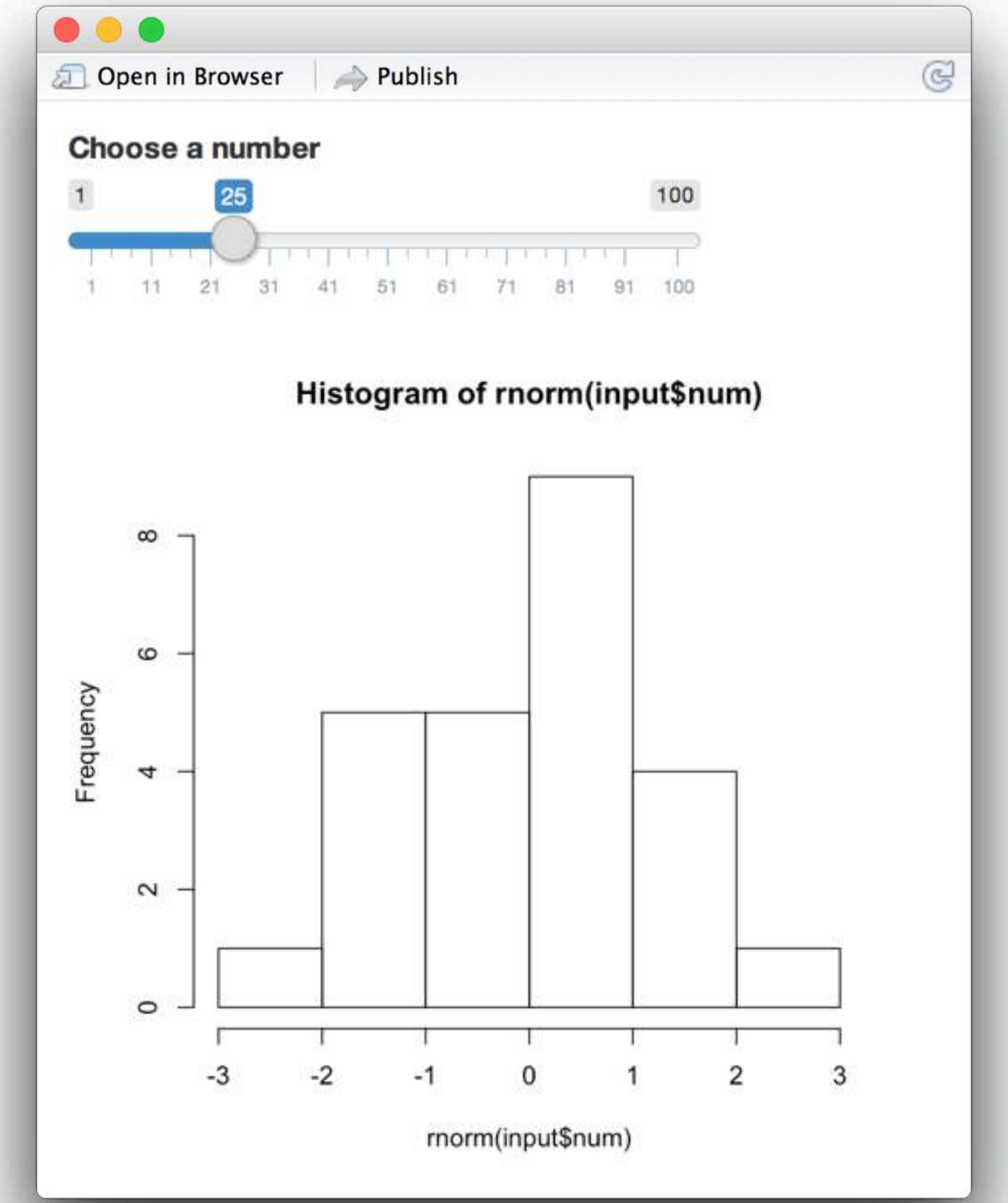
1 **Reactive values notify**
the functions that use them
when they become invalid

2 The objects created by
reactive functions respond
(different objects respond differently)

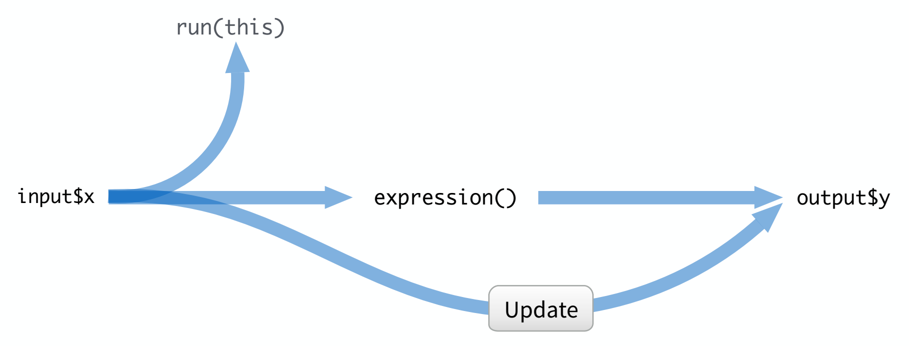


```
input$num
```

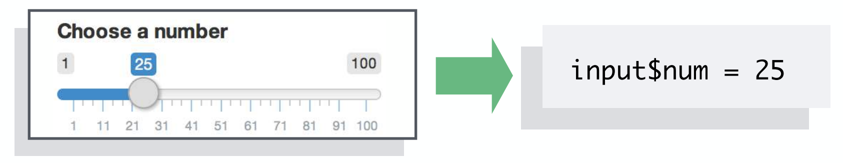
```
output$hist <- renderPlot({
  hist(rnorm(input$num))
})
```



Recap: Reactive values

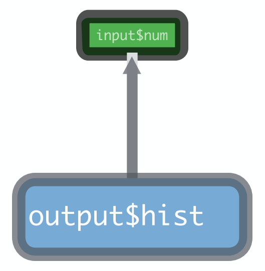


Reactive values act as the data streams that flow through your app.



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 indispensible 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 (from a data frame, matrix, or other table-like structure)
`renderImage()`	An image (saved as a link to a source file)
`renderPlot()`	A plot
`renderPrint()`	A code block of printed output
`renderTable()`	A table (from a data frame, matrix, or other table-like structure)
`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

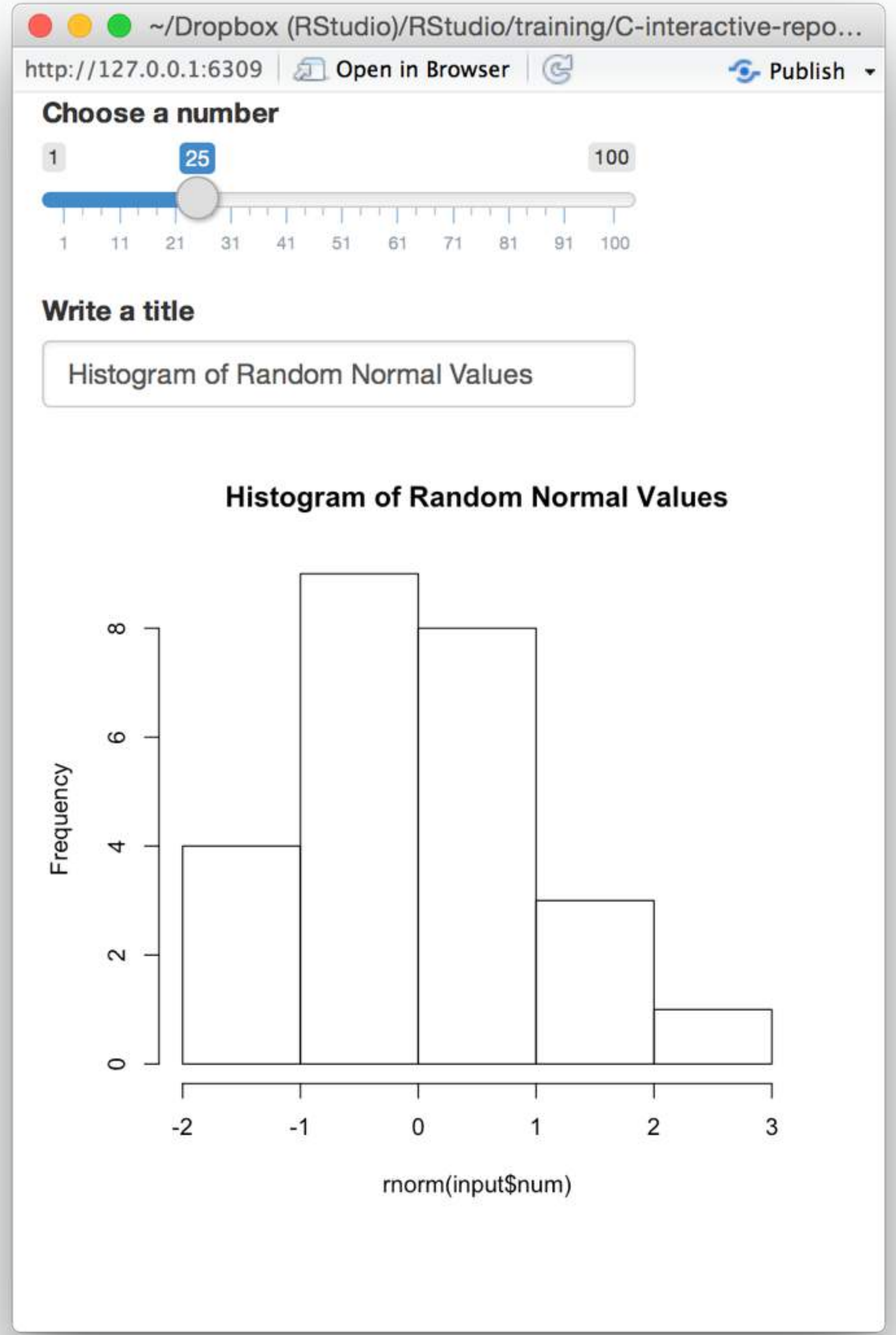
render*()

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)
```


~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...

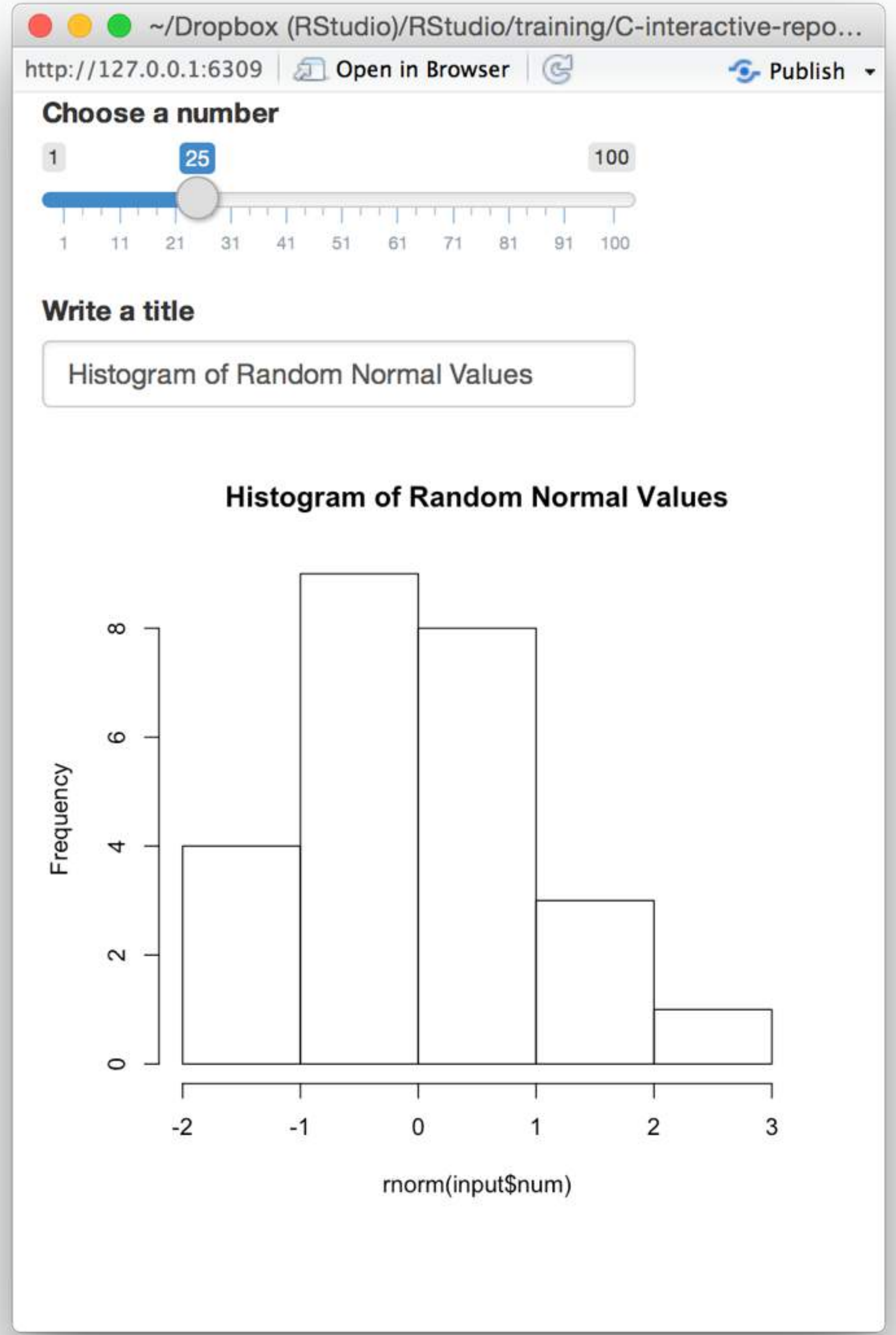
http://127.0.0.1:6309

Choose a number

Write a title

Histogram of Random Normal Values

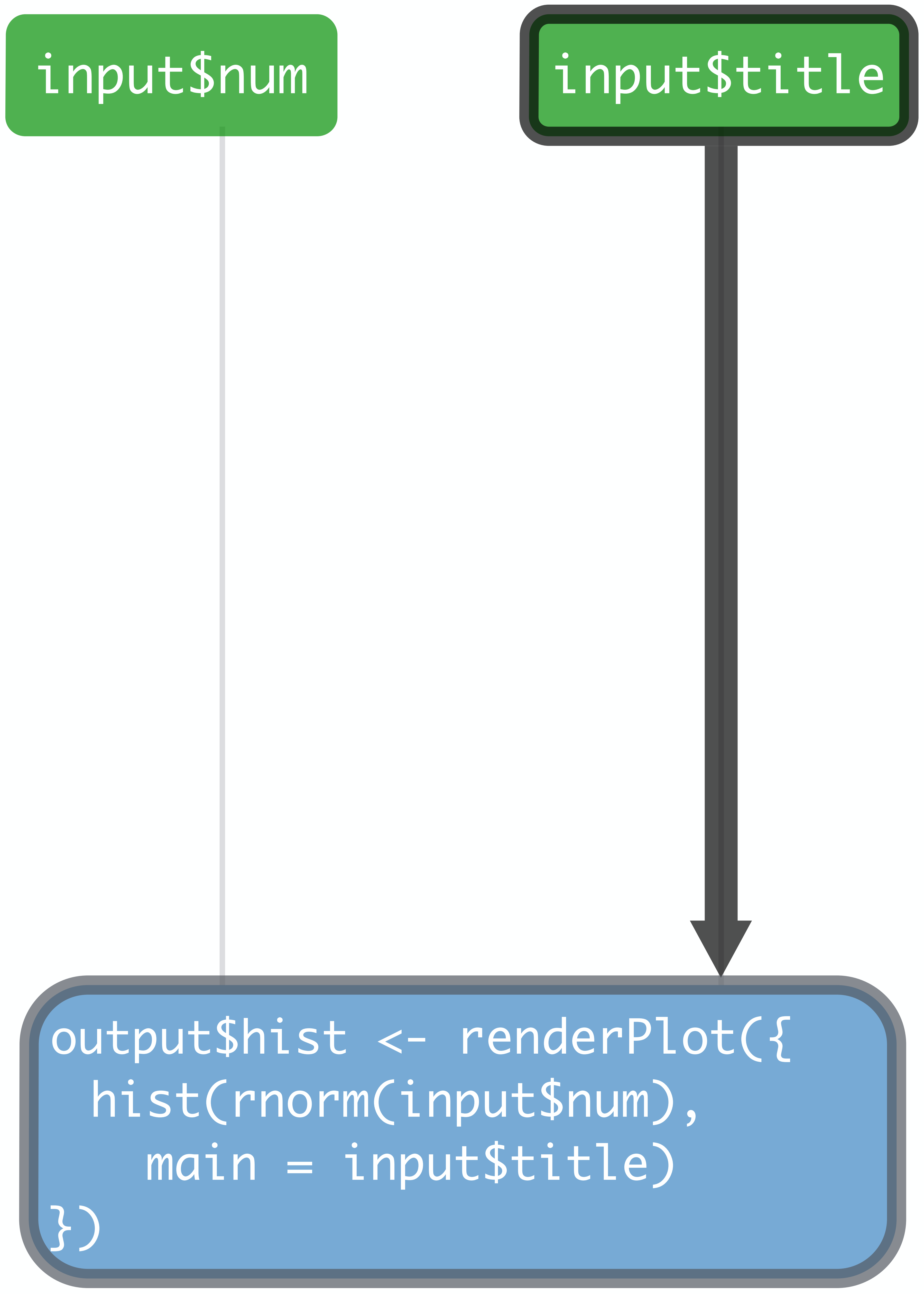
```r
# 01-two-inputs

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)
```



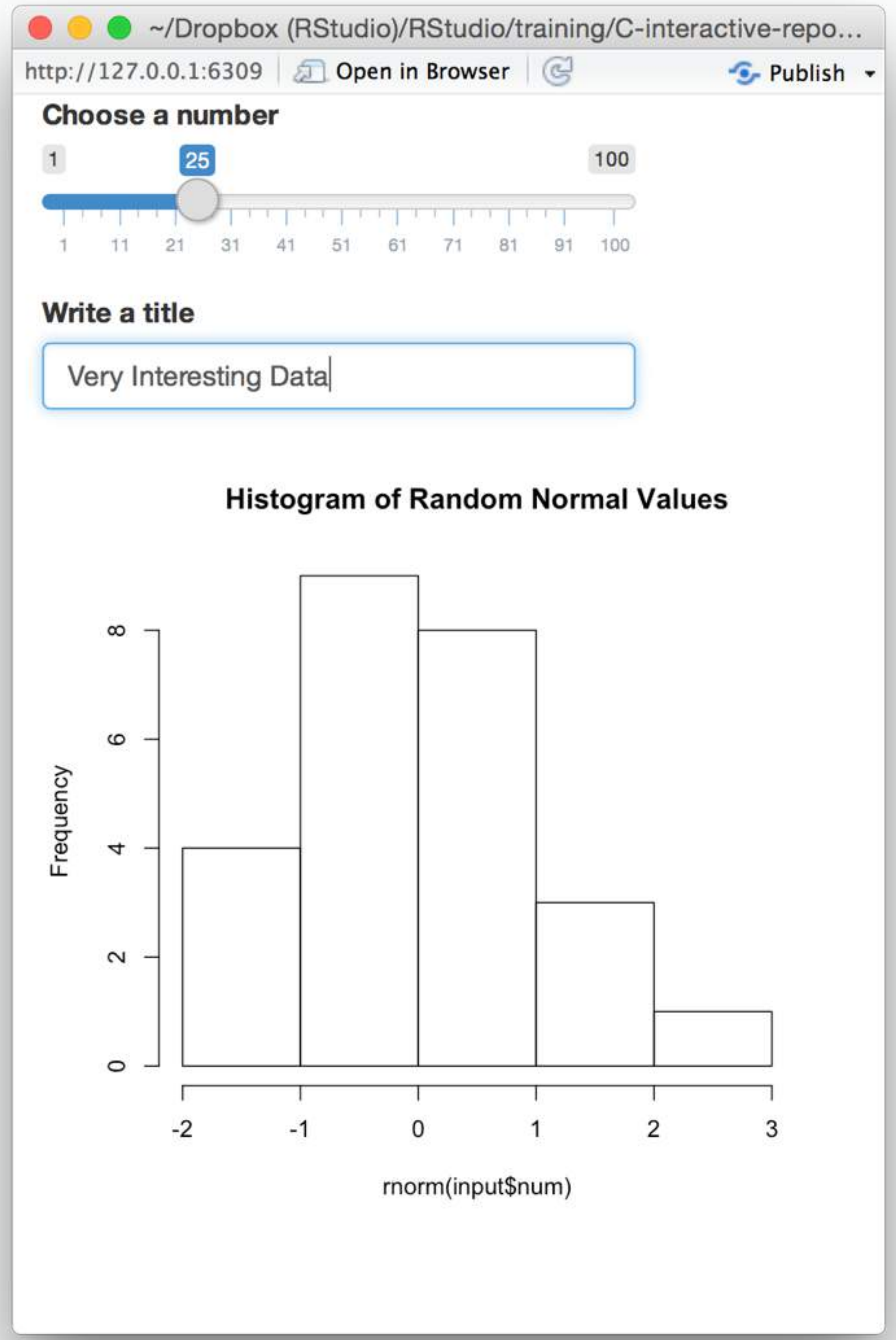
Choose a number

Write a title

Histogram of Random Normal Values



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



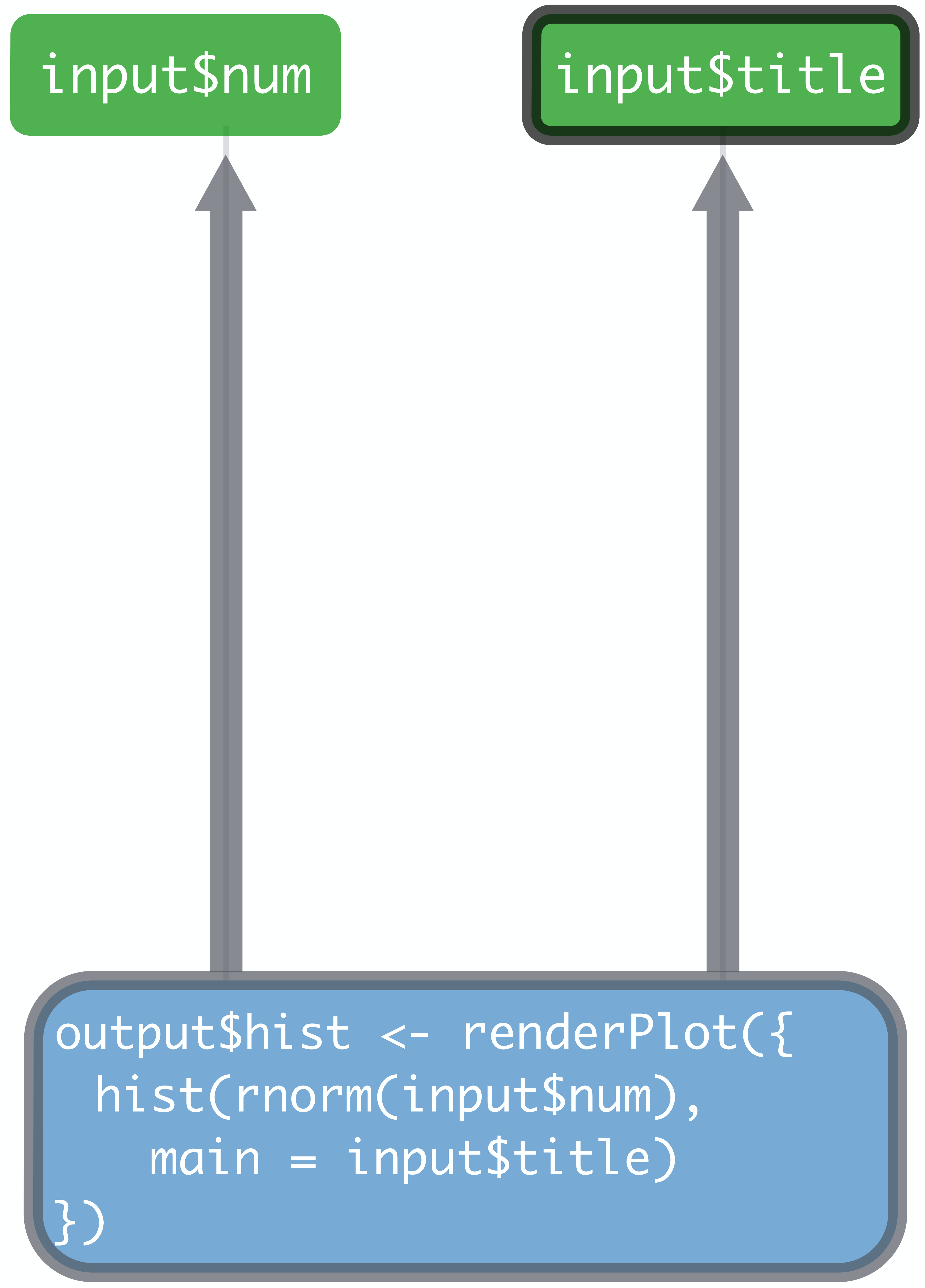
Choose a number

1 — 25 — 100

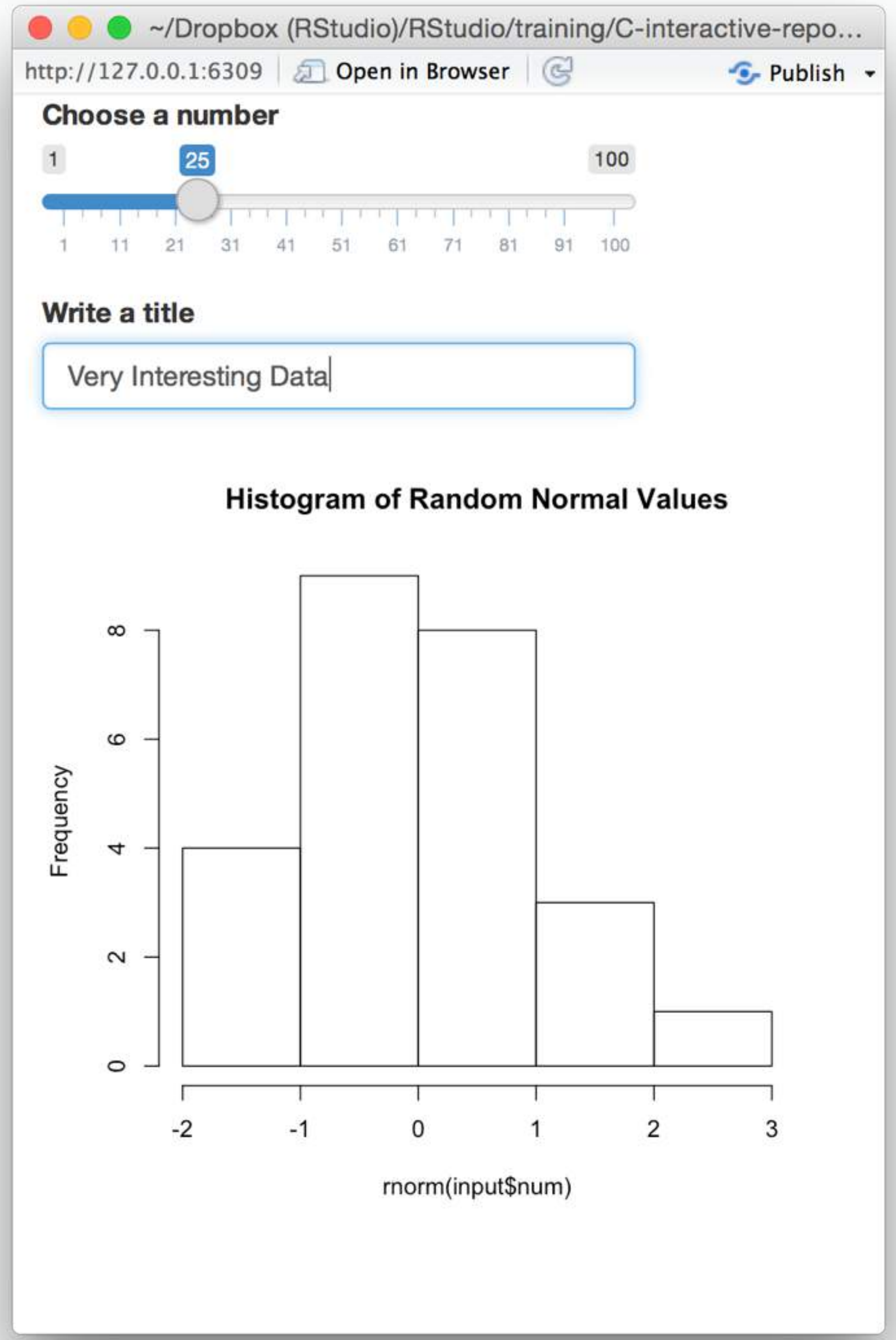
Write a title

Very Interesting Data

Histogram of Random Normal Values



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


~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish

Choose a number

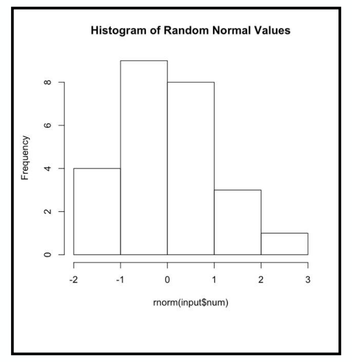
1 25 100

Write a title

Very Interesting Data

Histogram of Random Normal Values
Frequency
0
2
4
6
8
-2
-1
0
1
2
3
rnorm(input$num)

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)
})
```

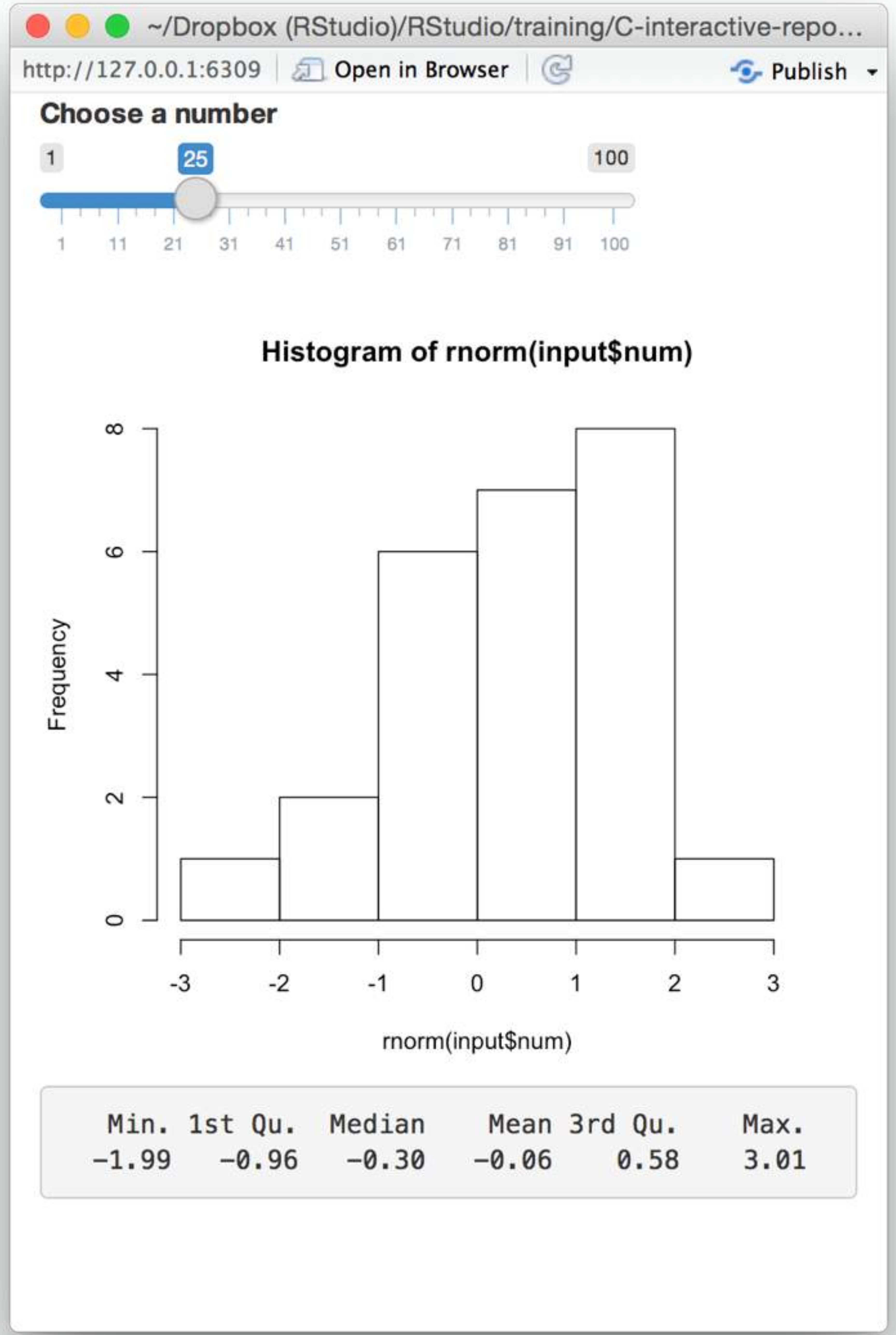
render*() makes an observer object that has a **block of code** associated with it

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

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

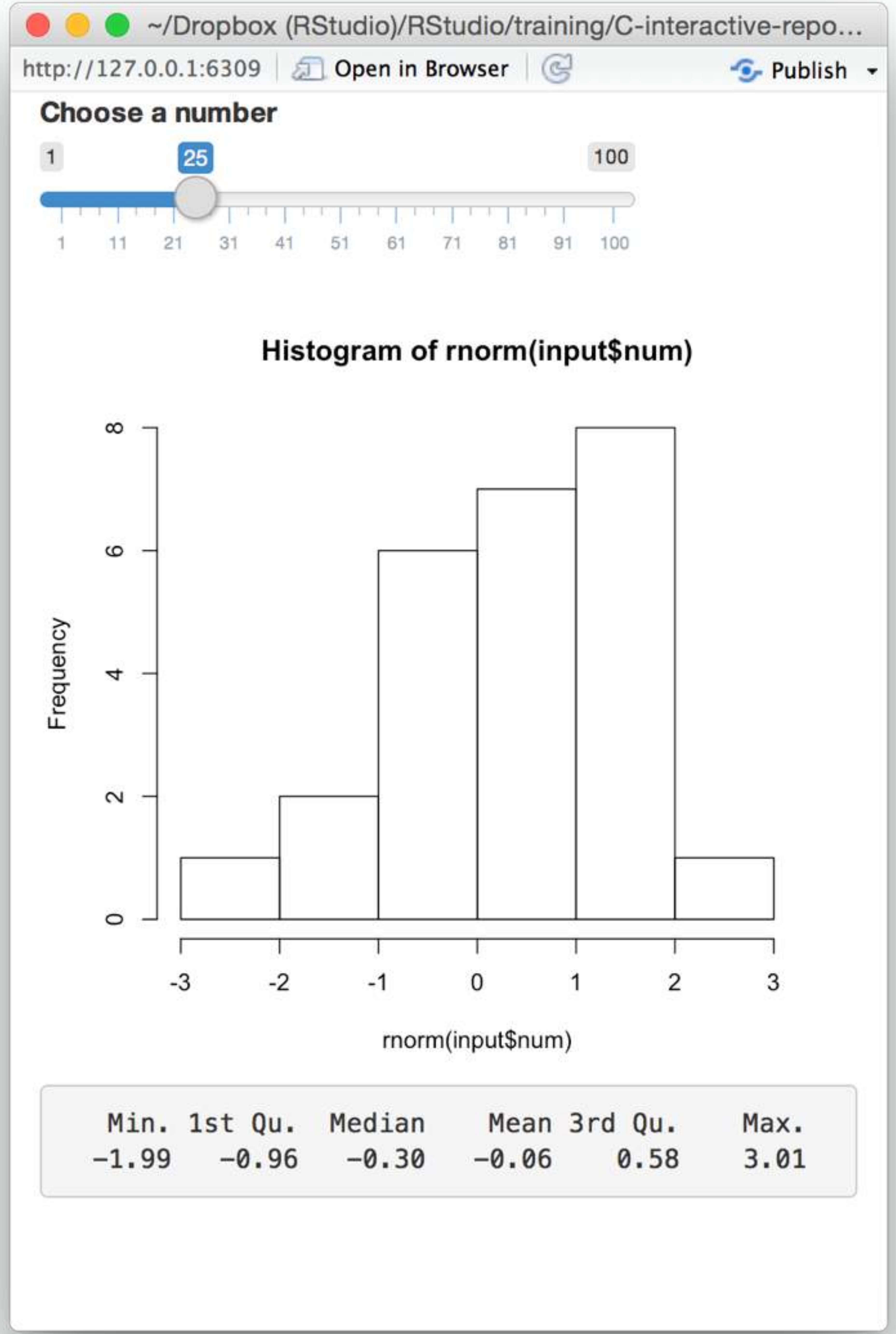
Modularize code with reactive()

```r
# 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)
```

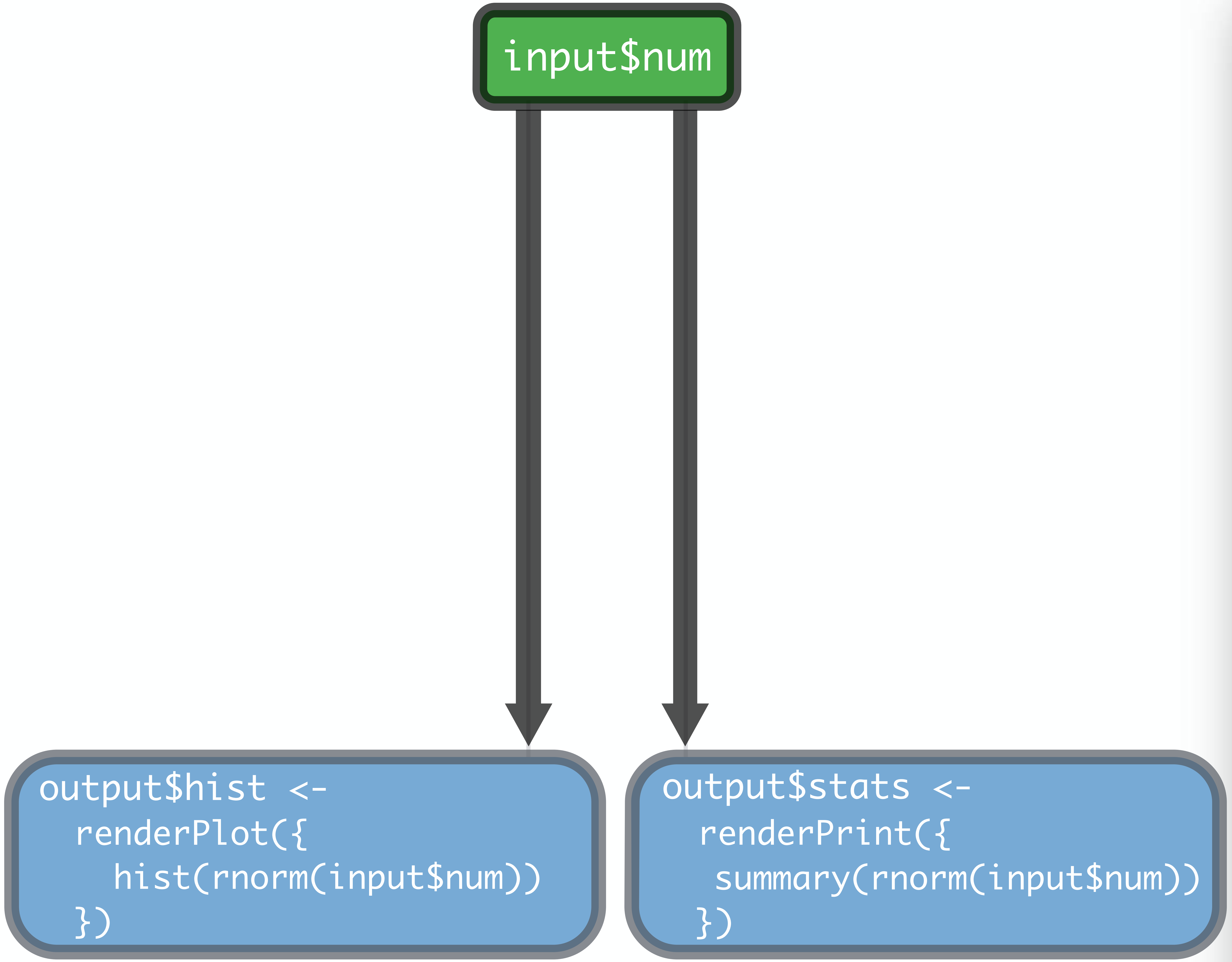


```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

```
# 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)
```

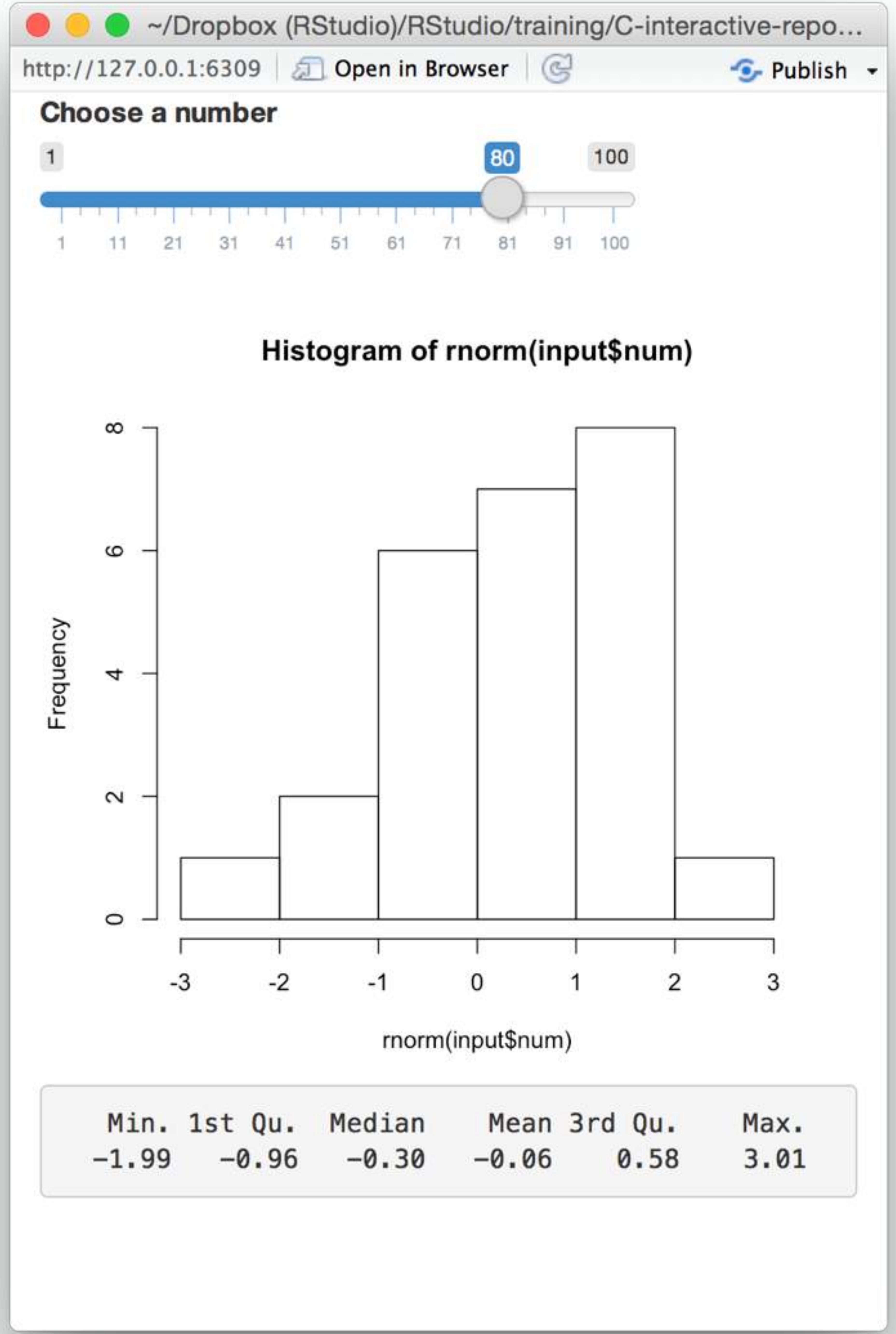


```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```



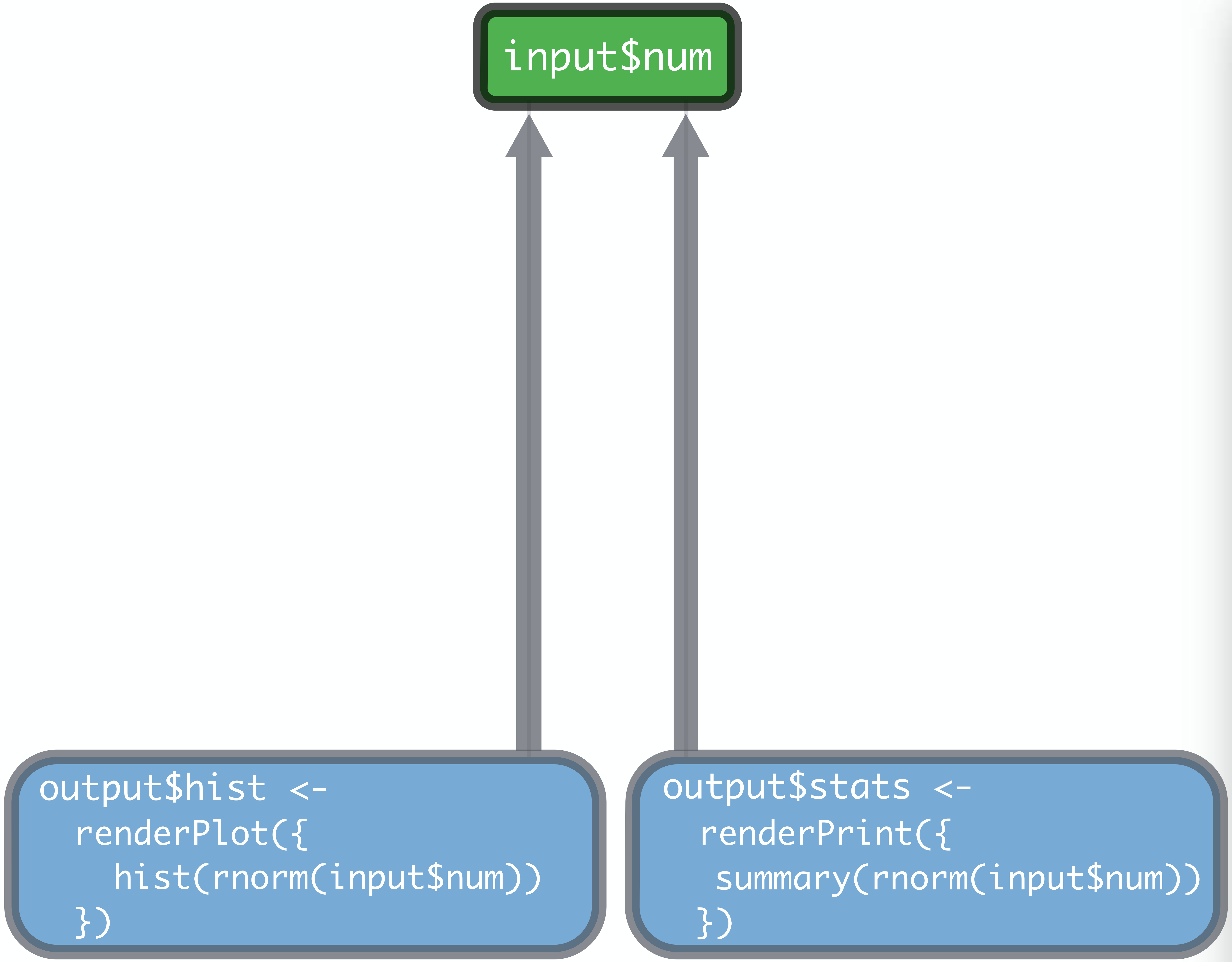
```
output$hist <-
  renderPlot({
    hist(rnorm(input$num))
  })
```

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



Choose a number

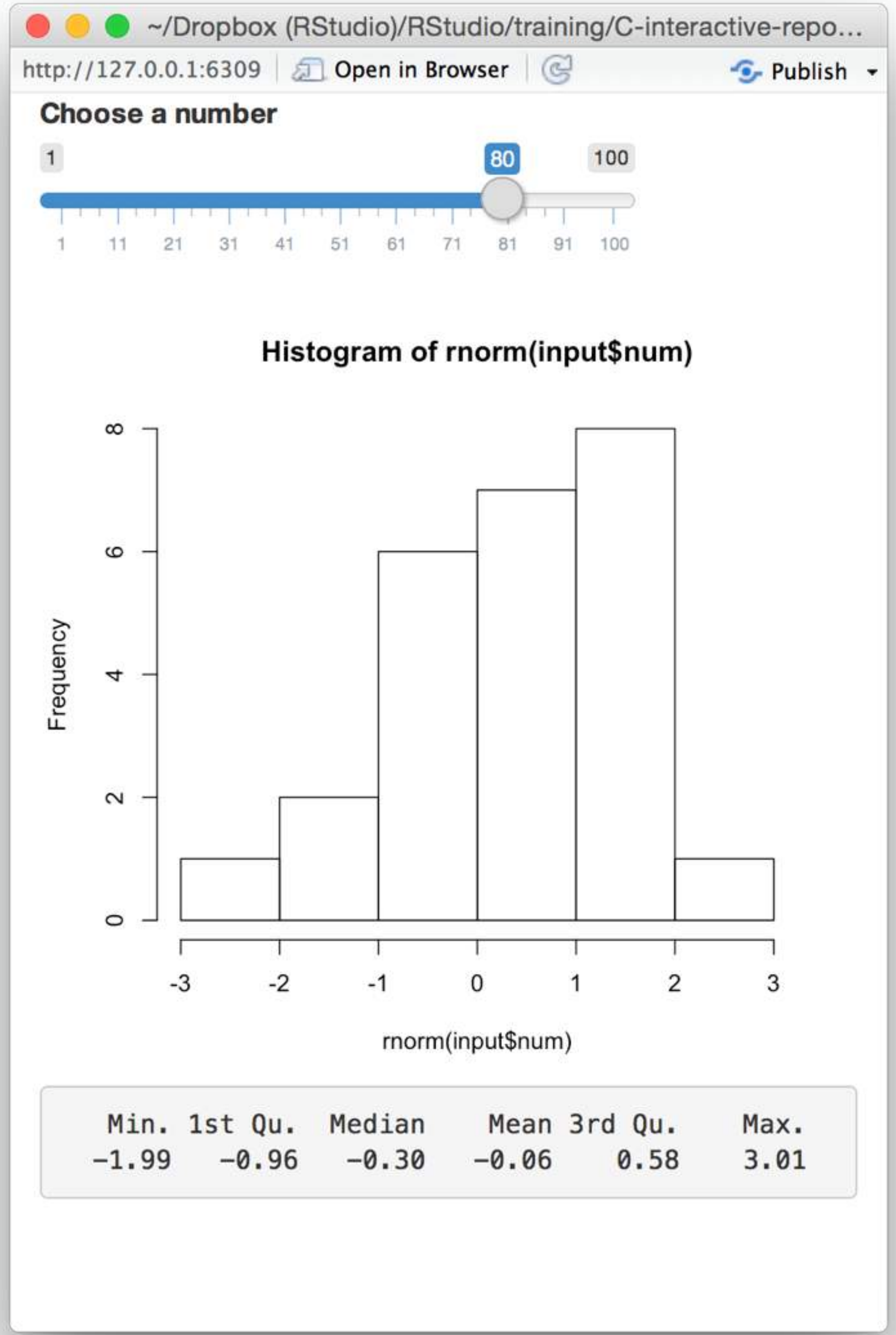
```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 -1.99   -0.96   -0.30   -0.06    0.58    3.01 
```



```
input$num
```

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

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



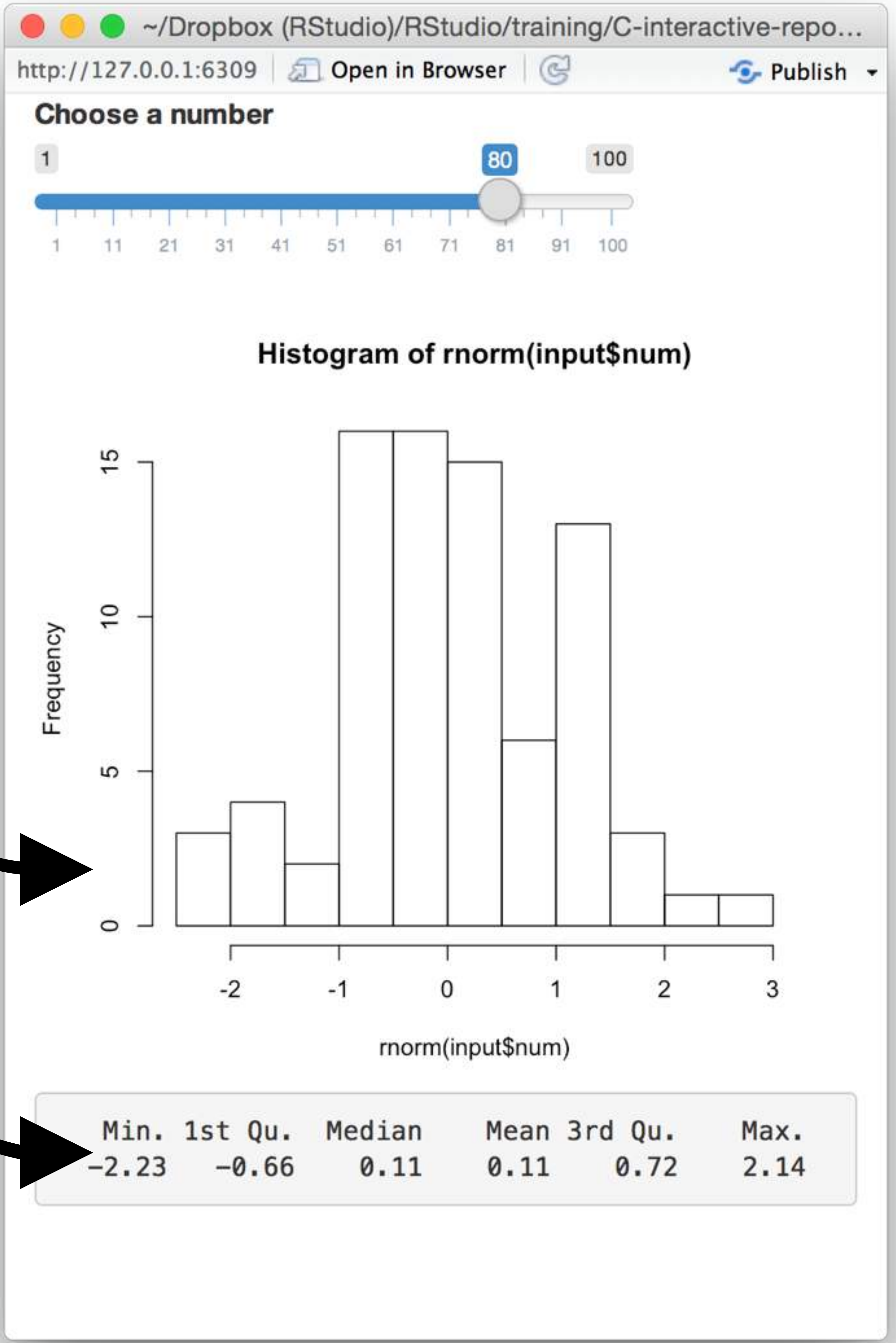
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

```
input$num
```

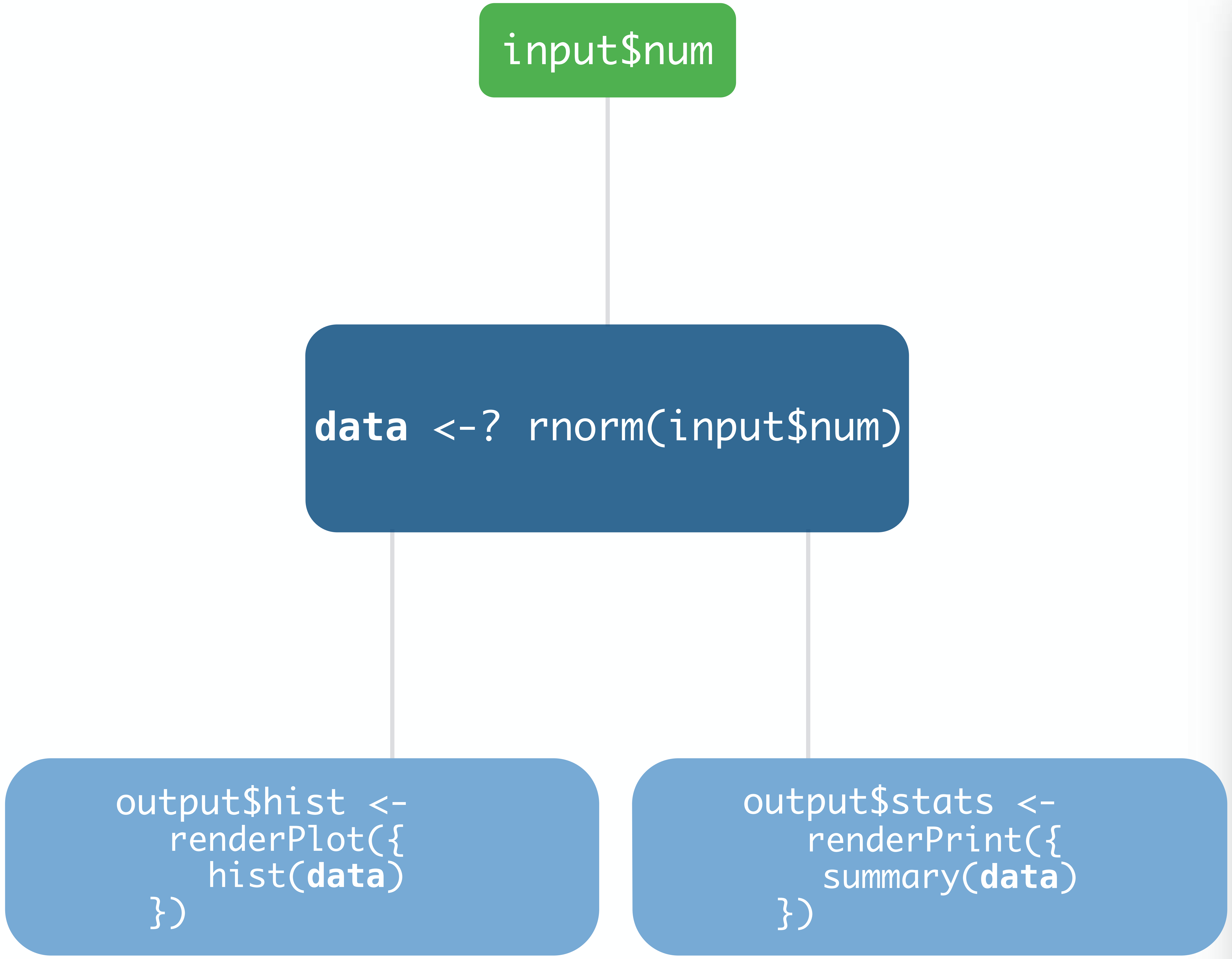
Can these describe the same data?

```
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
```

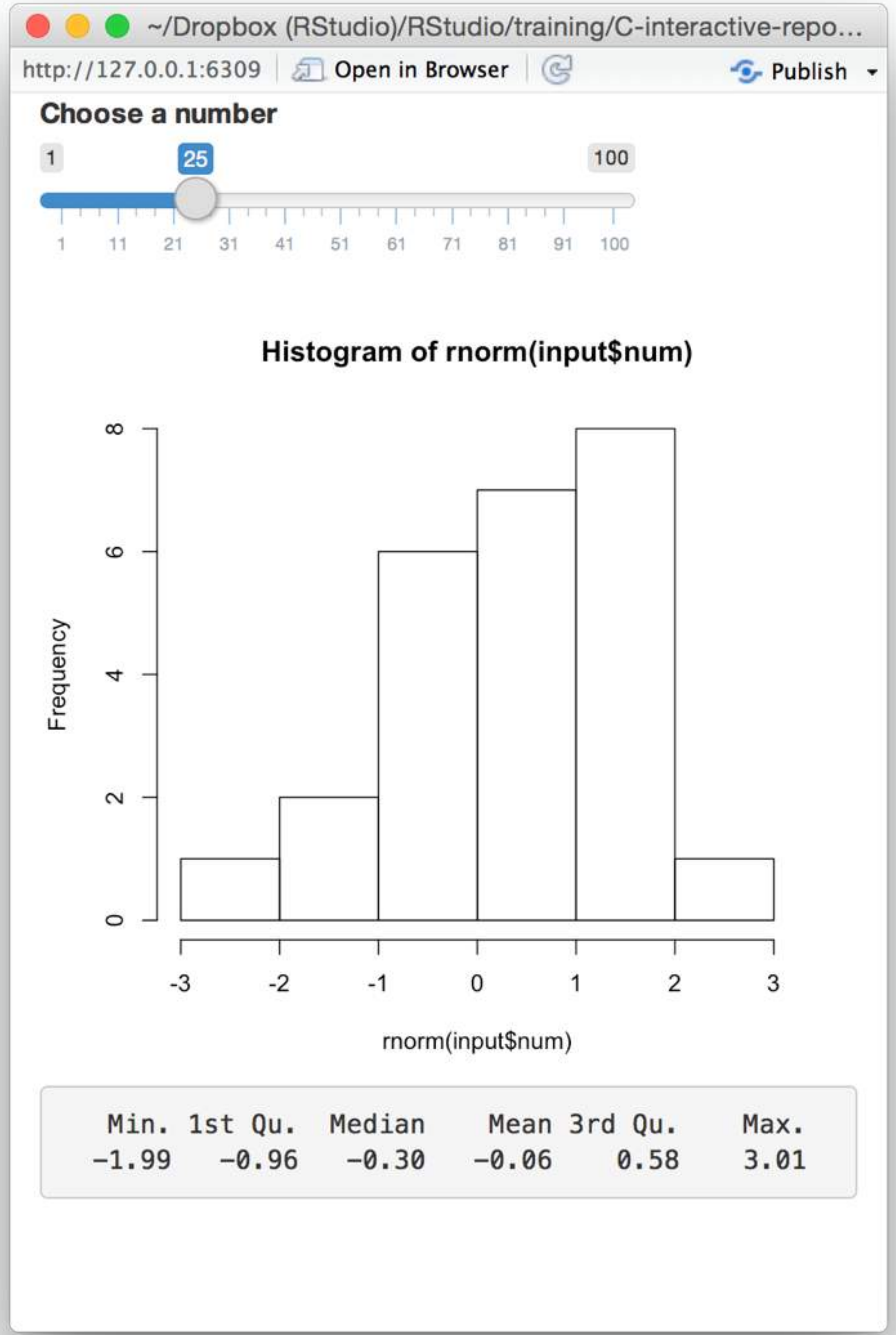


```
input$num
```

```
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*

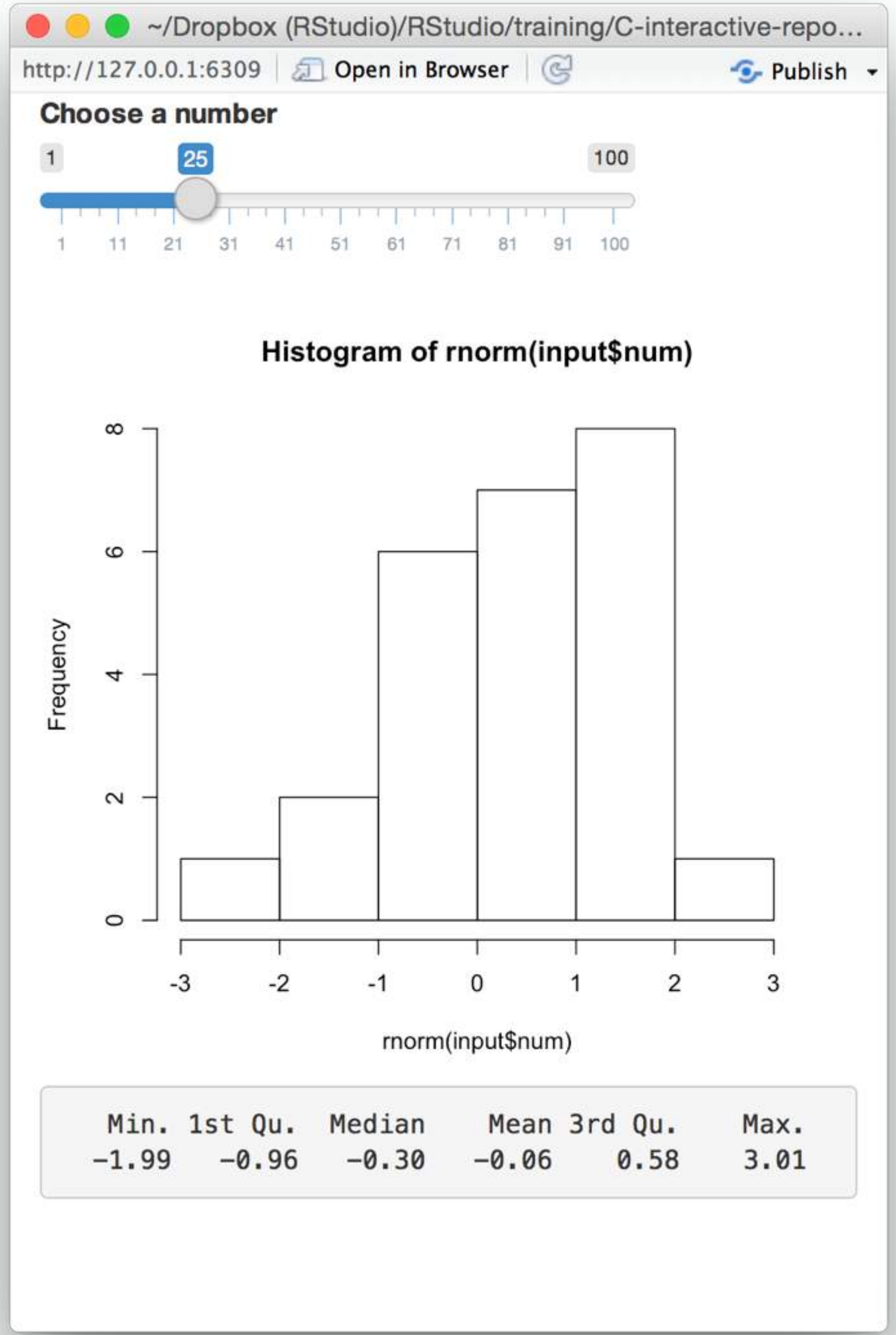
code used to build (and rebuild) object

A reactive expression is special in two ways

```
data()
```

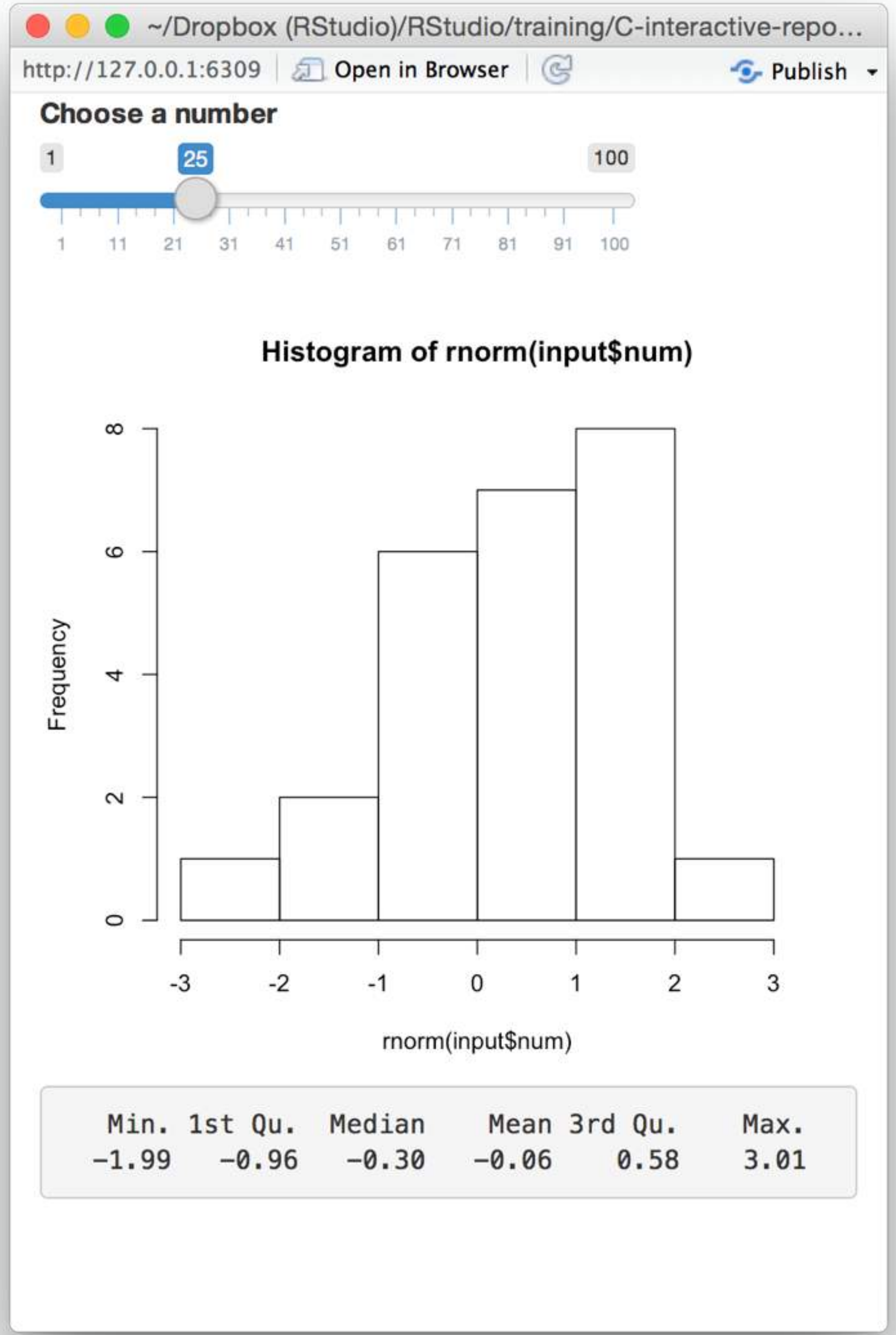
1 You call a reactive expression like a function

```r
# 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)
```



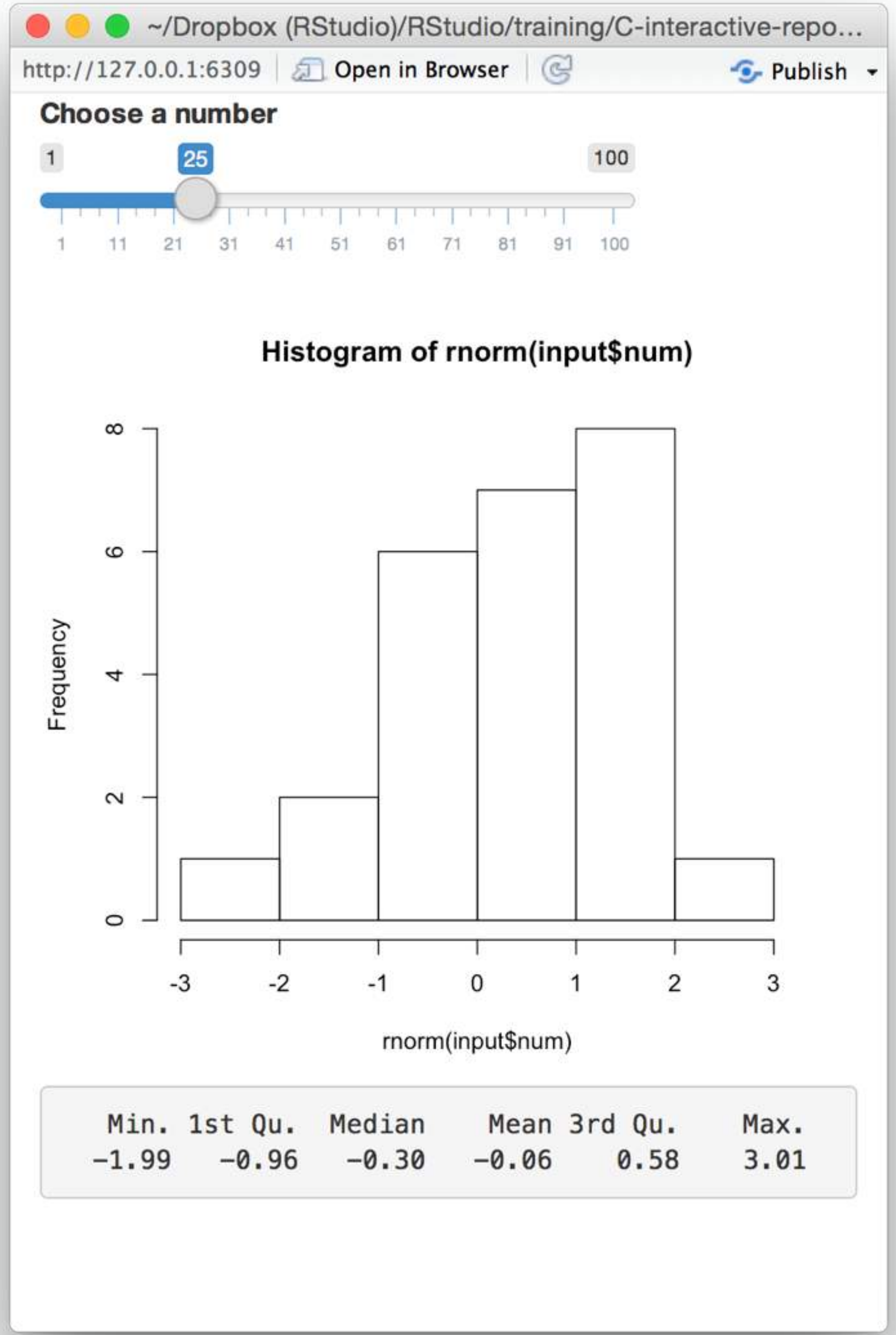
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

```
# 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)
```

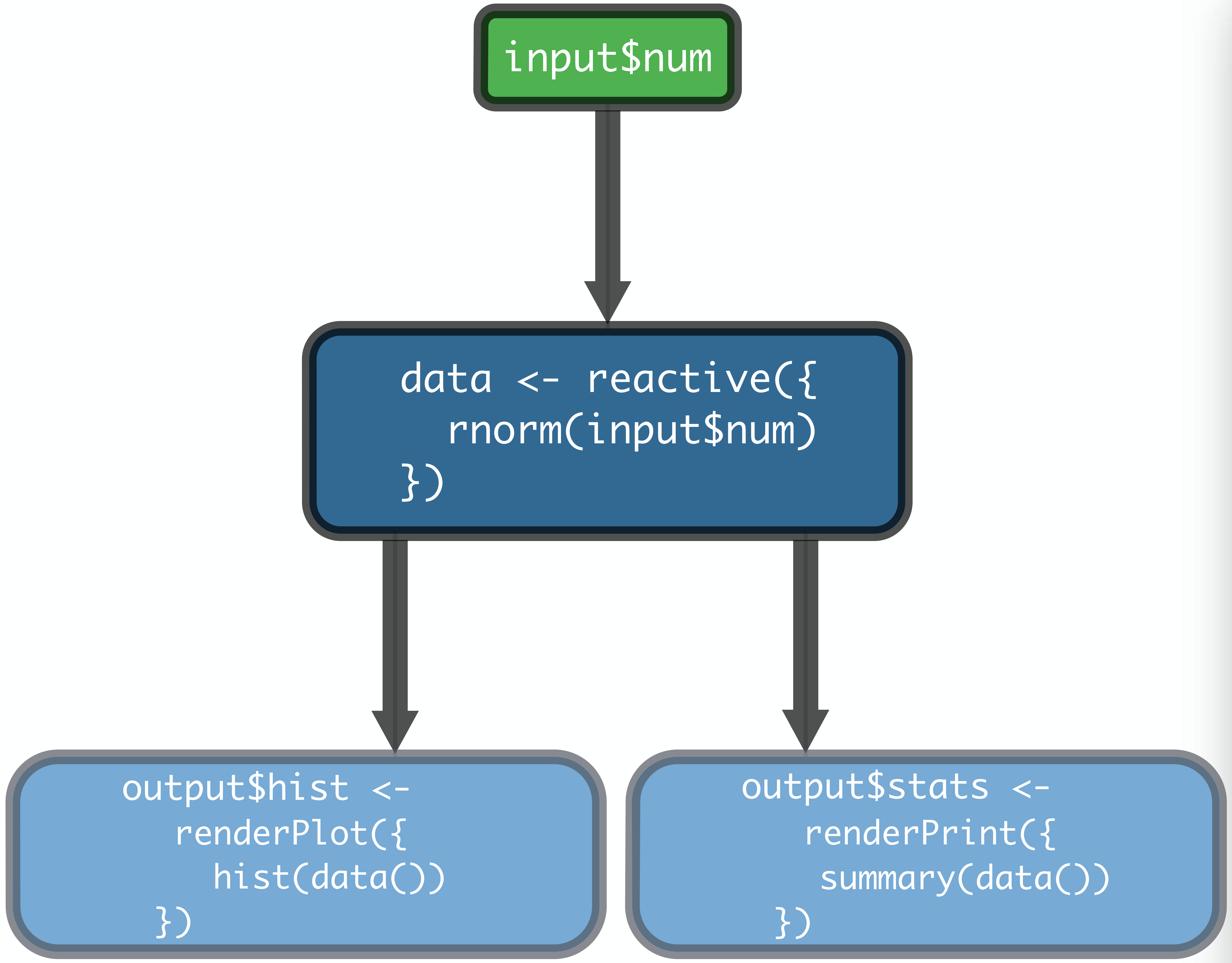


```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.99   -0.96   -0.30   -0.06    0.58    3.01
```

```r
# 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)
```



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

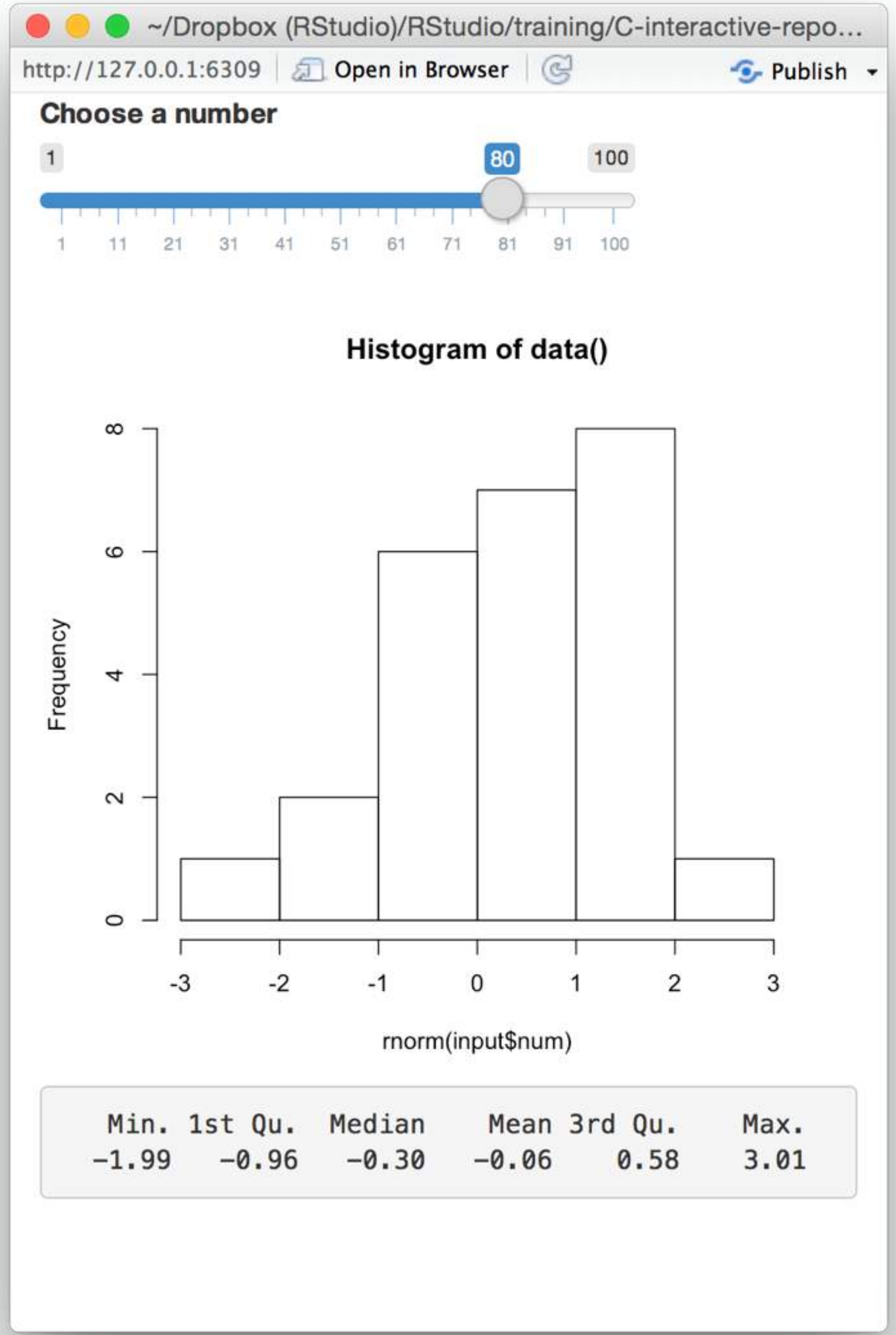


```
input$num
```

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

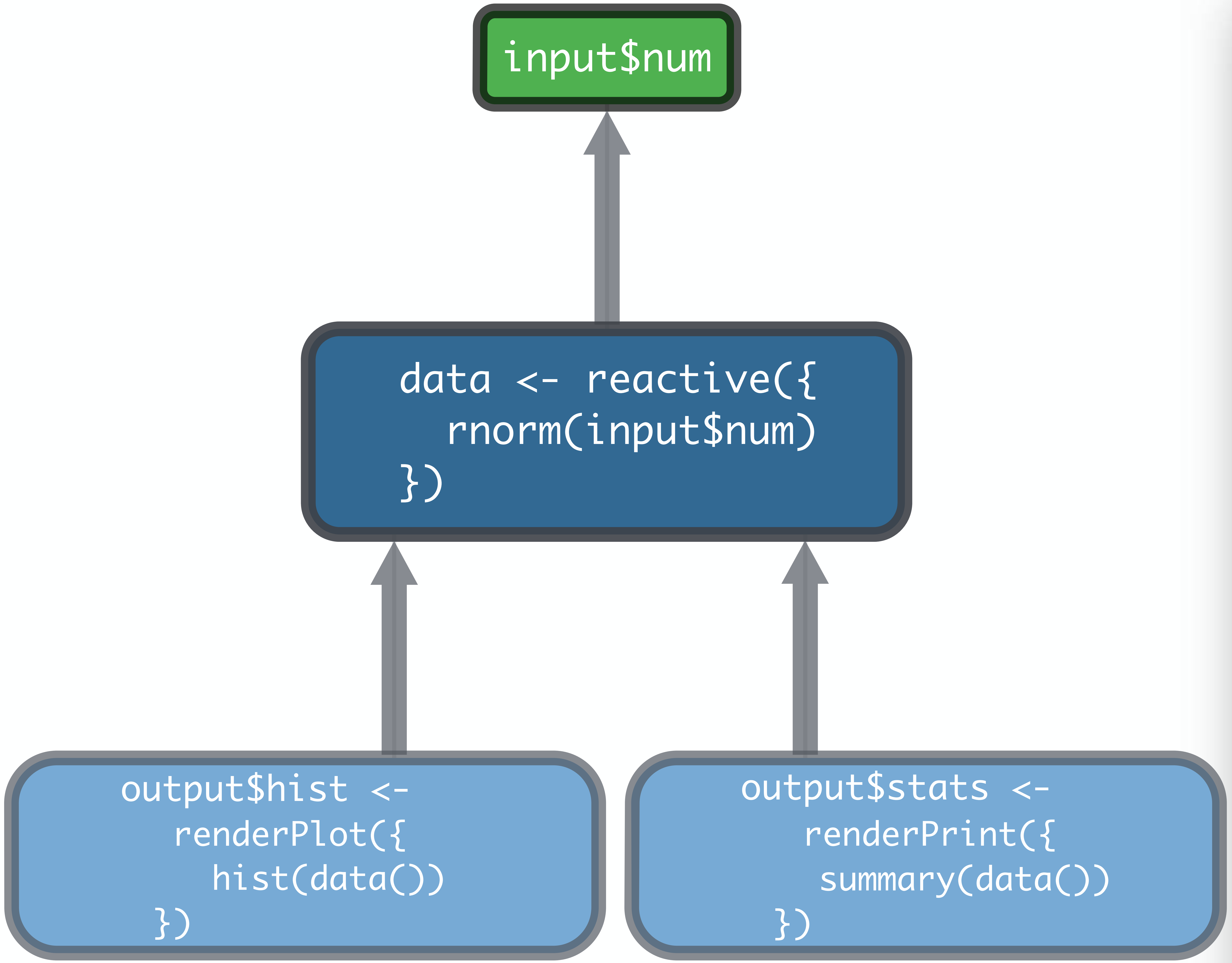
```
output$hist <-
  renderPlot({
    hist(data())
})
```

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



Choose a number

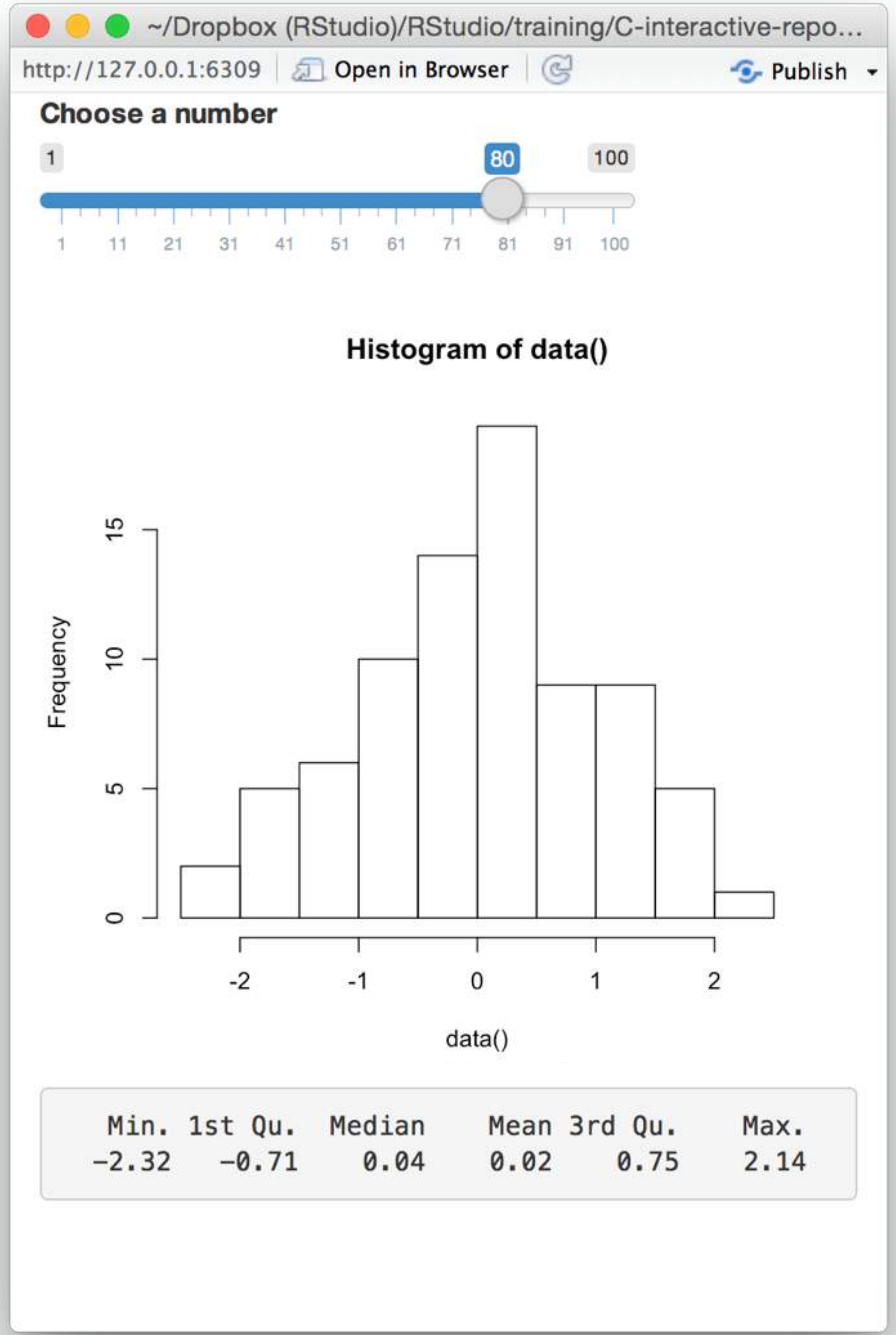
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  -1.99   -0.96   -0.30   -0.06    0.58    3.01 
```



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

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

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



```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -2.32   -0.71    0.04    0.02    0.75    2.14
```

A reactive expression is special in two ways

```
data()
```

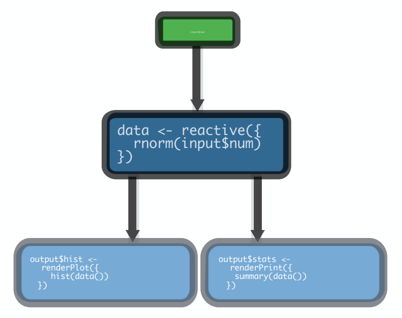
1 You call a reactive expression like a function

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**

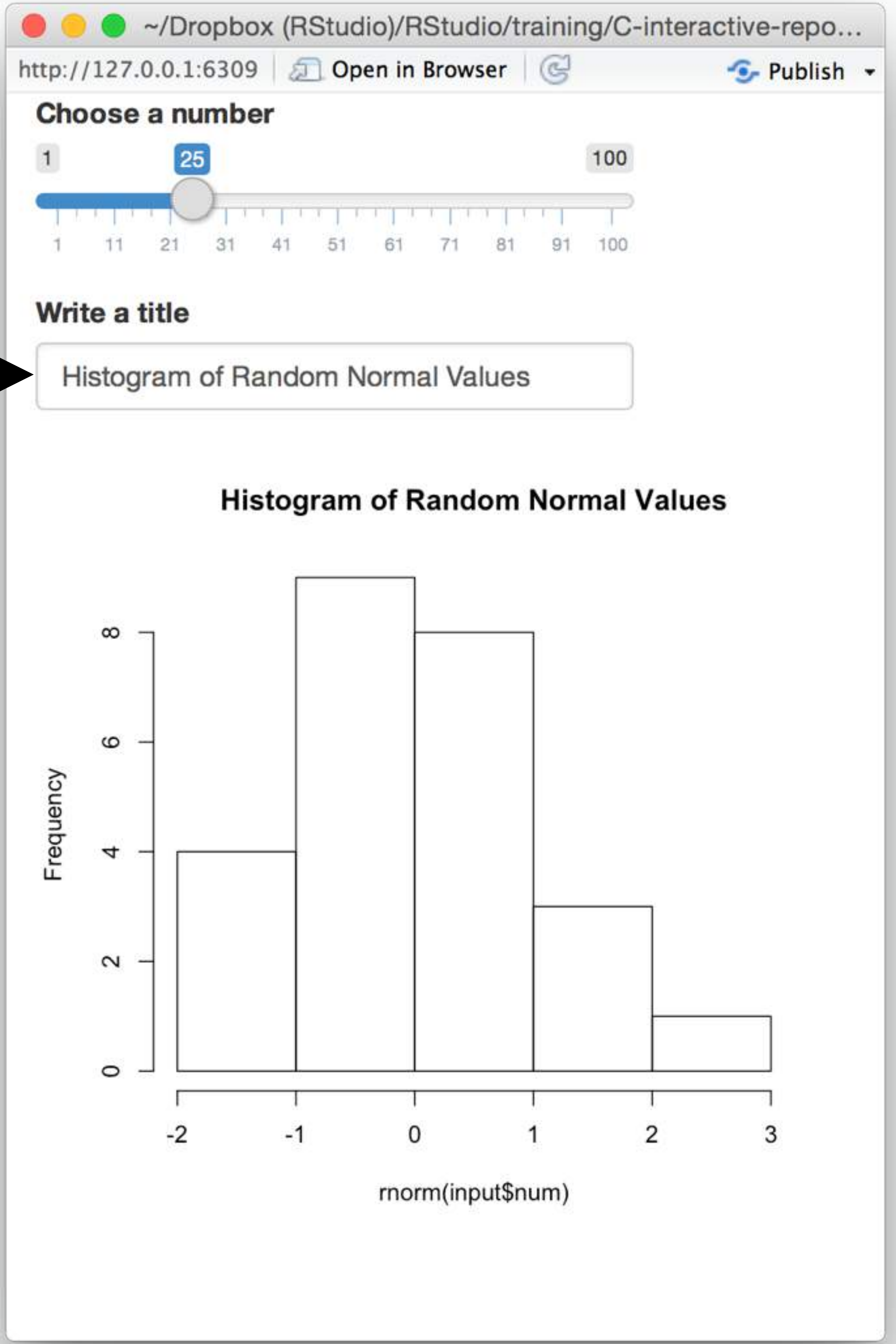
2

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

Prevent reactions with isolate()

```r
# 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)
```

Can we prevent the title field from updating the plot?



Choose a number

Write a title

Histogram of Random Normal Values

Histogram of Random Normal Values

isolate()

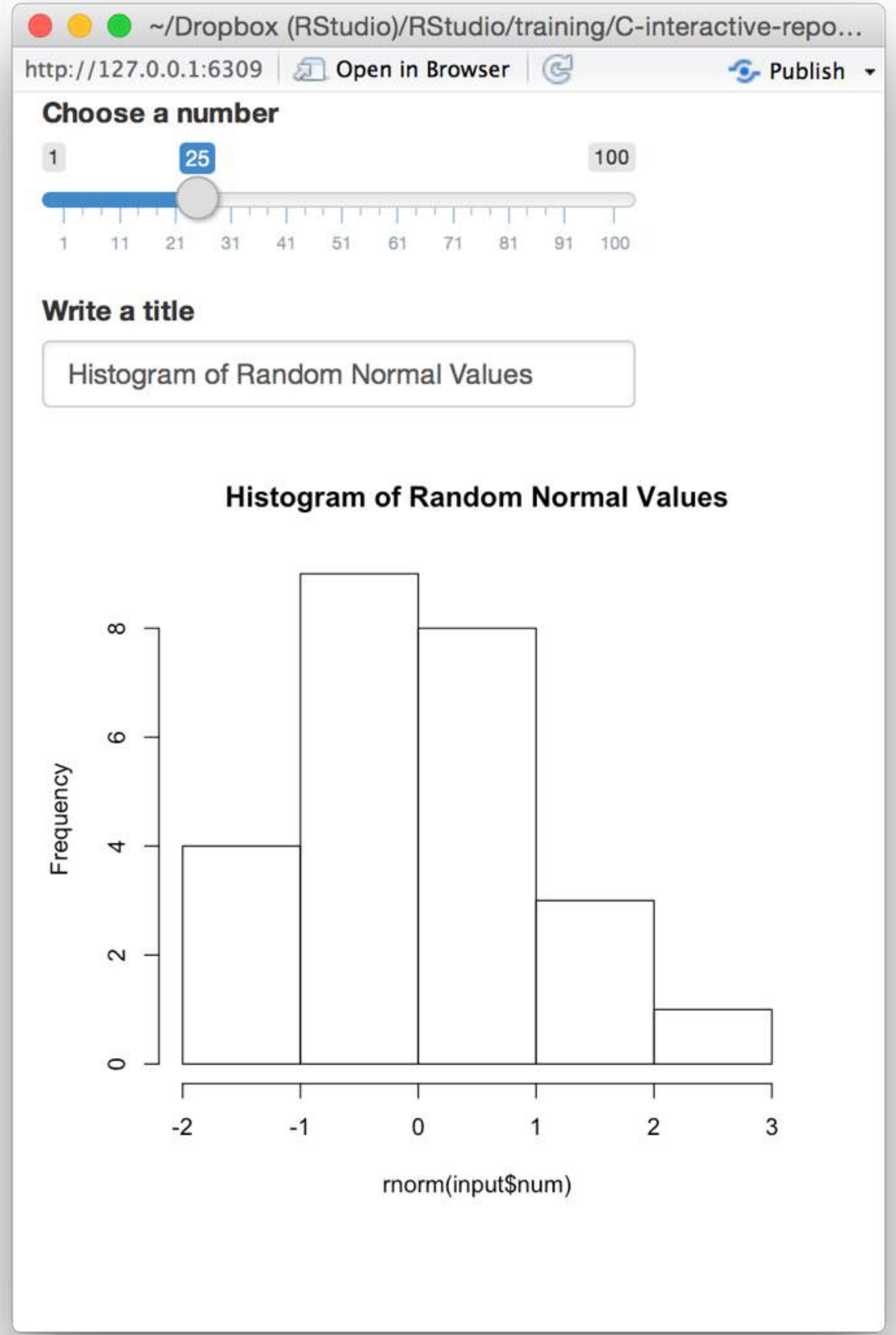
Returns the result as a non-reactive value

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

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

code used to build object

```r
# 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)
```


~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish

Choose a number

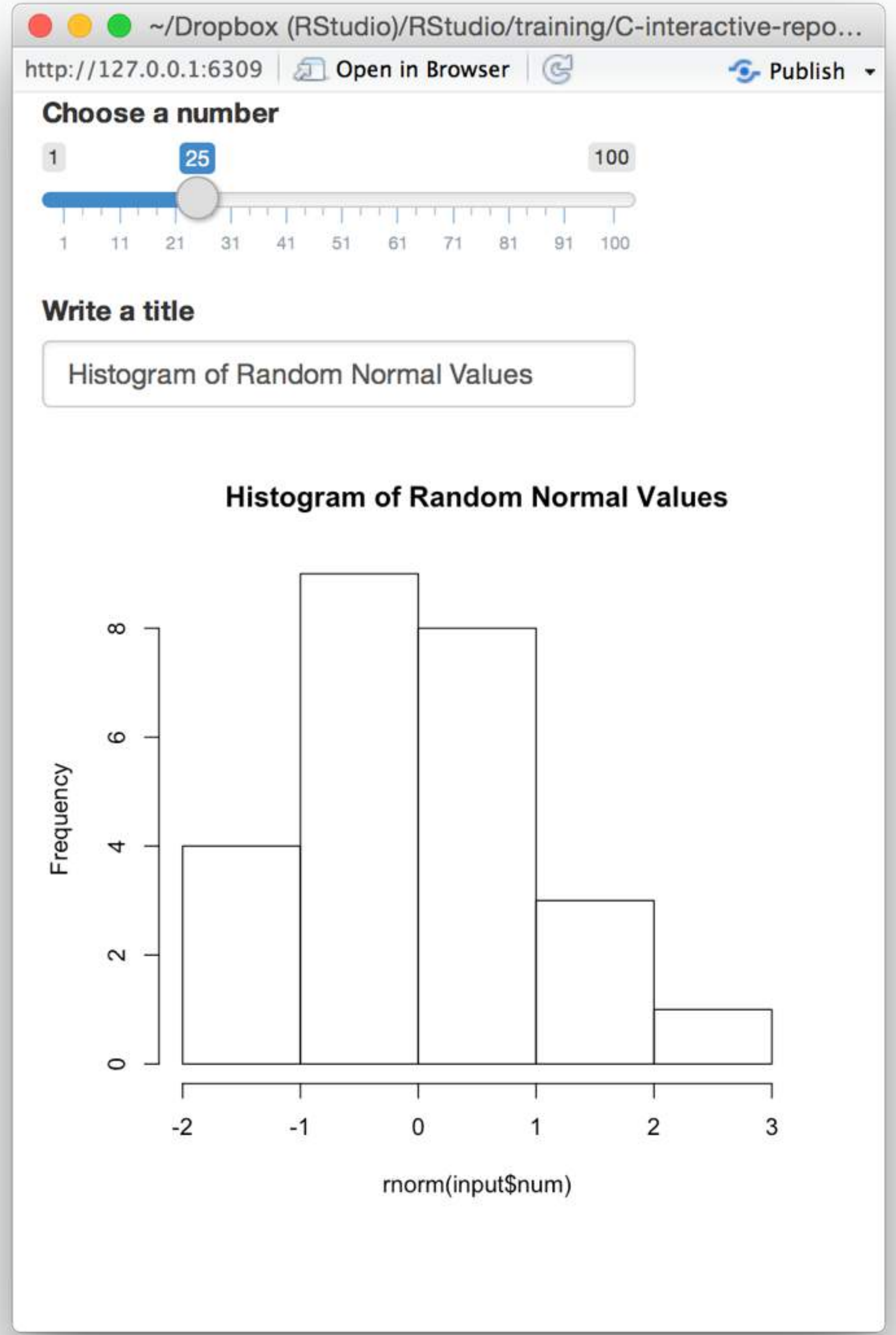
1 25 100

Write a title

Histogram of Random Normal Values

Histogram of Random Normal Values
Frequency
rnorm(input$num)

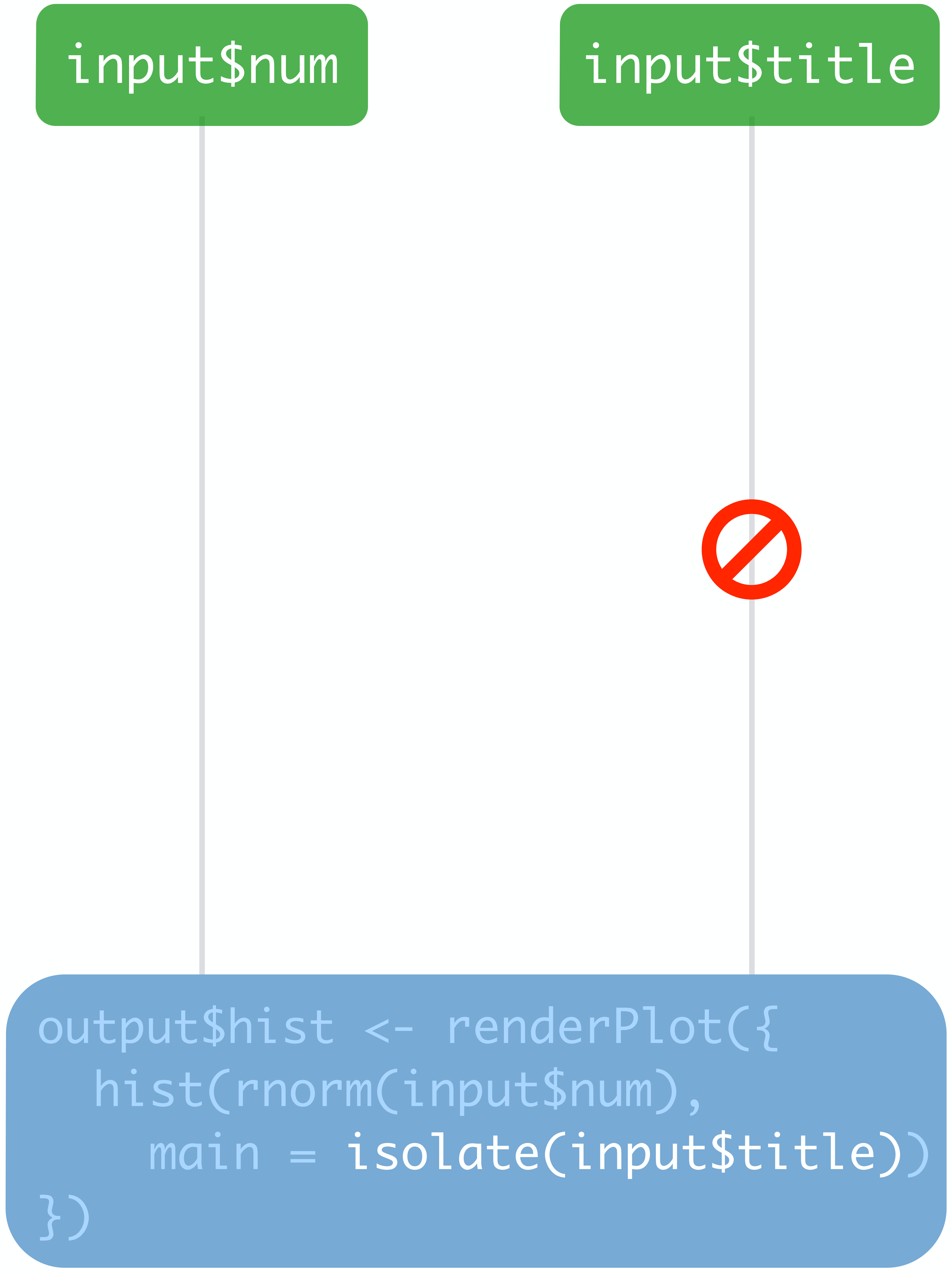
```r
# 04-isolate

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)
```


~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...

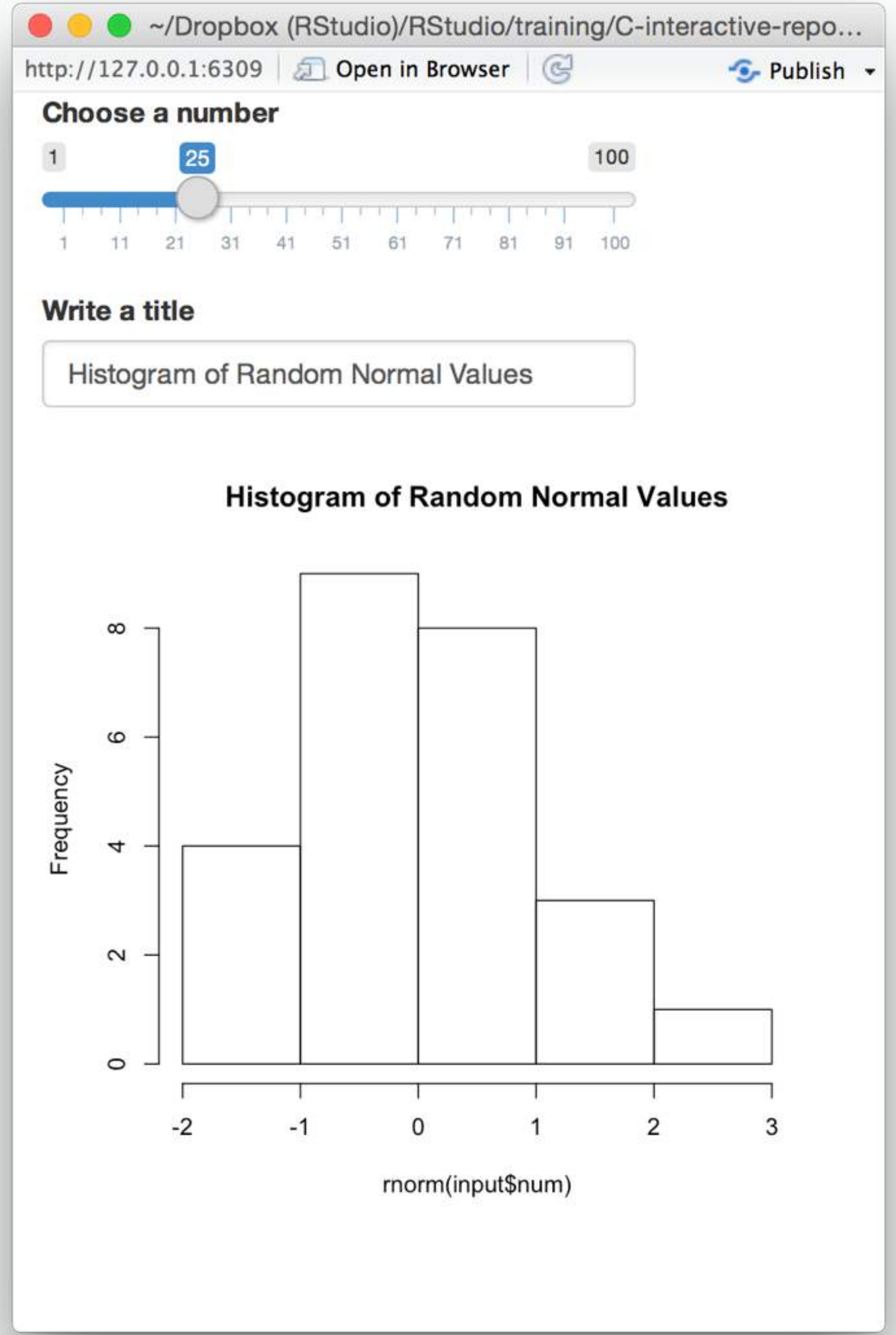
http://127.0.0.1:6309

Choose a number: 25

Write a title: Histogram of Random Normal Values

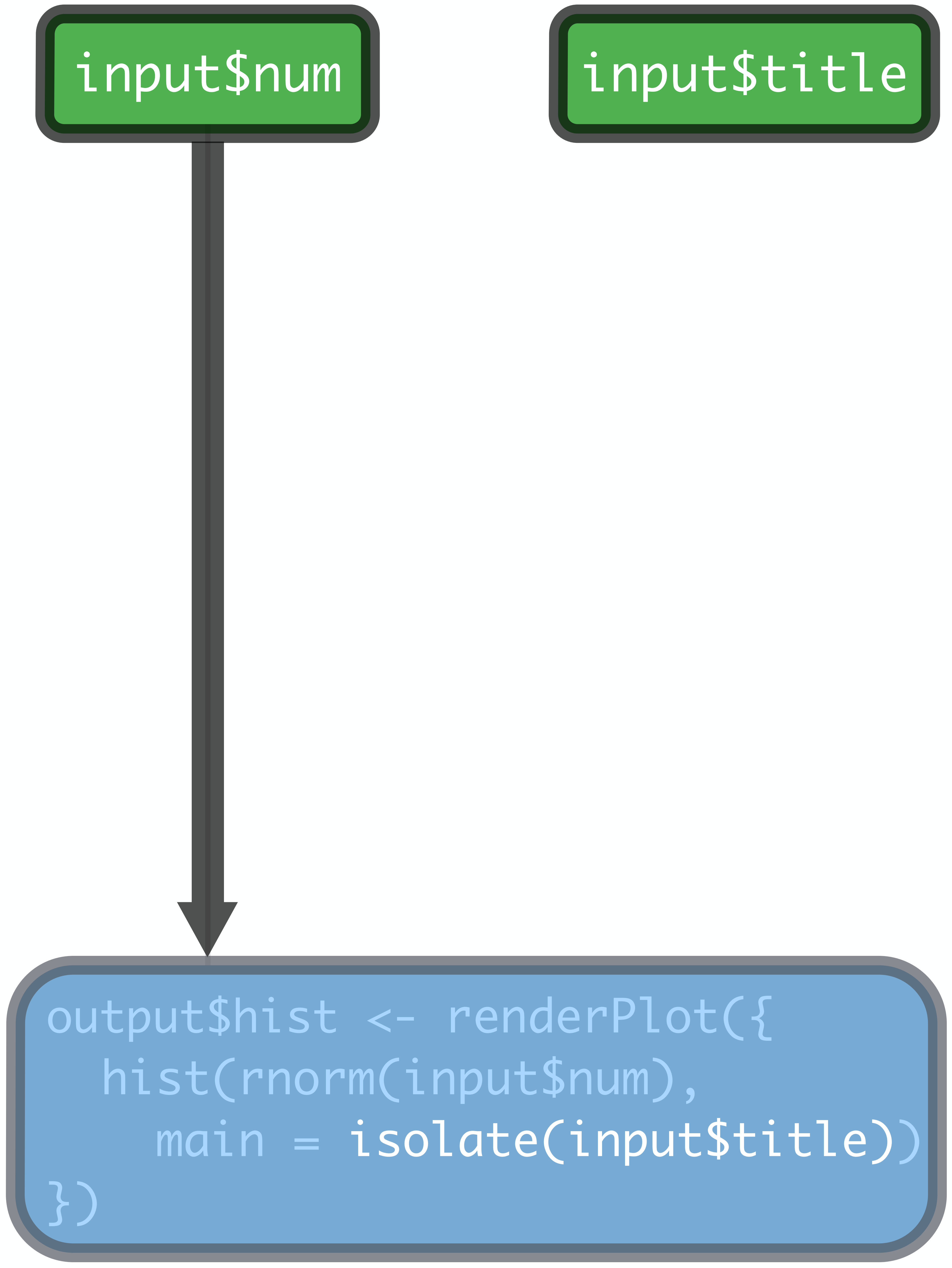


```
output$hist <- renderPlot({
  hist(rnorm(input$num),
    main = isolate(input$title))
})
```

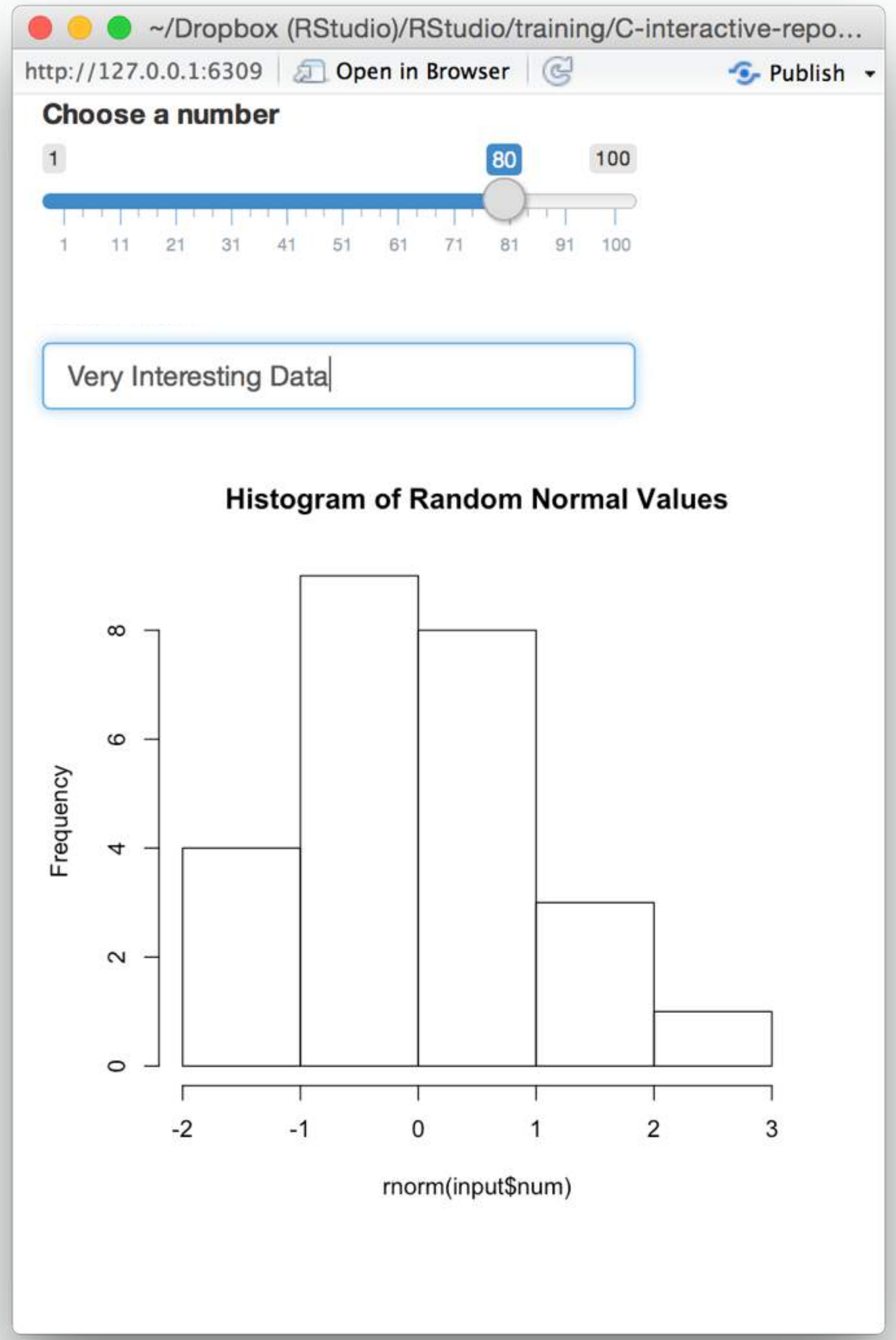


Choose a number: 25

Write a title: Histogram of Random Normal Values



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

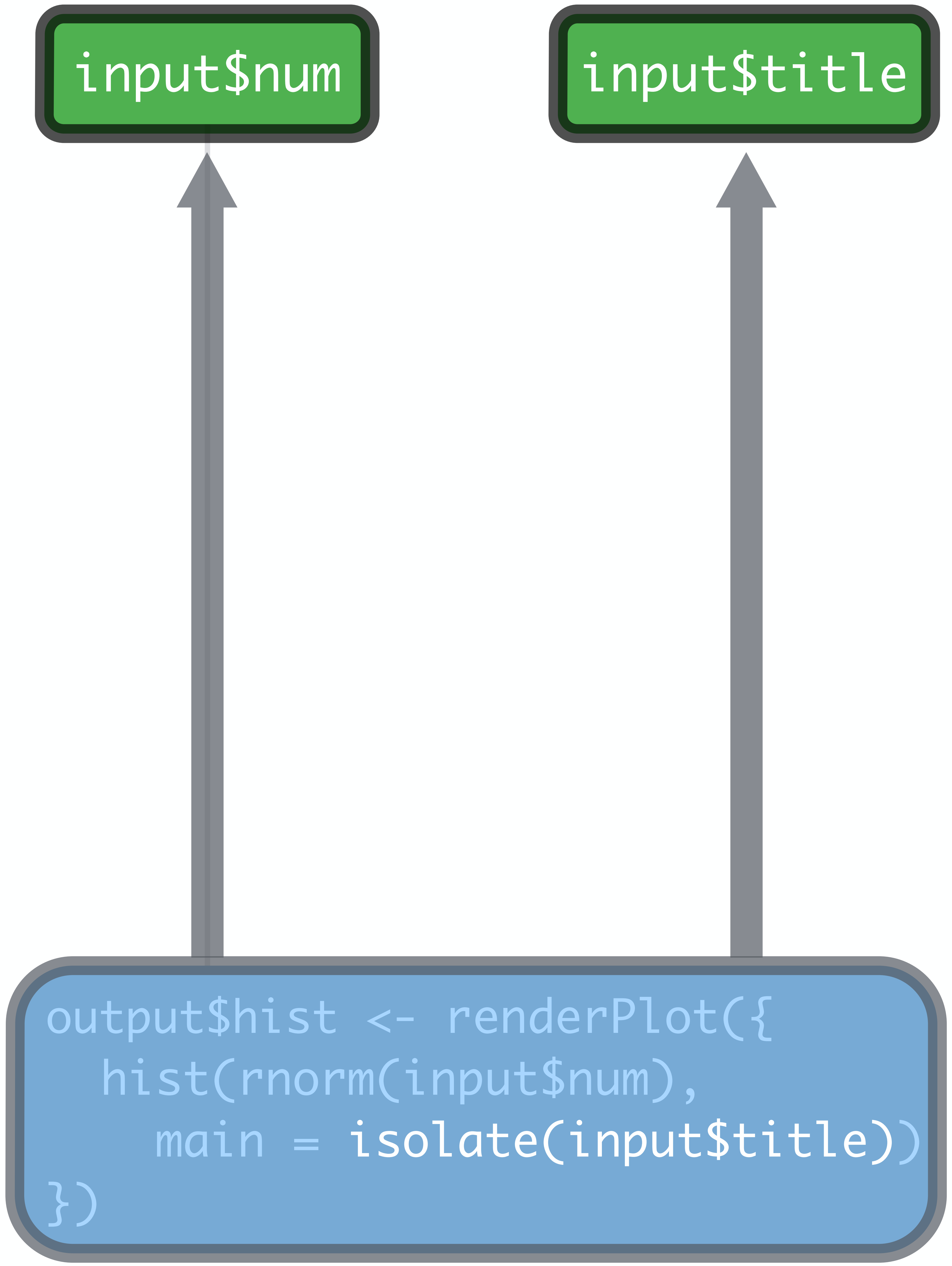


Choose a number

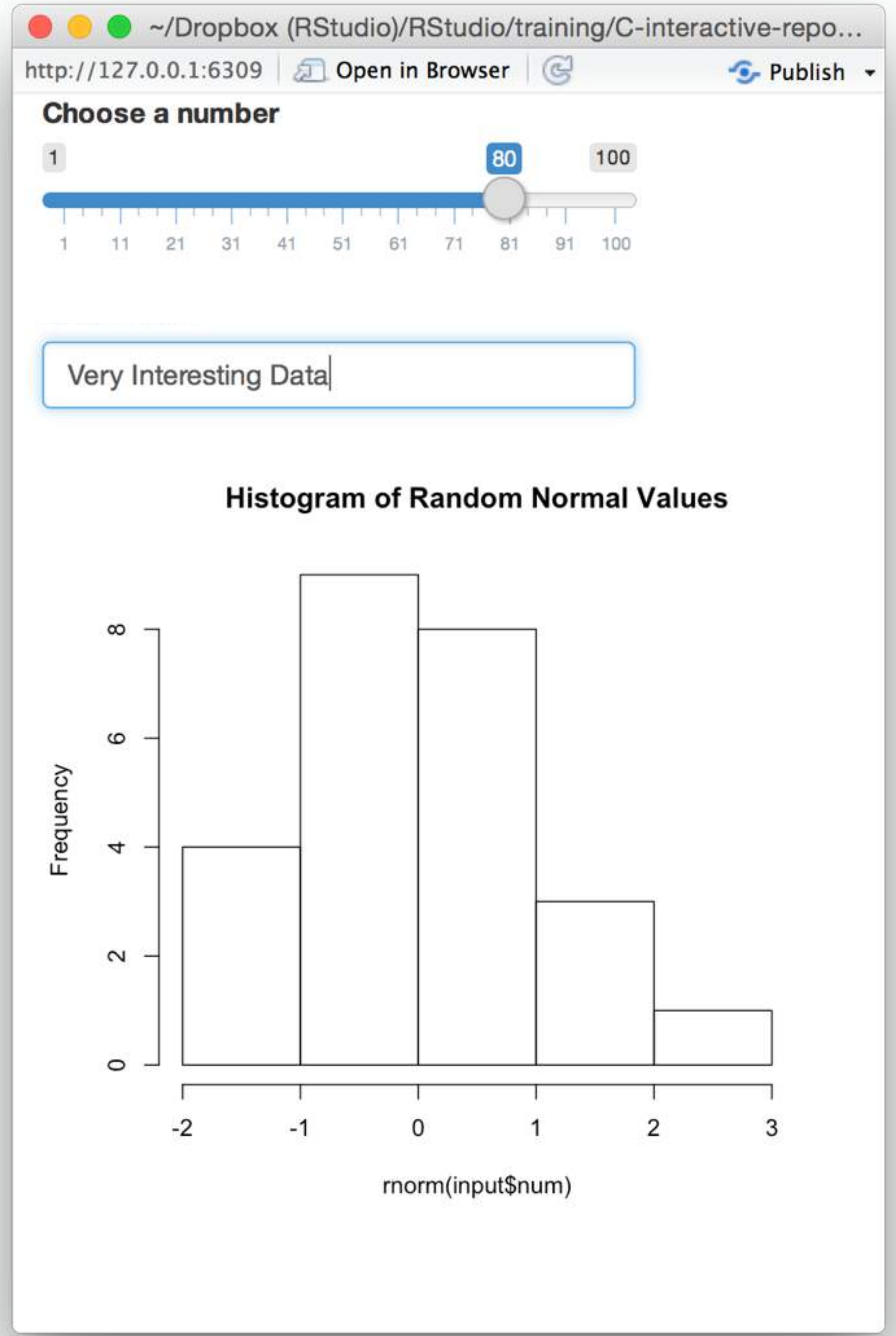
1 80 100

Very Interesting Data

Histogram of Random Normal Values



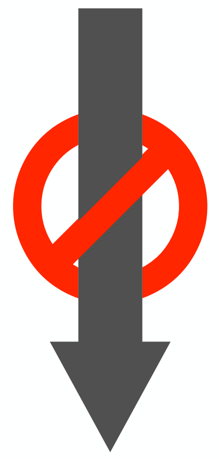
```r
output$hist <- renderPlot({
  hist(rnorm(input$num),
    main = isolate(input$title))
})
```



Choose a number

Very Interesting Data

Recap: isolate()



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

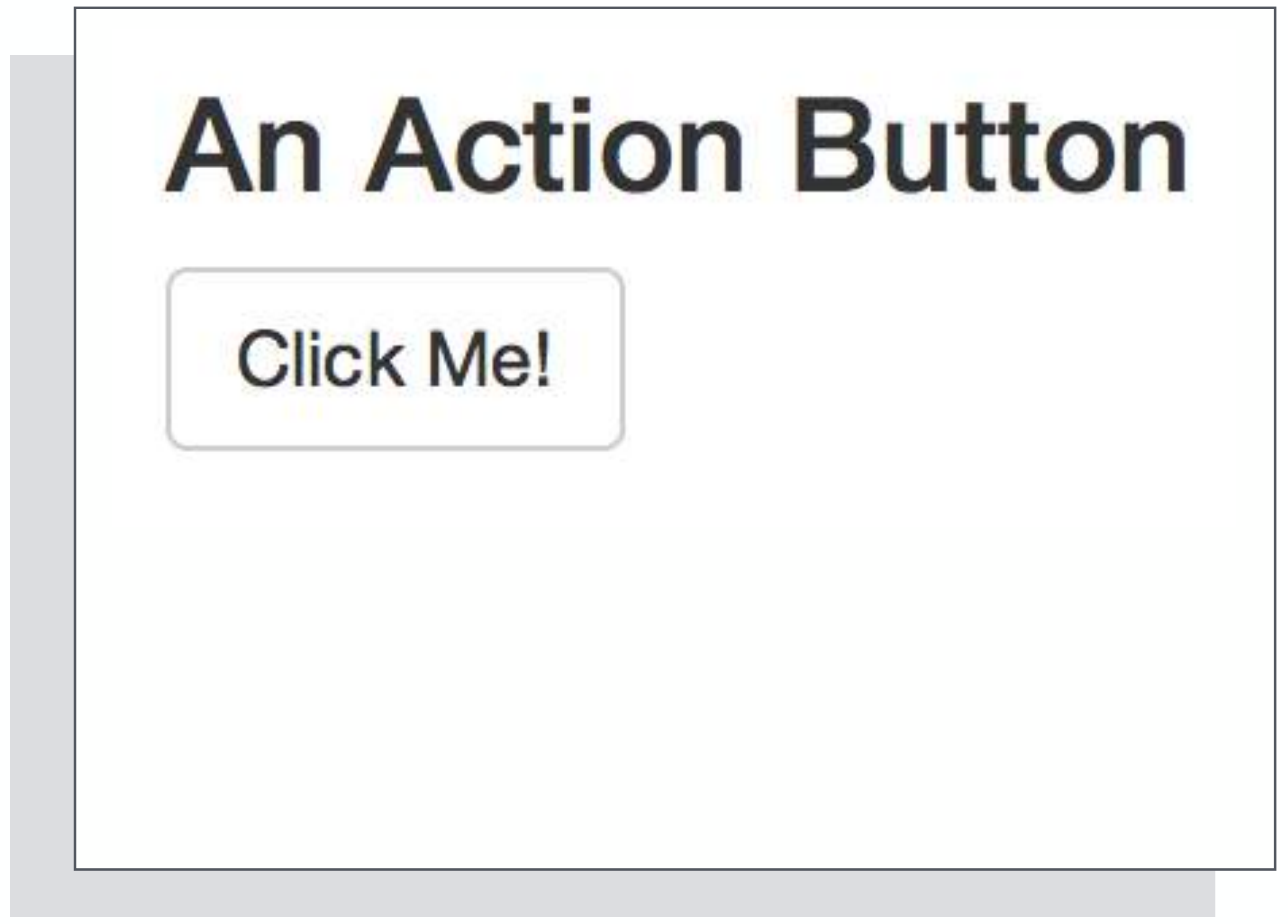
✓ Use isolate() to treat reactive values like normal R values

Trigger code
with observeEvent()

$$\begin{tikzcd}\texttt{input\$x} \arrow[r] & \texttt{expression()} & \texttt{output\$y} \arrow[r] & {}\end{tikzcd}$$

$$\begin{tikzcd}
& \texttt{run(this)} & & \\
\texttt{input\$x} \arrow[ur, bend left] \arrow[rr] & & \texttt{expression()} \arrow[r] & \texttt{output\$y}
\end{tikzcd}$$

Action buttons



An Action Button

input function

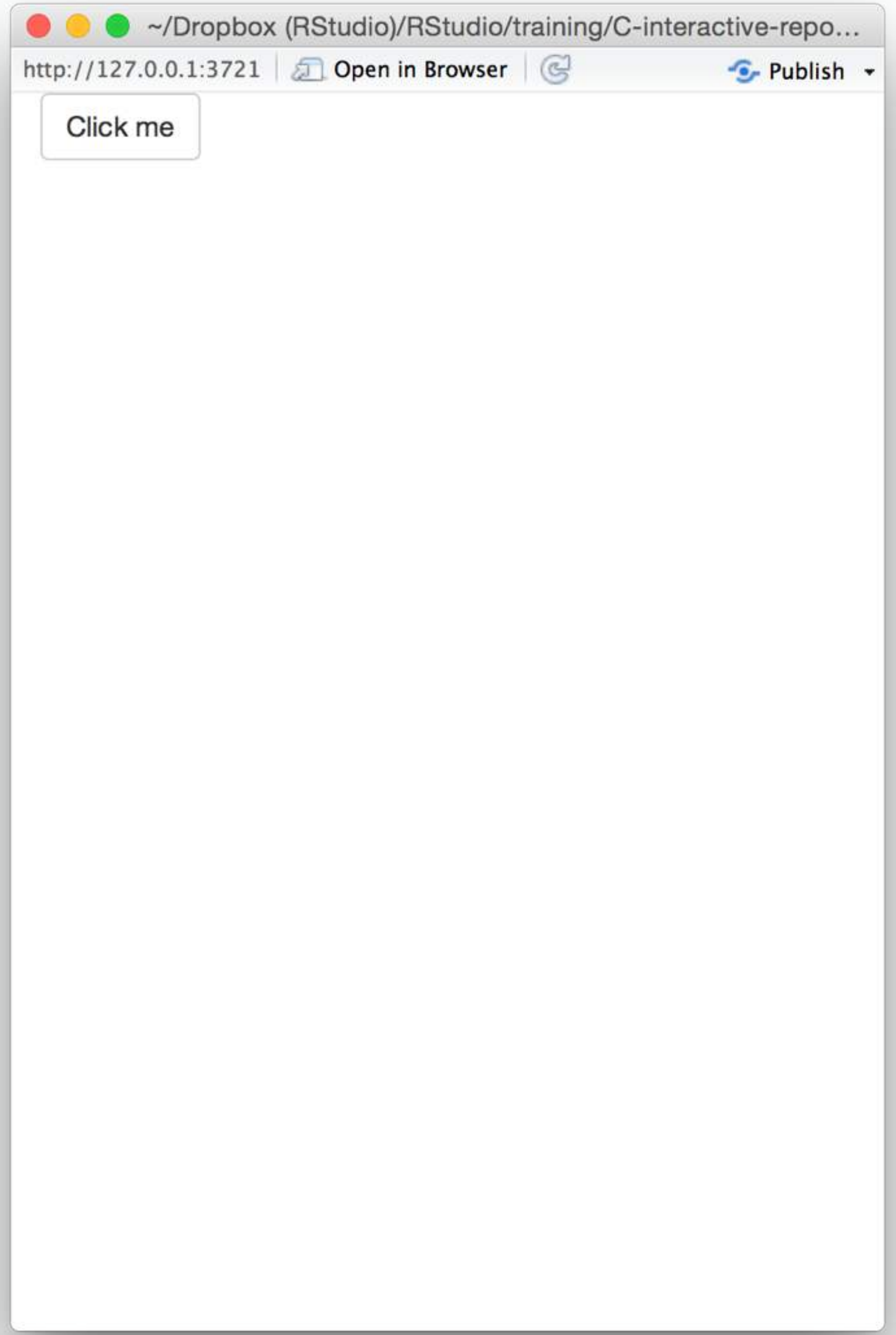
input name (for internal use)

label to display

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

Notice: Id not ID

```
# 05-actionButton

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) })
```

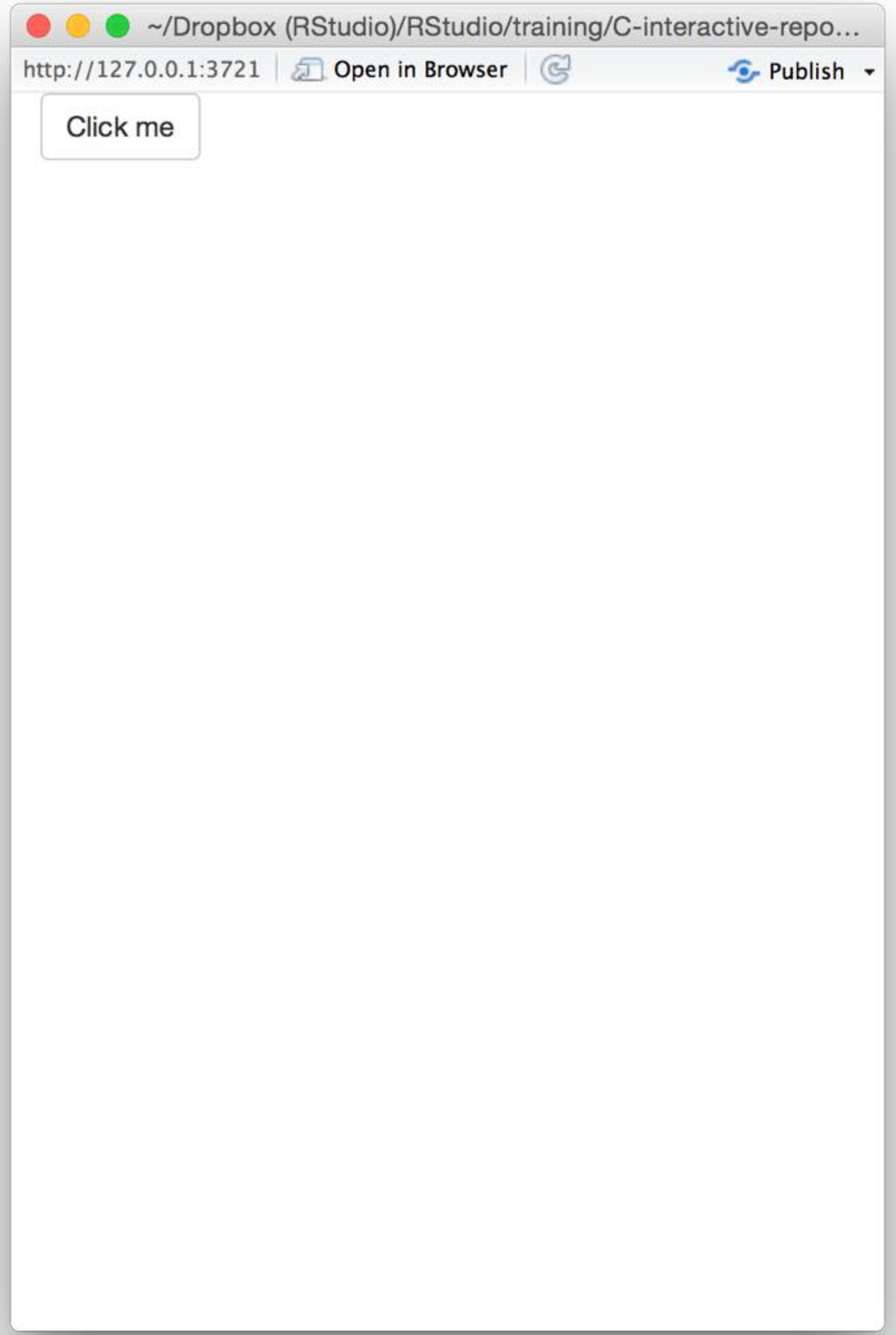
reactive value(s) to respond to

(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()

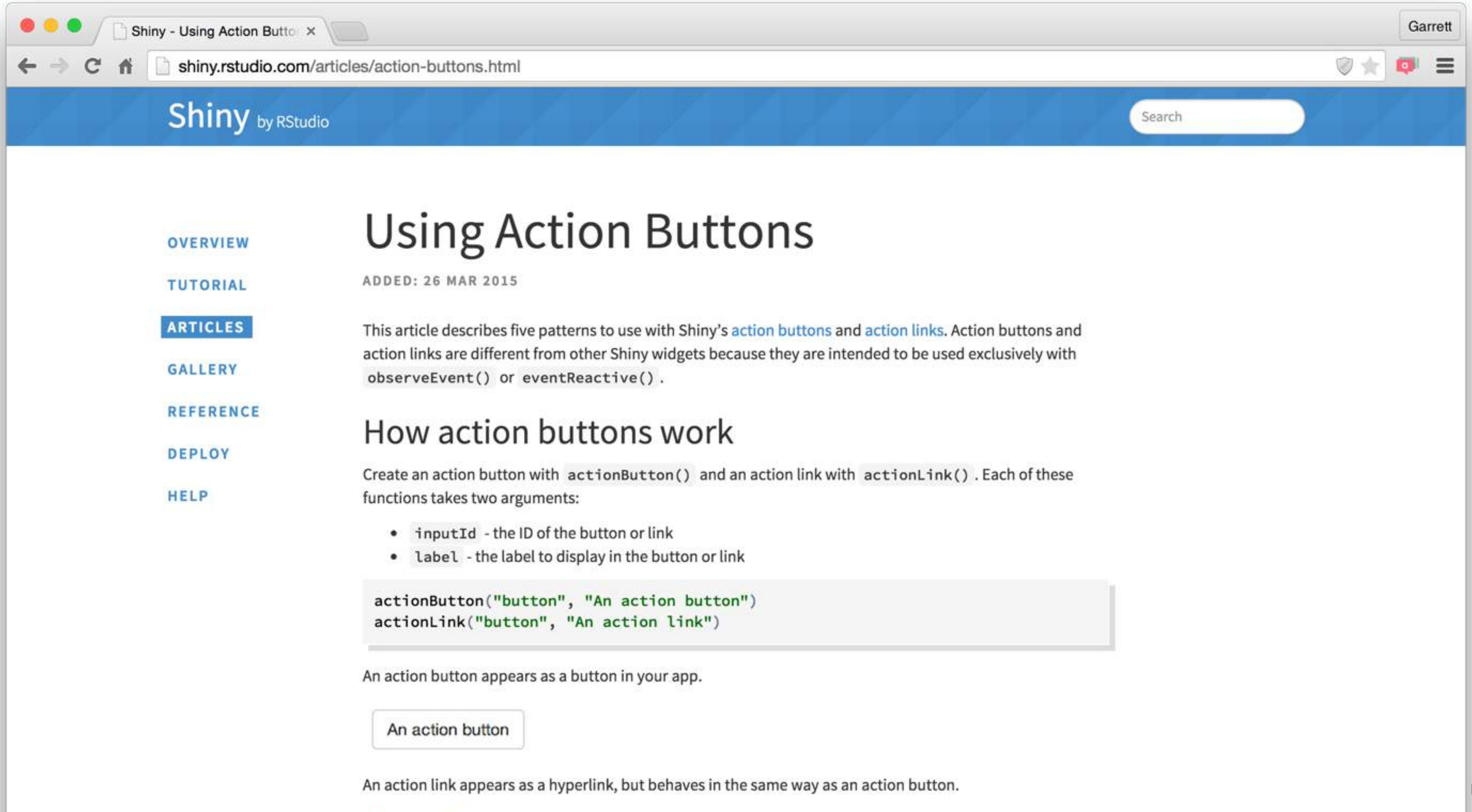
```r
# 05-actionButton

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)
```


~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...

http://127.0.0.1:3721

Action buttons article

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



Using Action Buttons

ADDED: 26 MAR 2015

This article describes five patterns to use with Shiny's action buttons and action links. Action buttons and action links are different from other Shiny widgets because they are intended to be used exclusively with `observeEvent()` or `eventReactive()`.

How action buttons work

Create an action button with `actionButton()` and an action link with `actionLink()`. Each of these functions takes two arguments:

- `inputId` - the ID of the button or link
- `label` - the label to display in the button or link

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

An action button appears as a button in your app.

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.

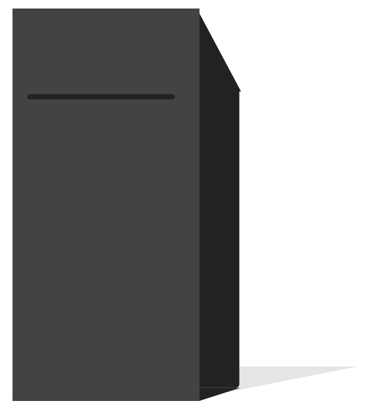
Uses same syntax as render*(), reactive(), and isolate()

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

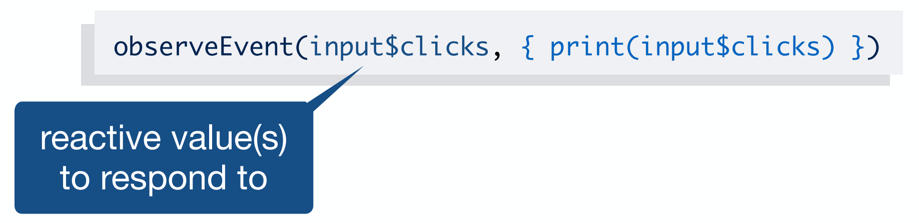
observer will respond to *every reactive value in the code*

code block to run whenever observer is invalidated

Recap: observeEvent()



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



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

Specify **precisely** which reactive values should invalidate the observer

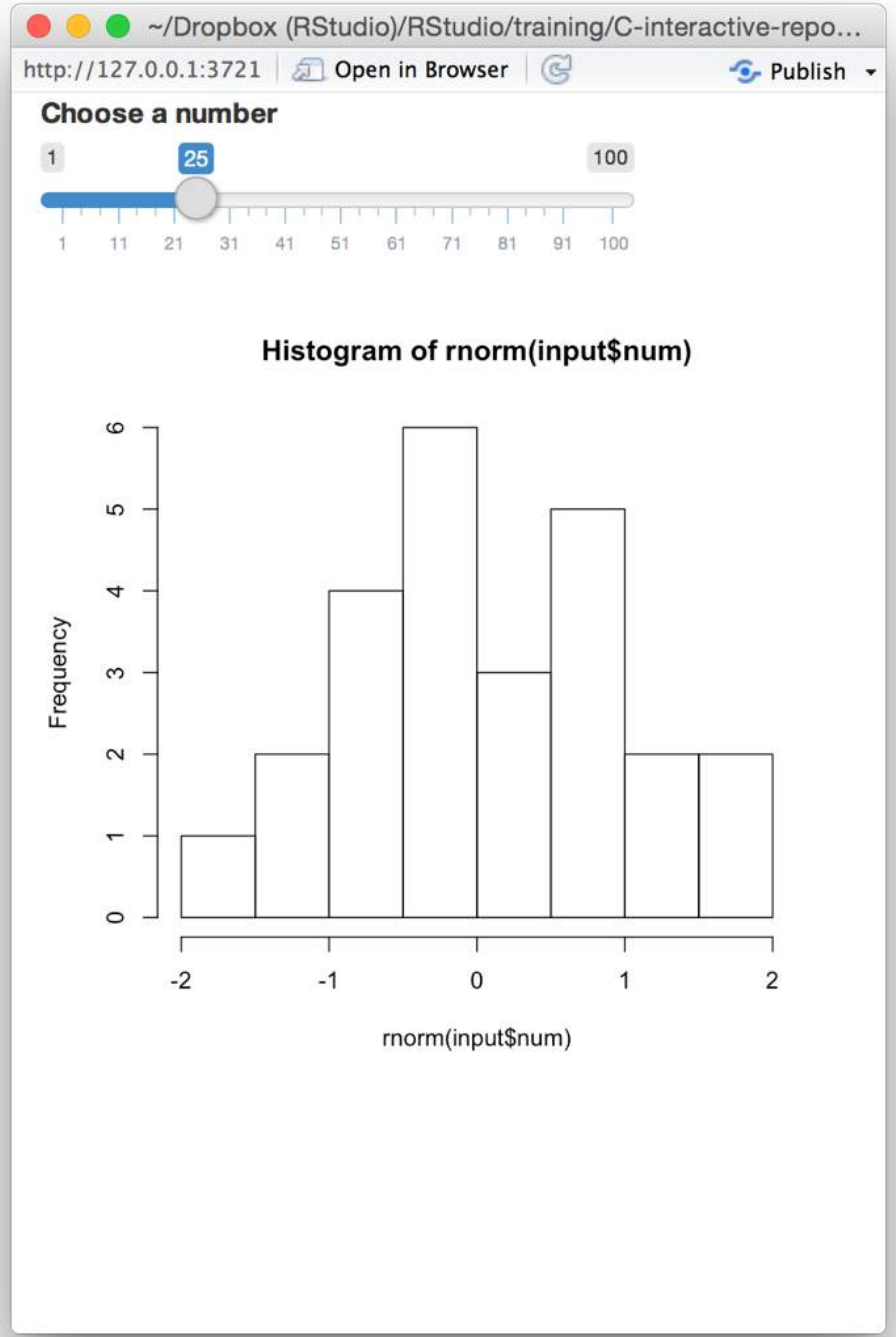
`observe()`

Use **observe()** for a more implicit syntax

Delay reactions

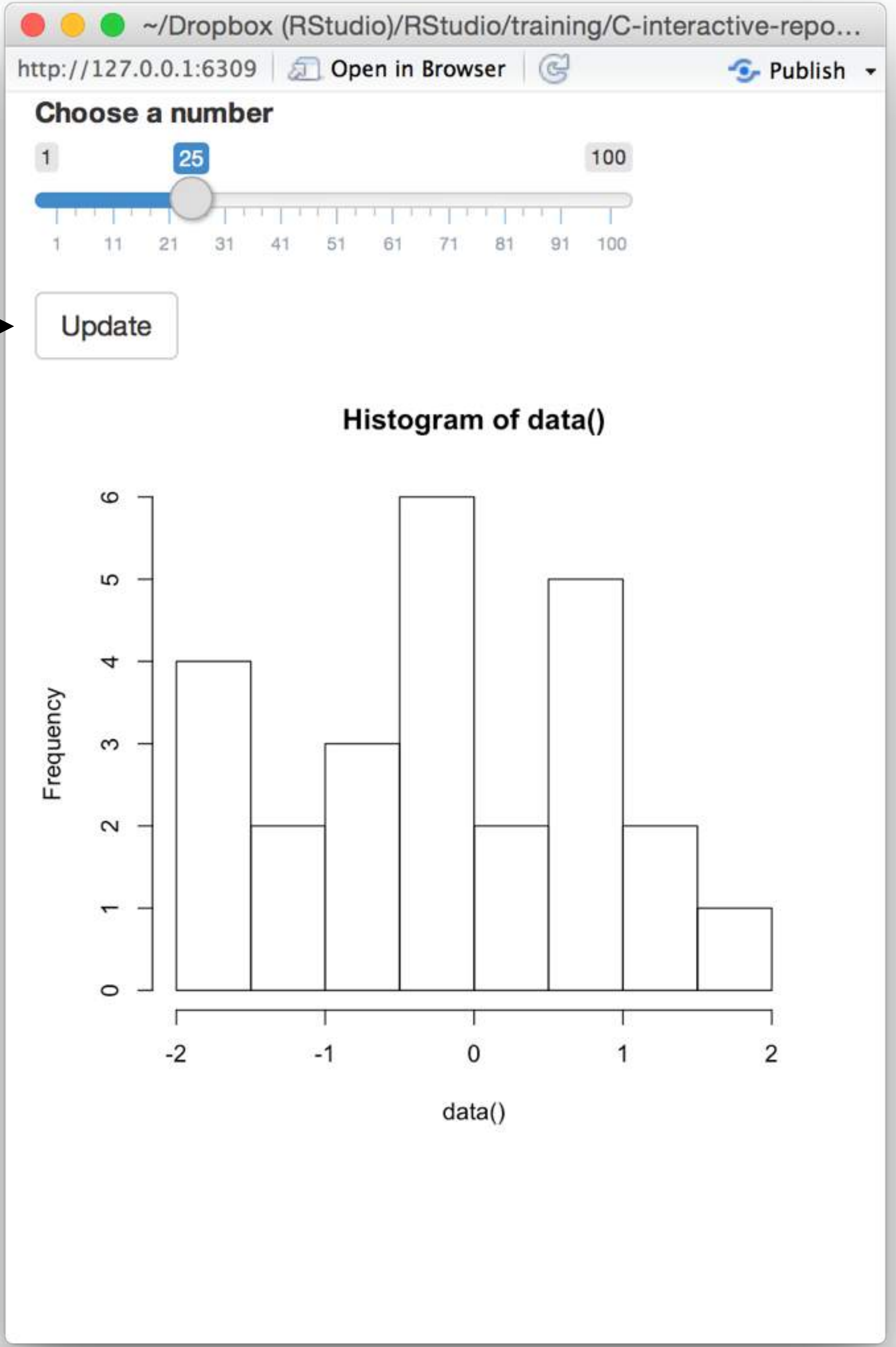
with eventReactive()

```
# 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) })
```

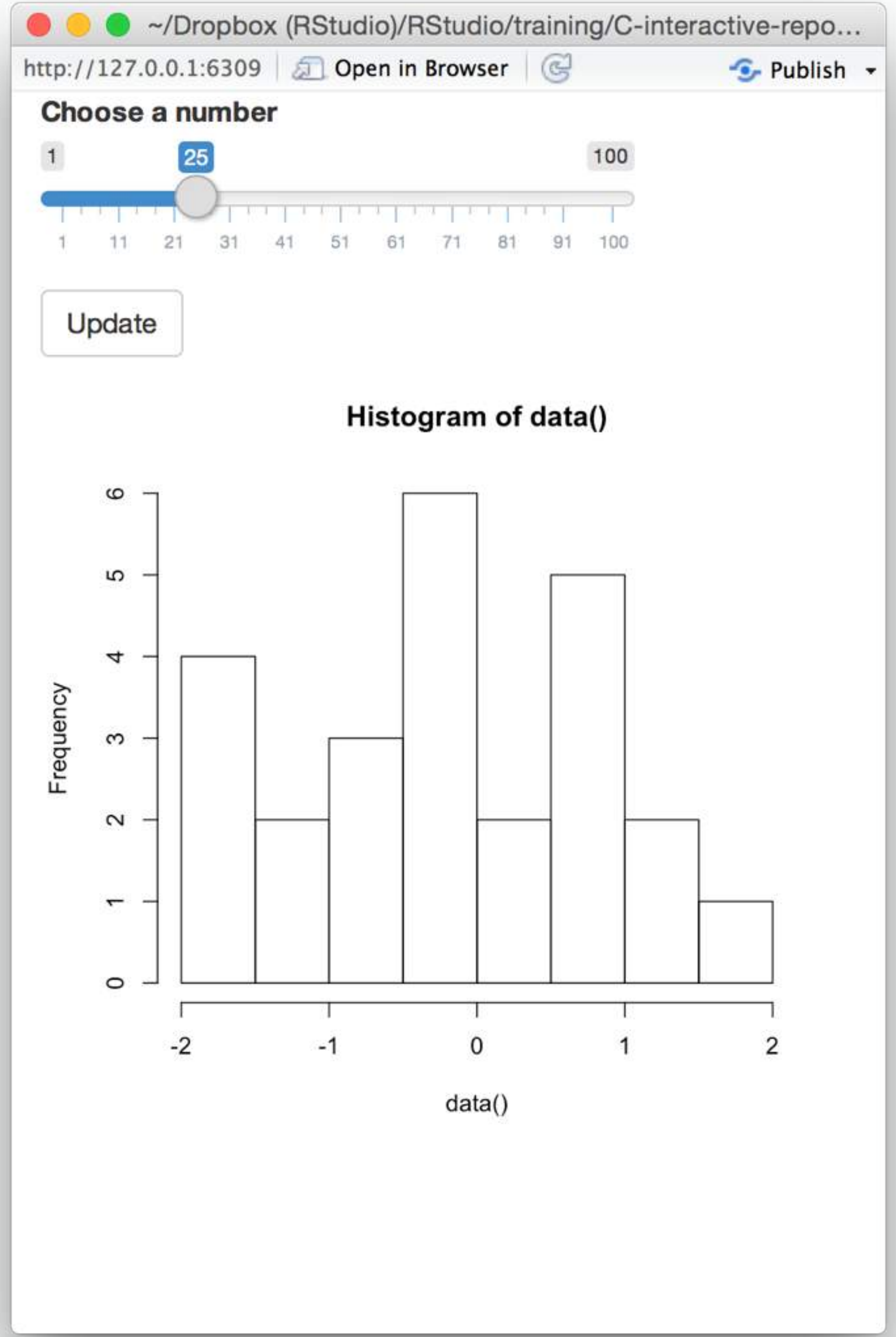
reactive value(s) to respond to

(expression invalidates ONLY when this value changes)

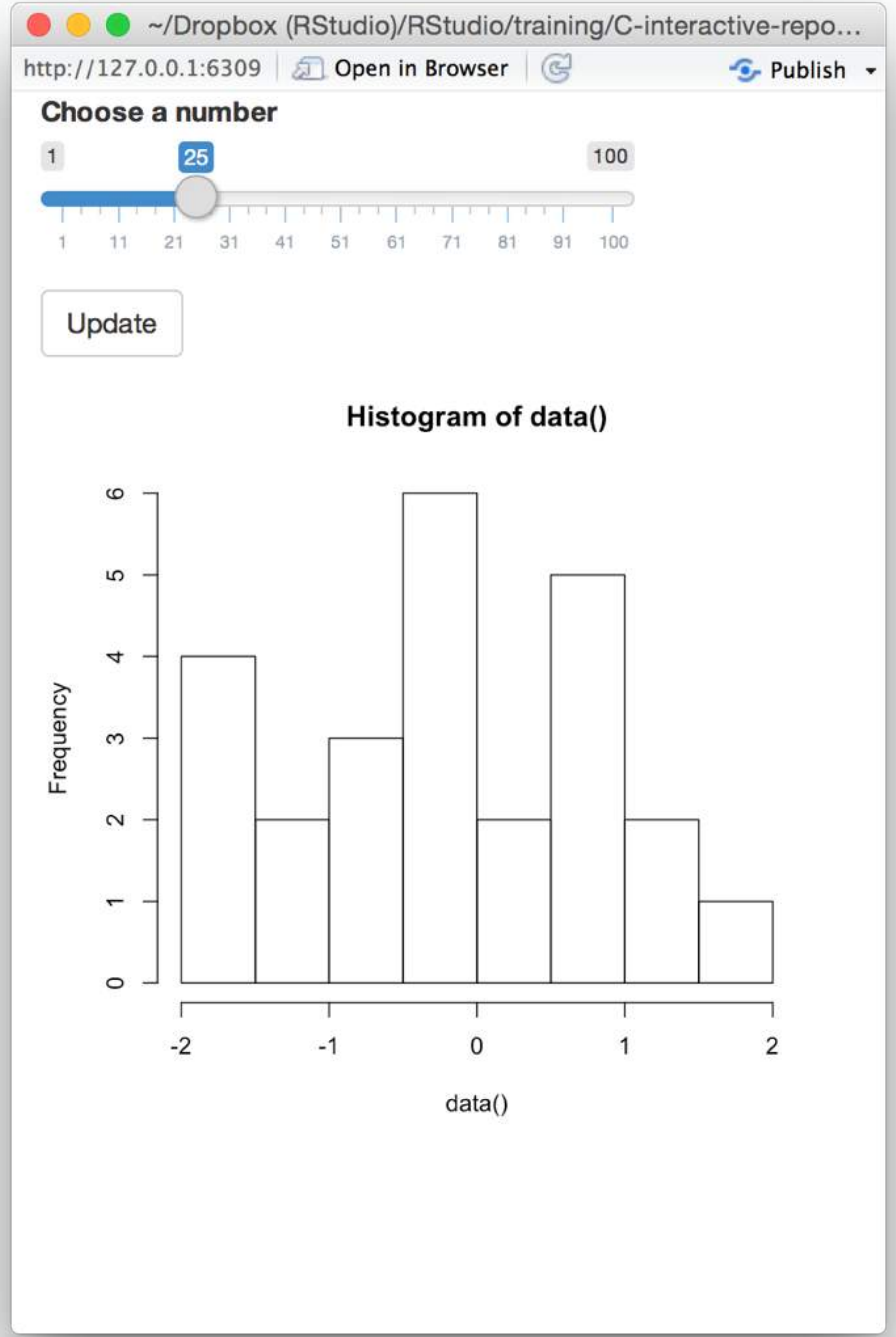
code used to build (and rebuild) object

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

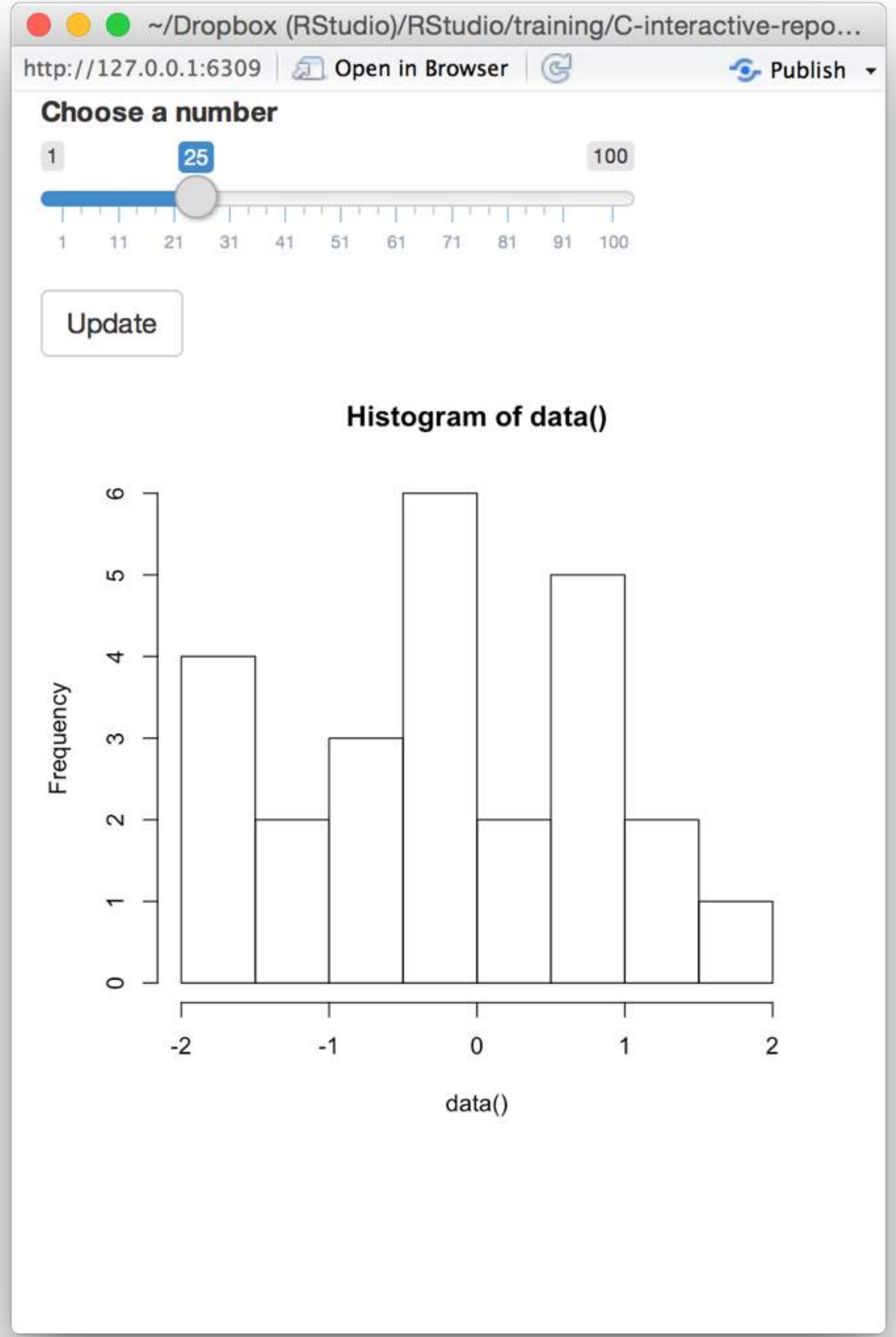
```r
# 07-eventReactive

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)
```



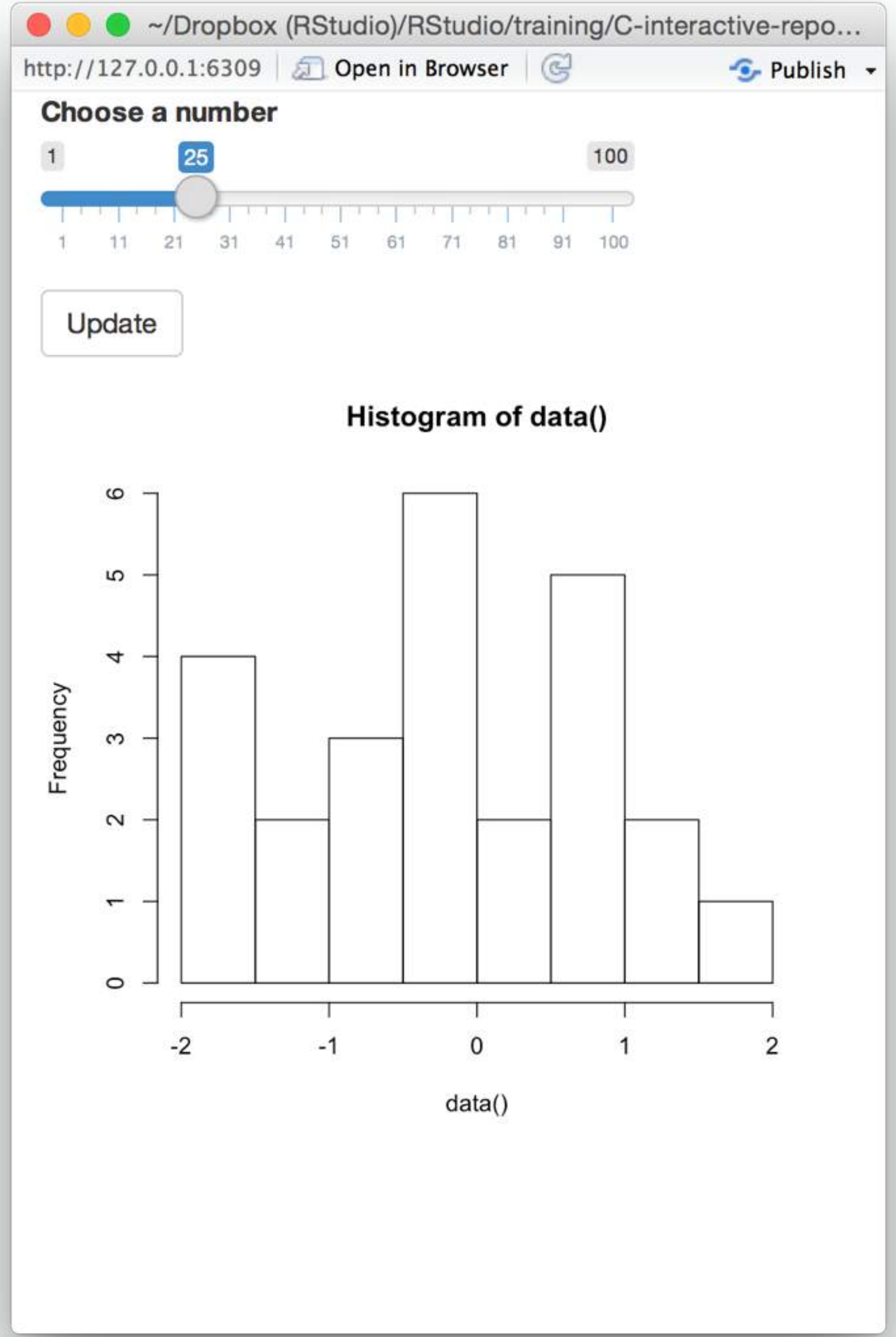
```
# 07-eventReactive

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)
```



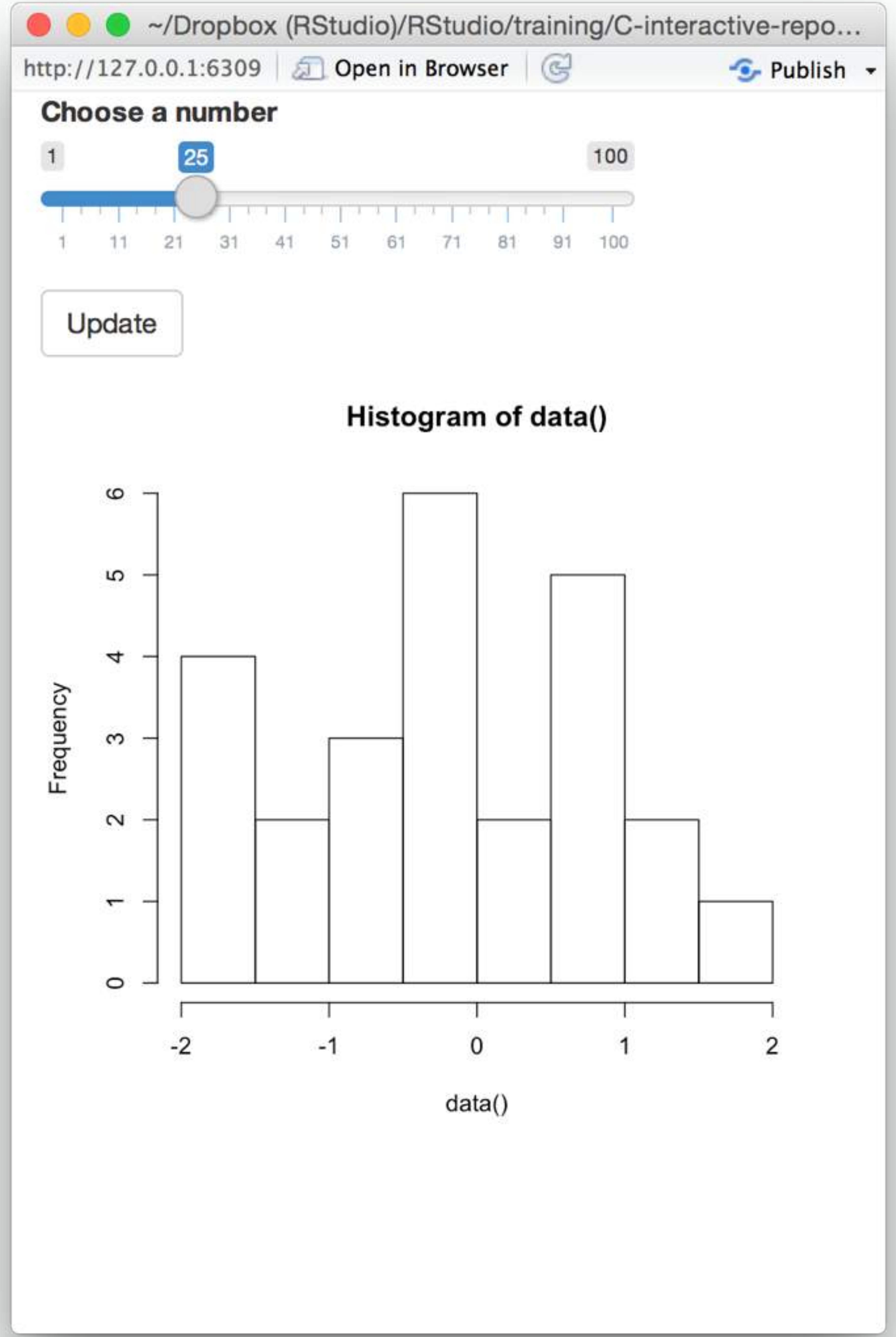
```
# 07-eventReactive

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)
```

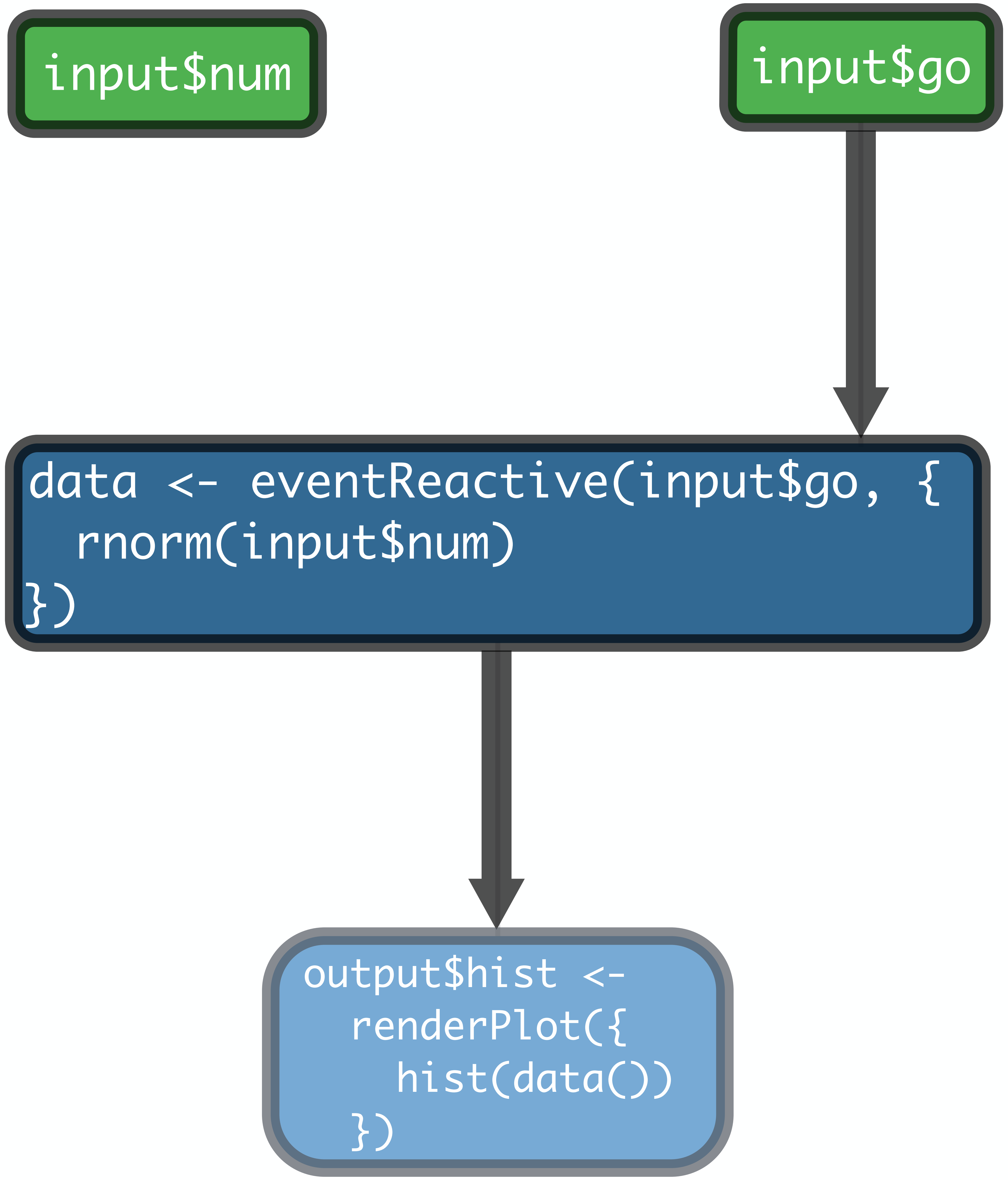


```r
# 07-eventReactive

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)
```

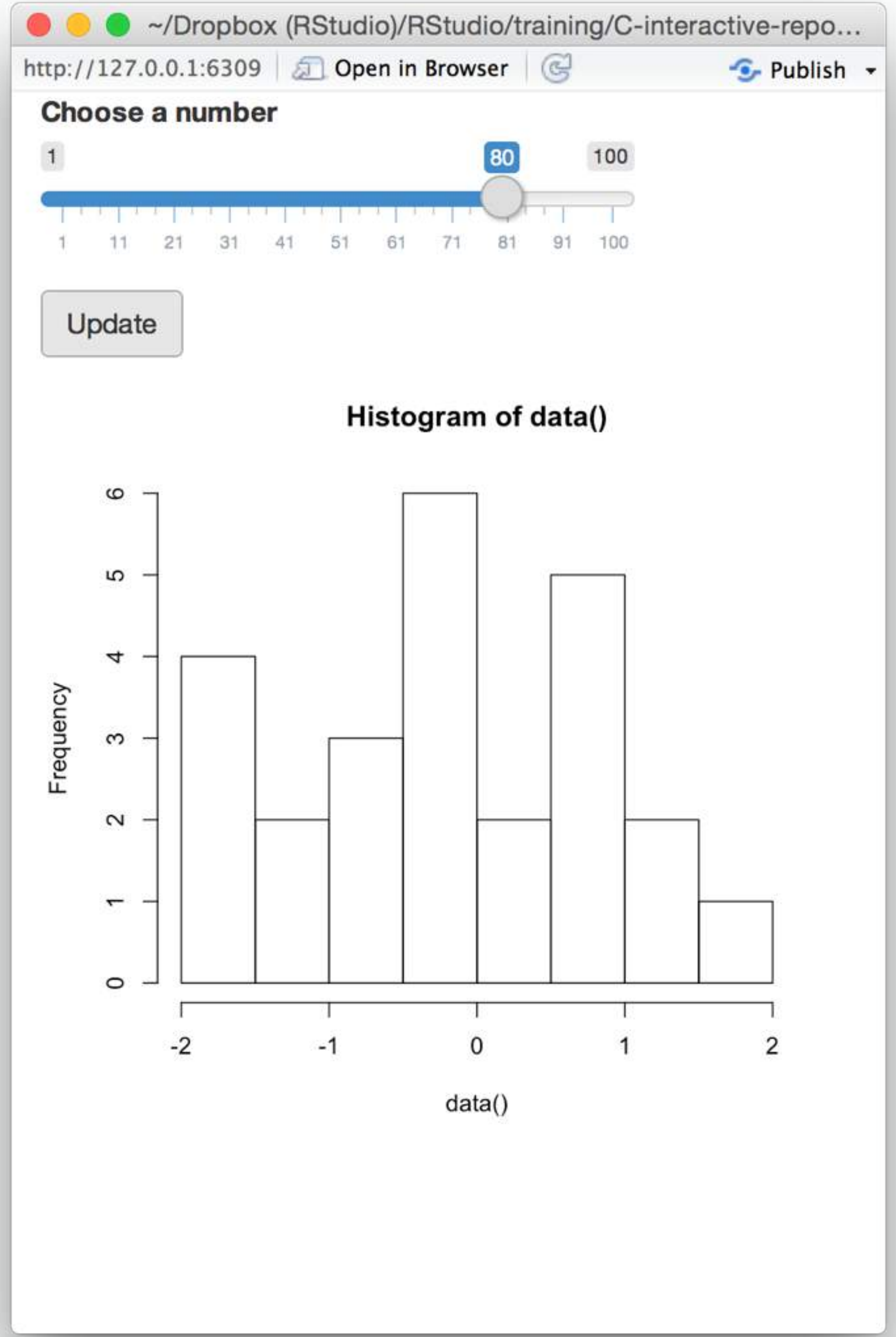


```
# 07-eventReactive

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)
```

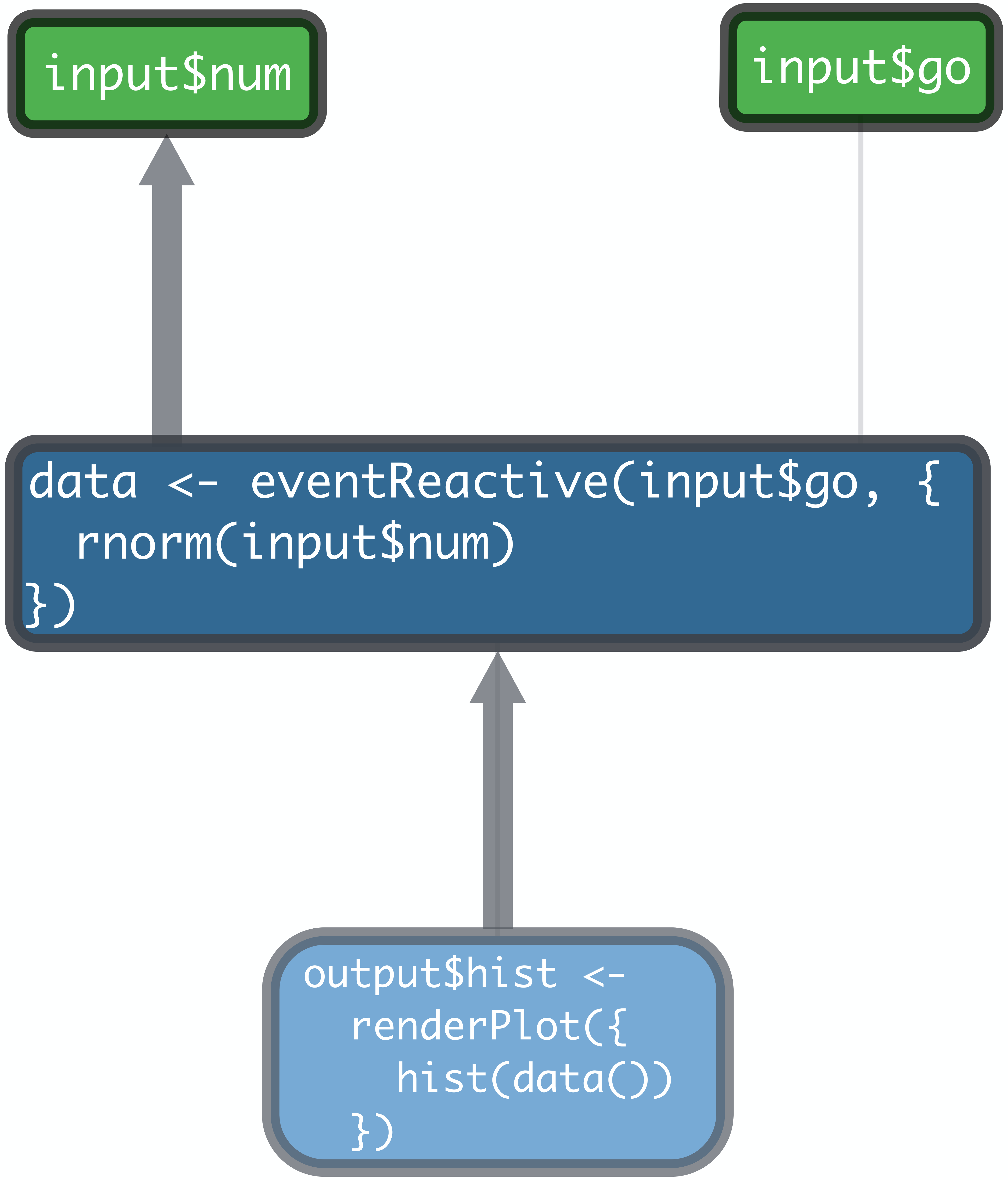




```
input$num

input$go

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

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



Choose a number: 80

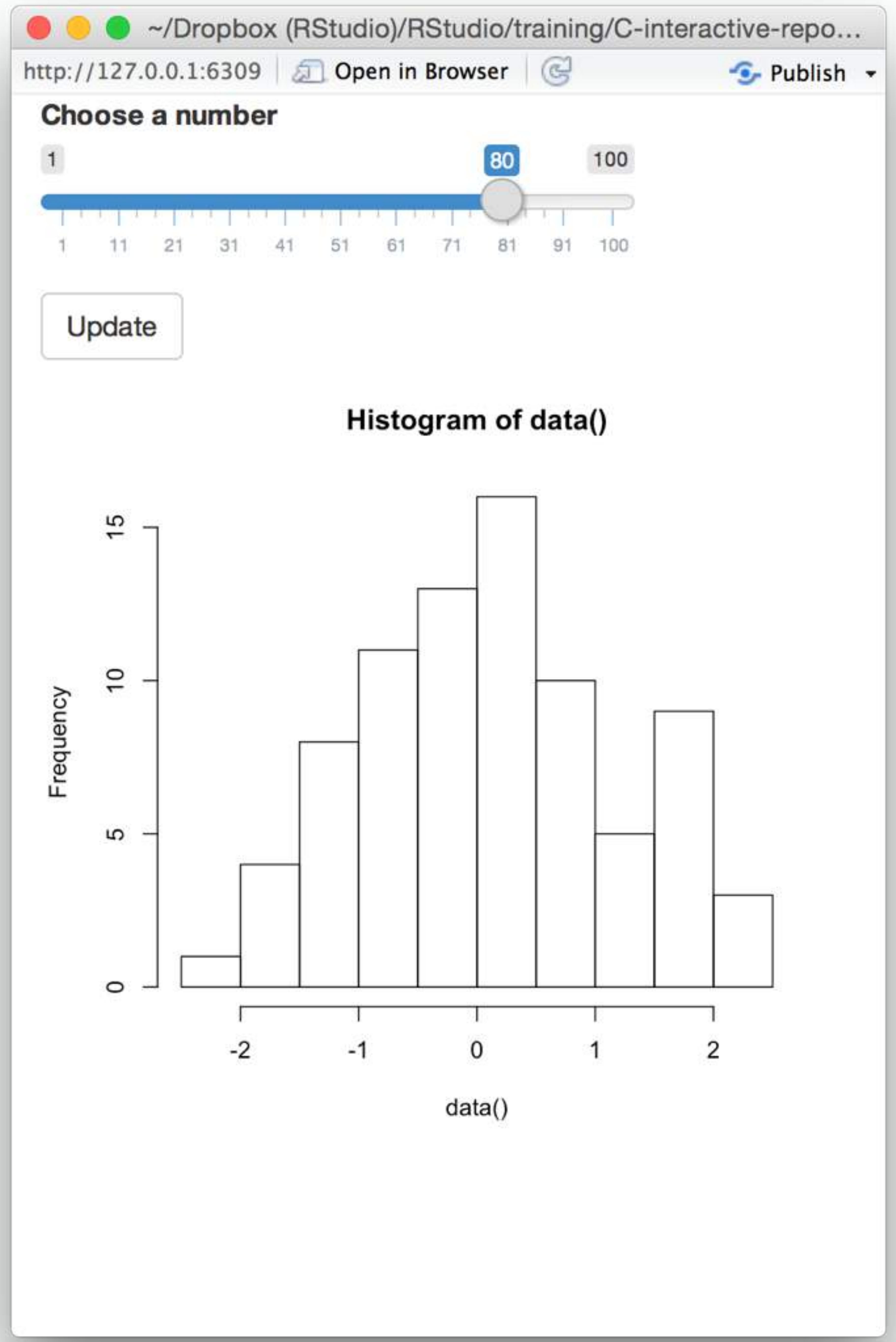


```
input$num
```

```
input$go
```

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

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



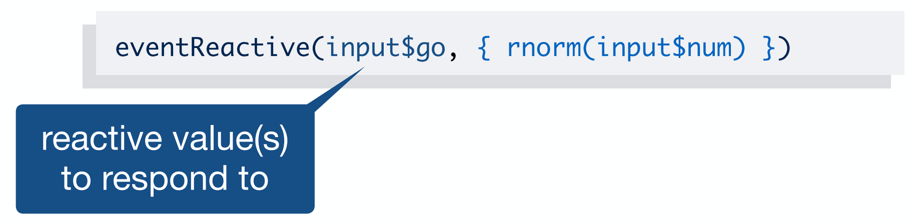
Recap: eventReactive()

Update

Use eventReactive() to **delay reactions**

data()

eventReactive() creates a **reactive expression**



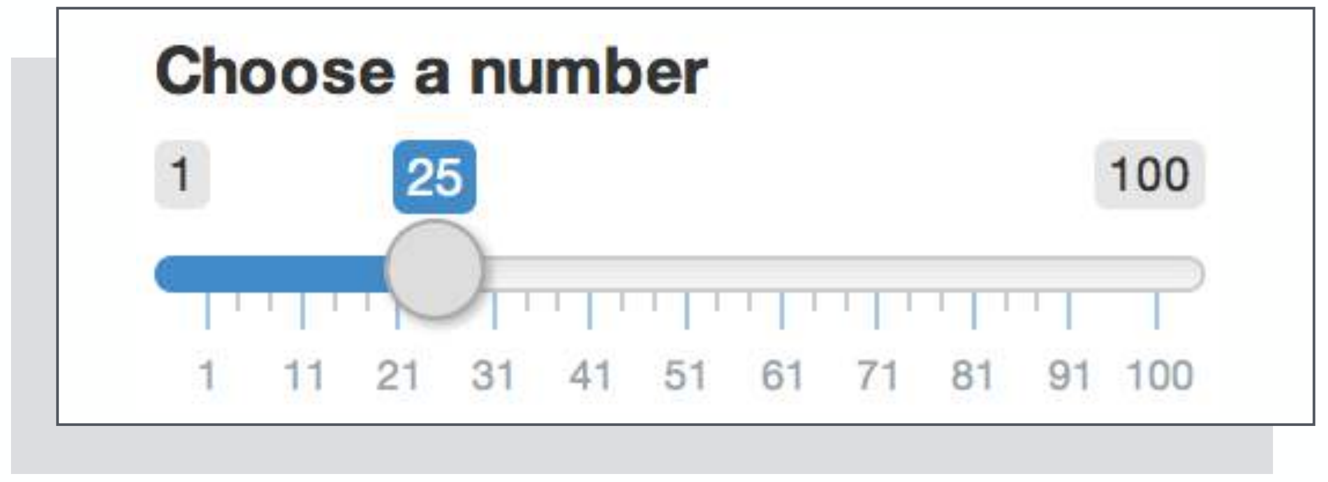
```
eventReactive(input$go, { rnorm(input$num) })
```

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

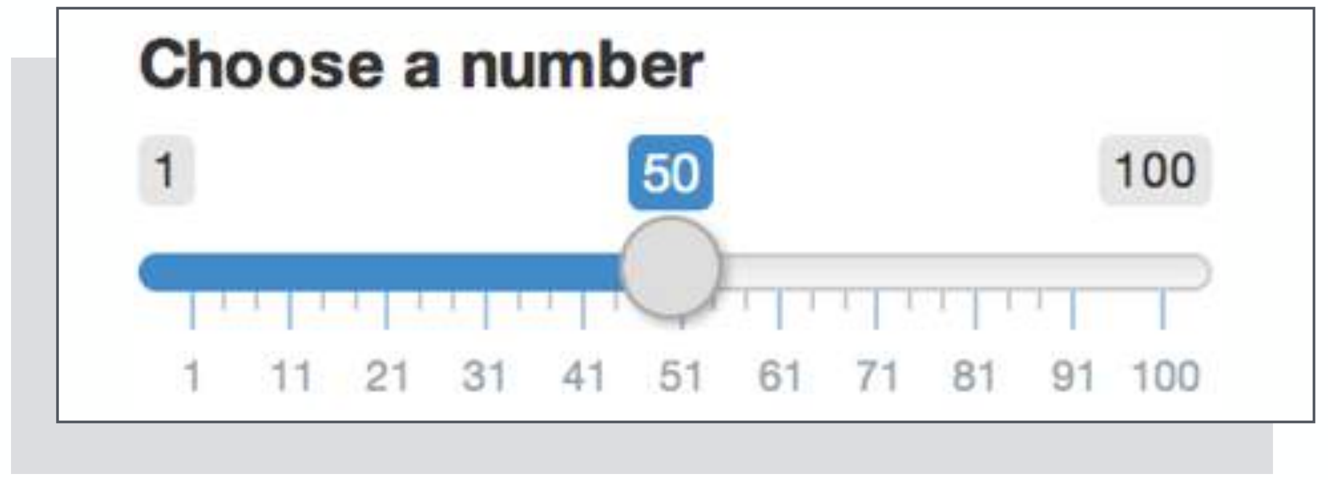
Manage state
with reactiveValues()

Input values

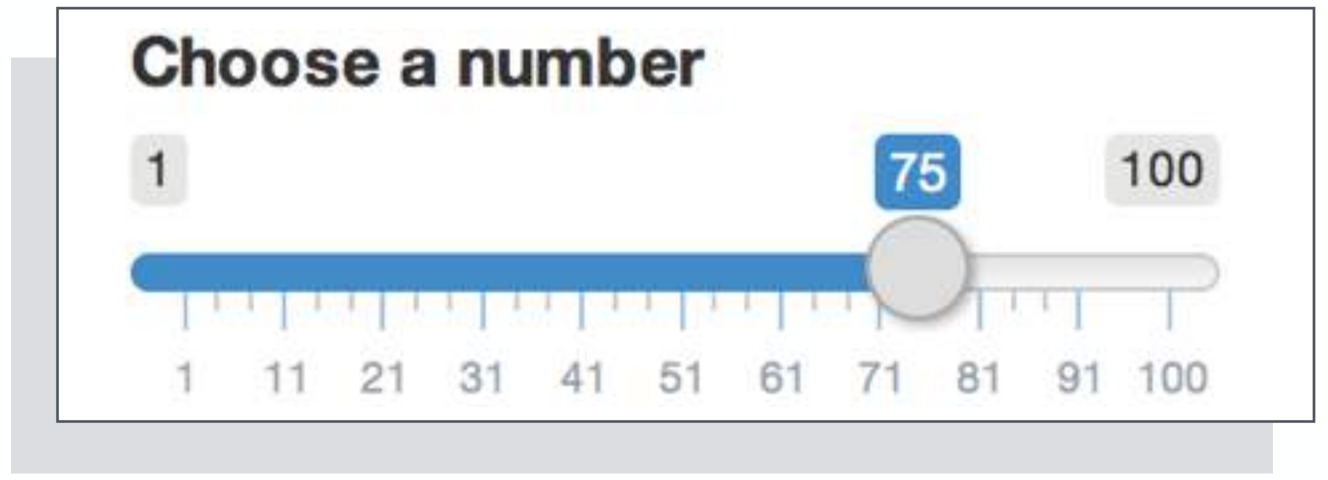
The input value changes whenever a user changes the input.



```
input$num = 25
```



```
input$num = 50
```



```
input$num = 75
```

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

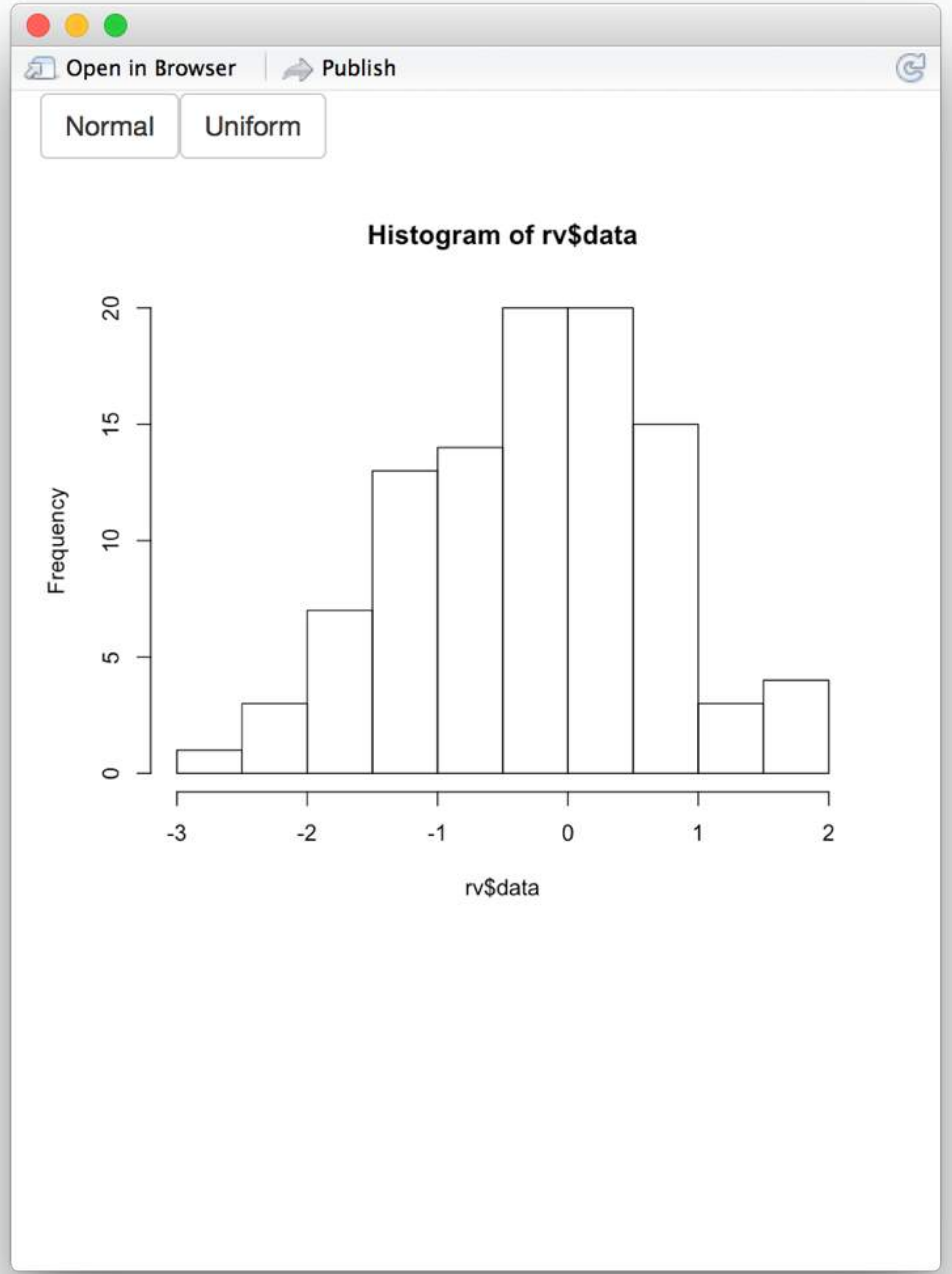
reactiveValues()

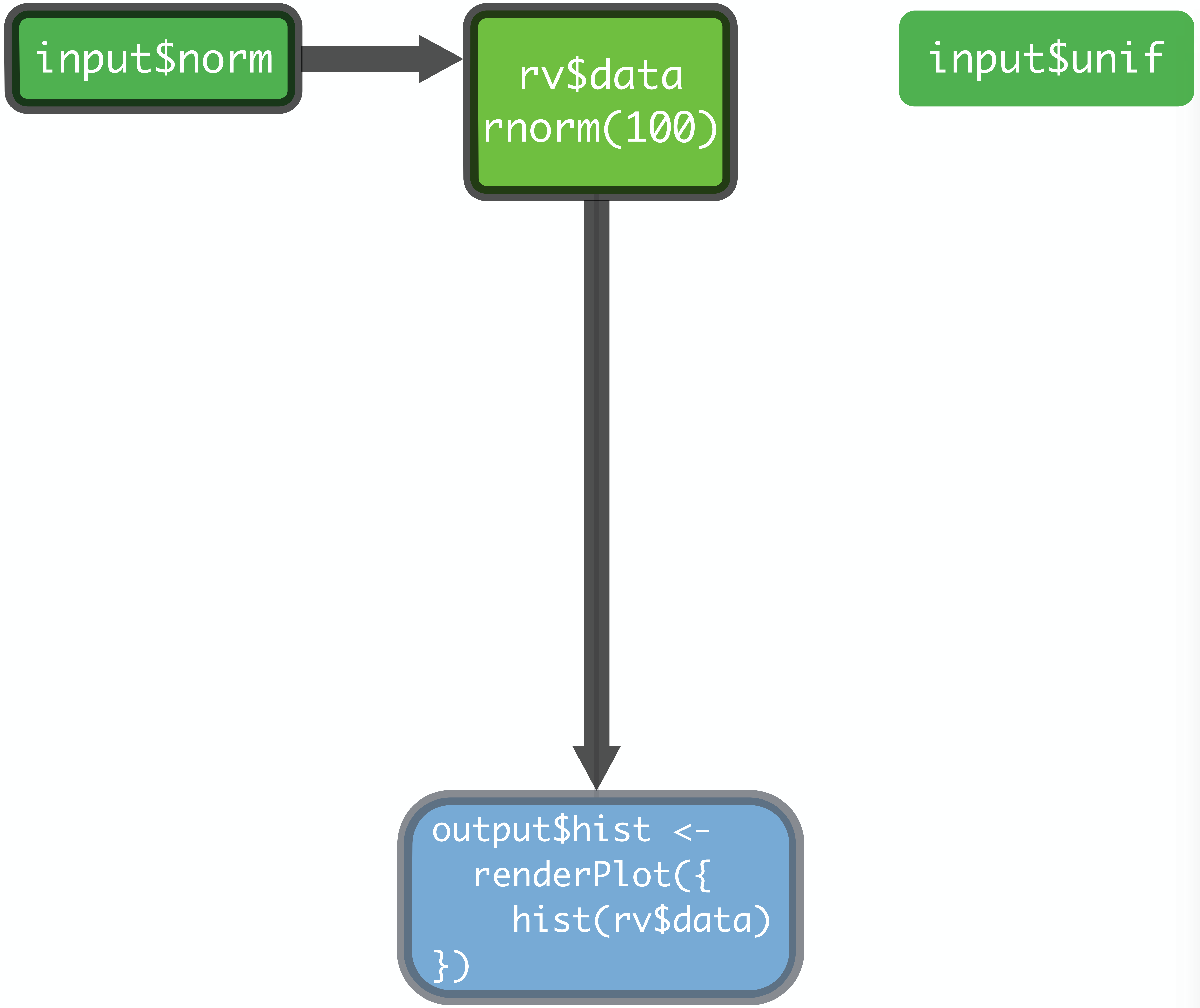
Creates a list of reactive values to manipulate programmatically

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

(optional) elements to add to the list

```r
# 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)
```



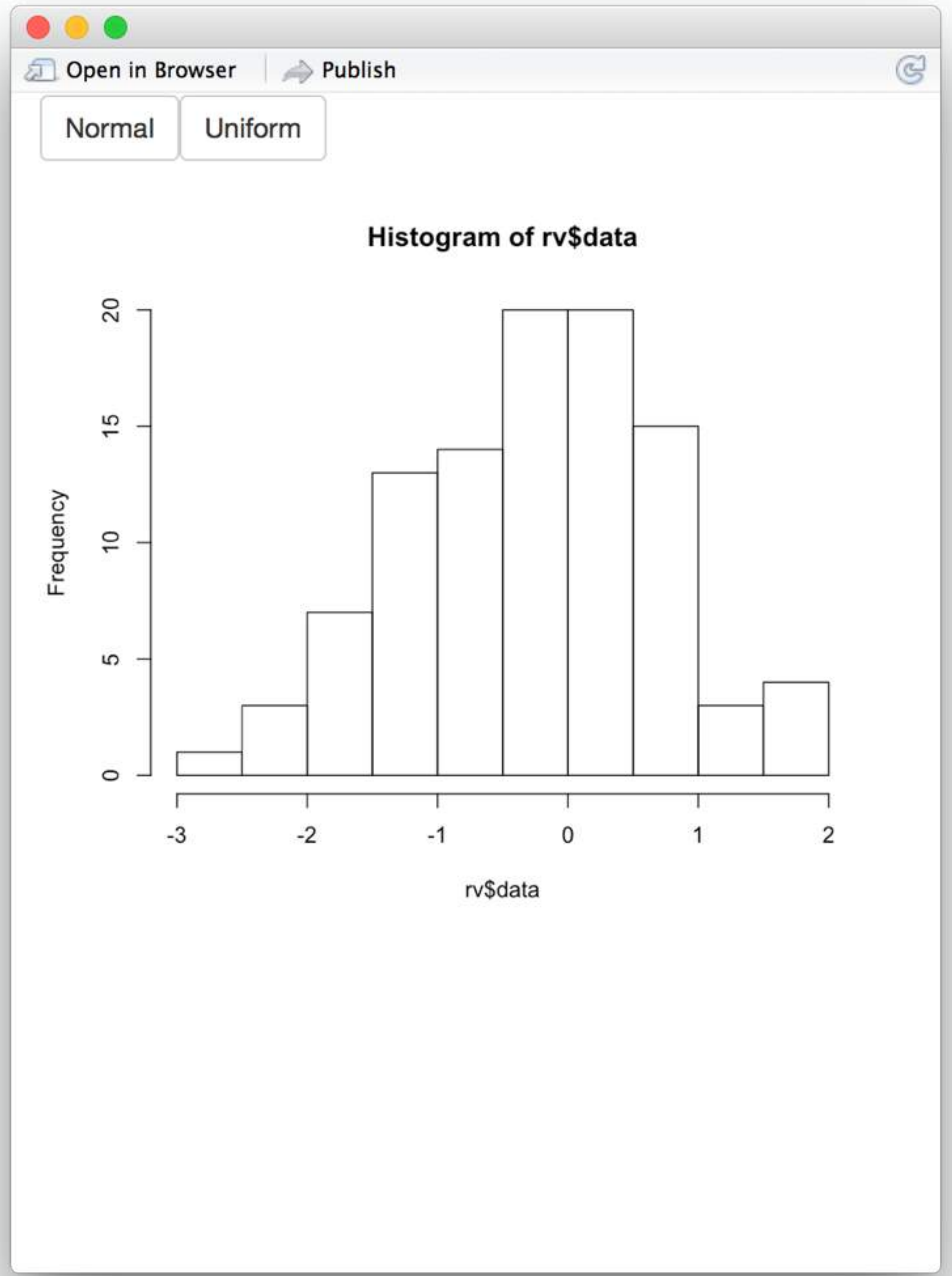


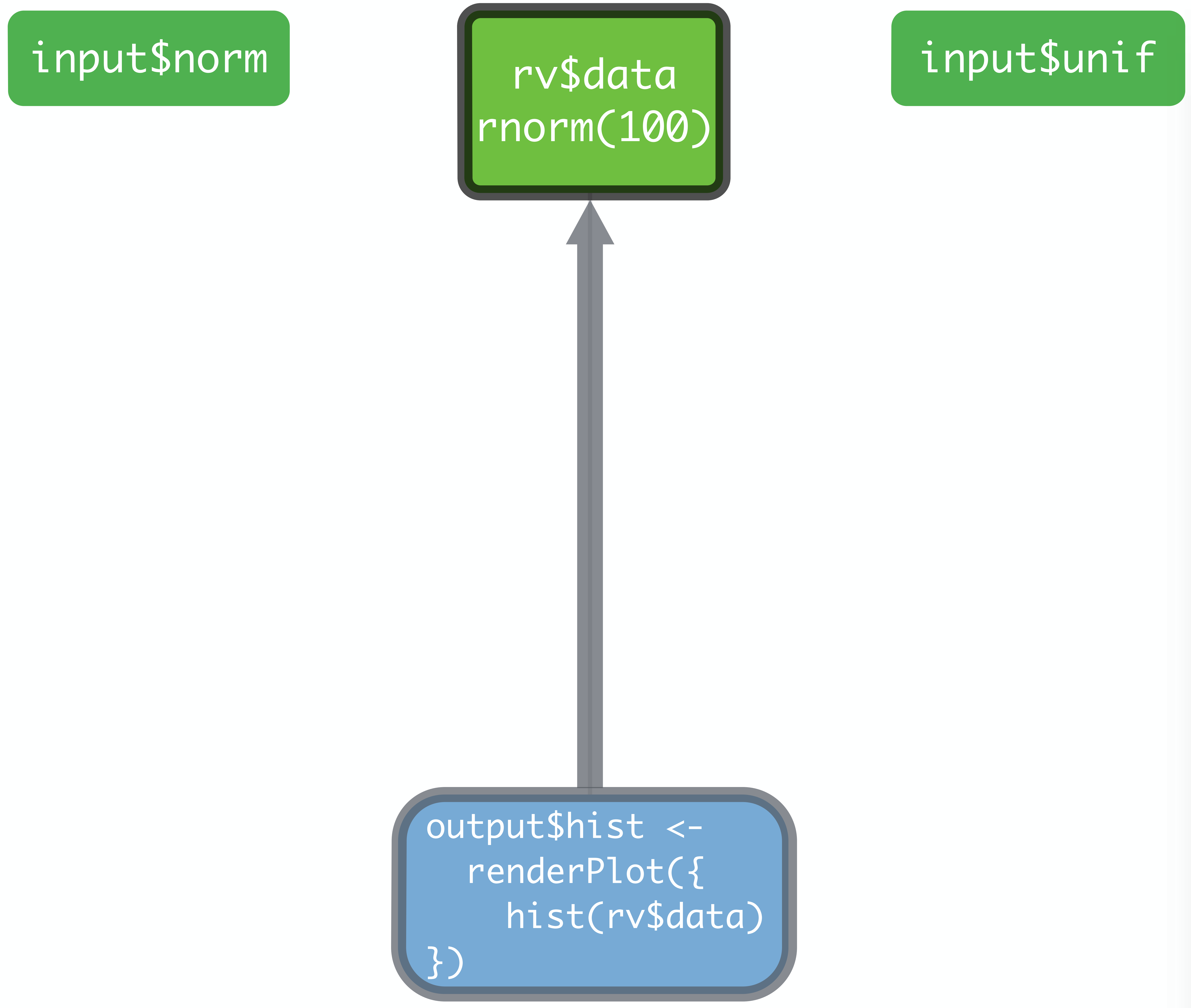
```
input$norm
```

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

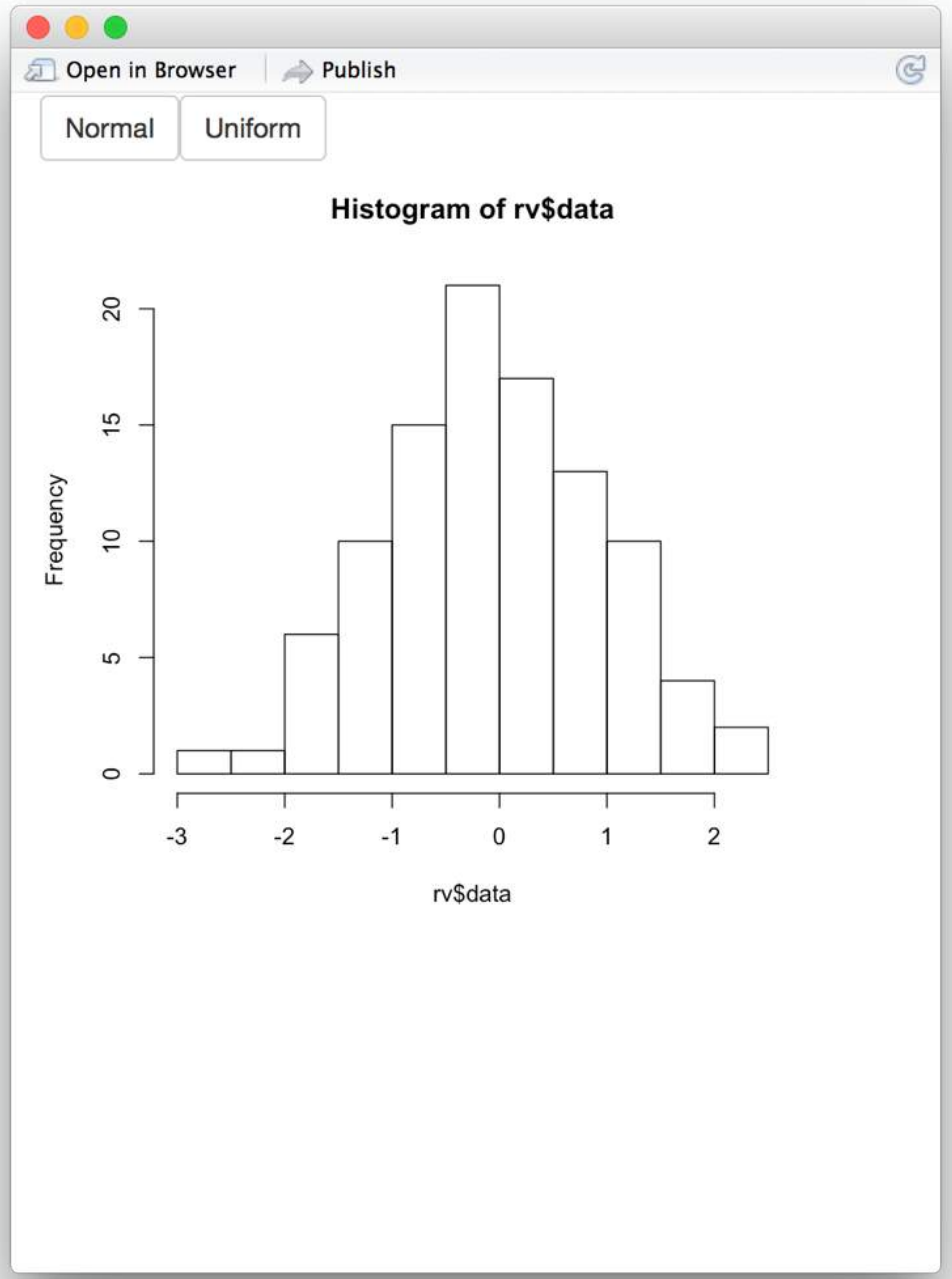
```
input$unif
```

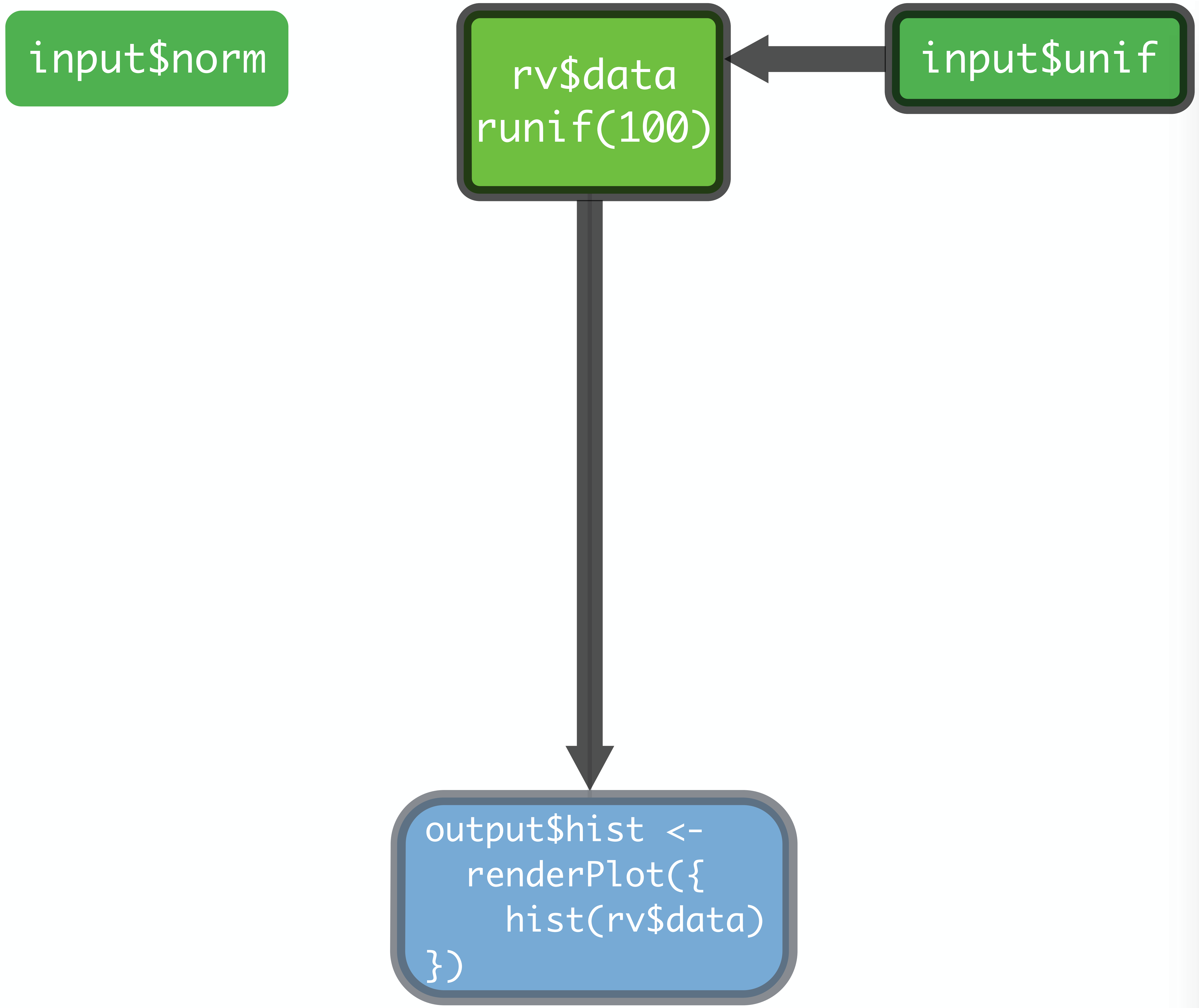
```
output$hist <-
  renderPlot({
    hist(rv$data)
})
```



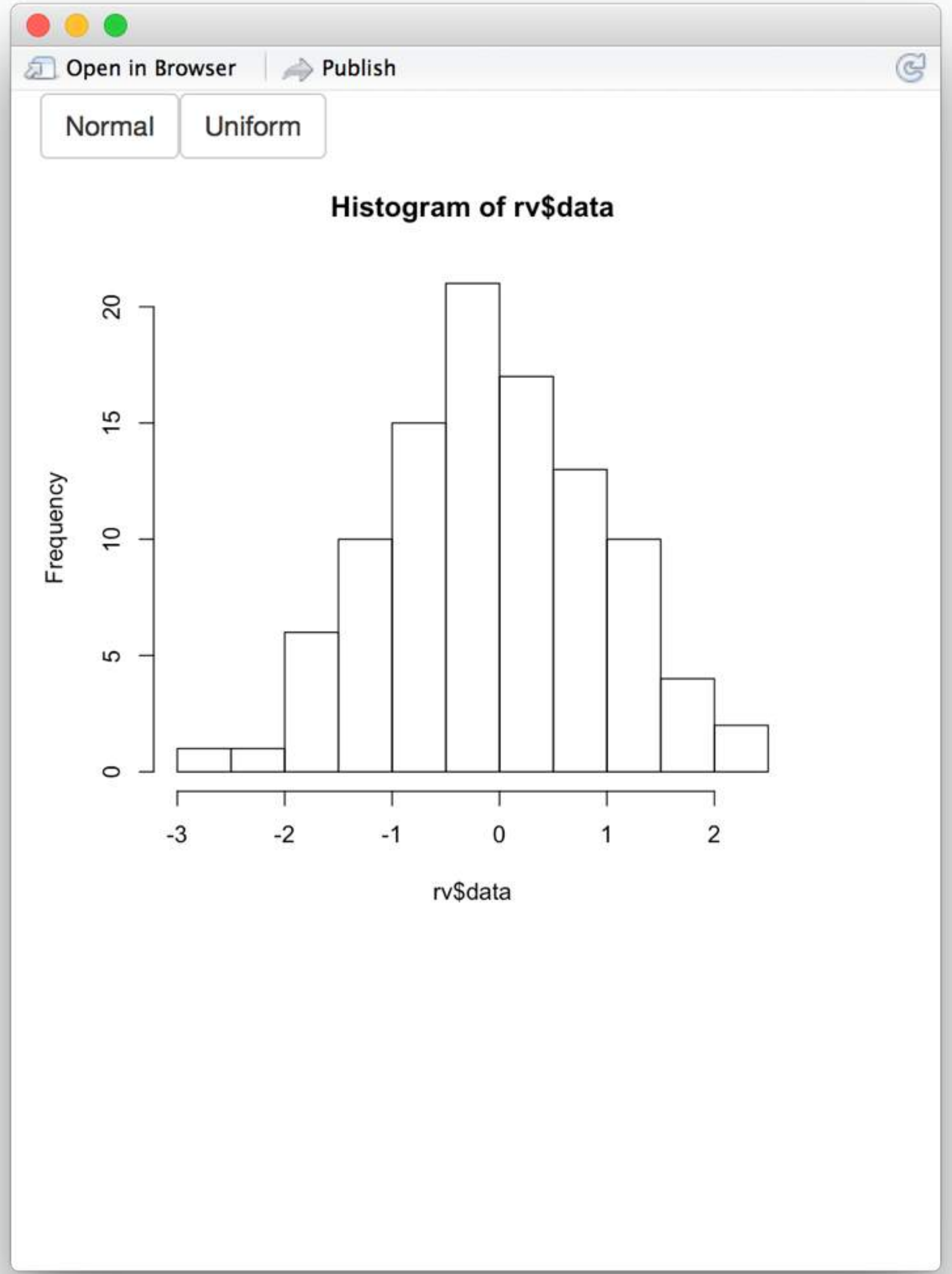


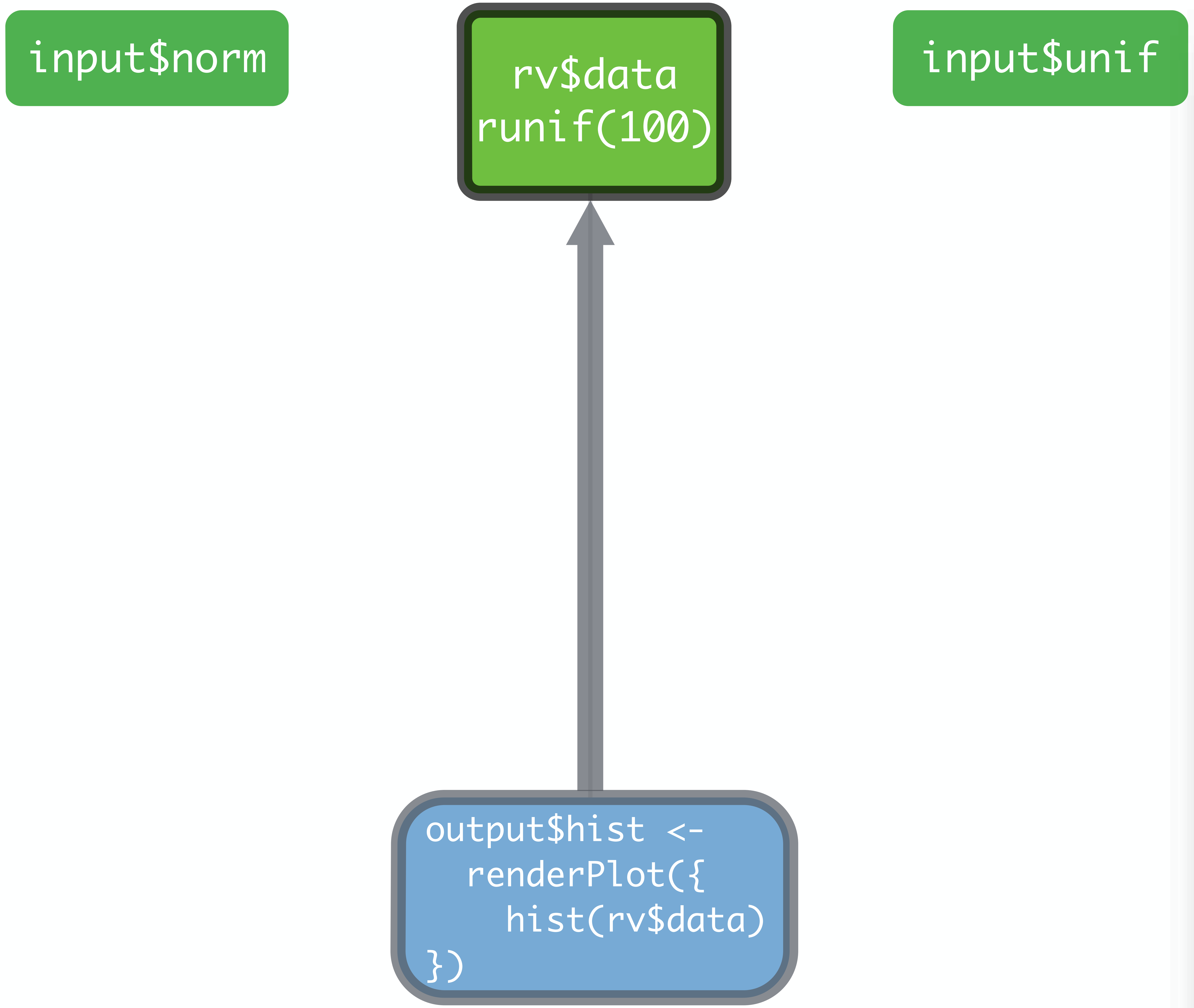
```
output$hist <-
  renderPlot({
    hist(rv$data)
})
```



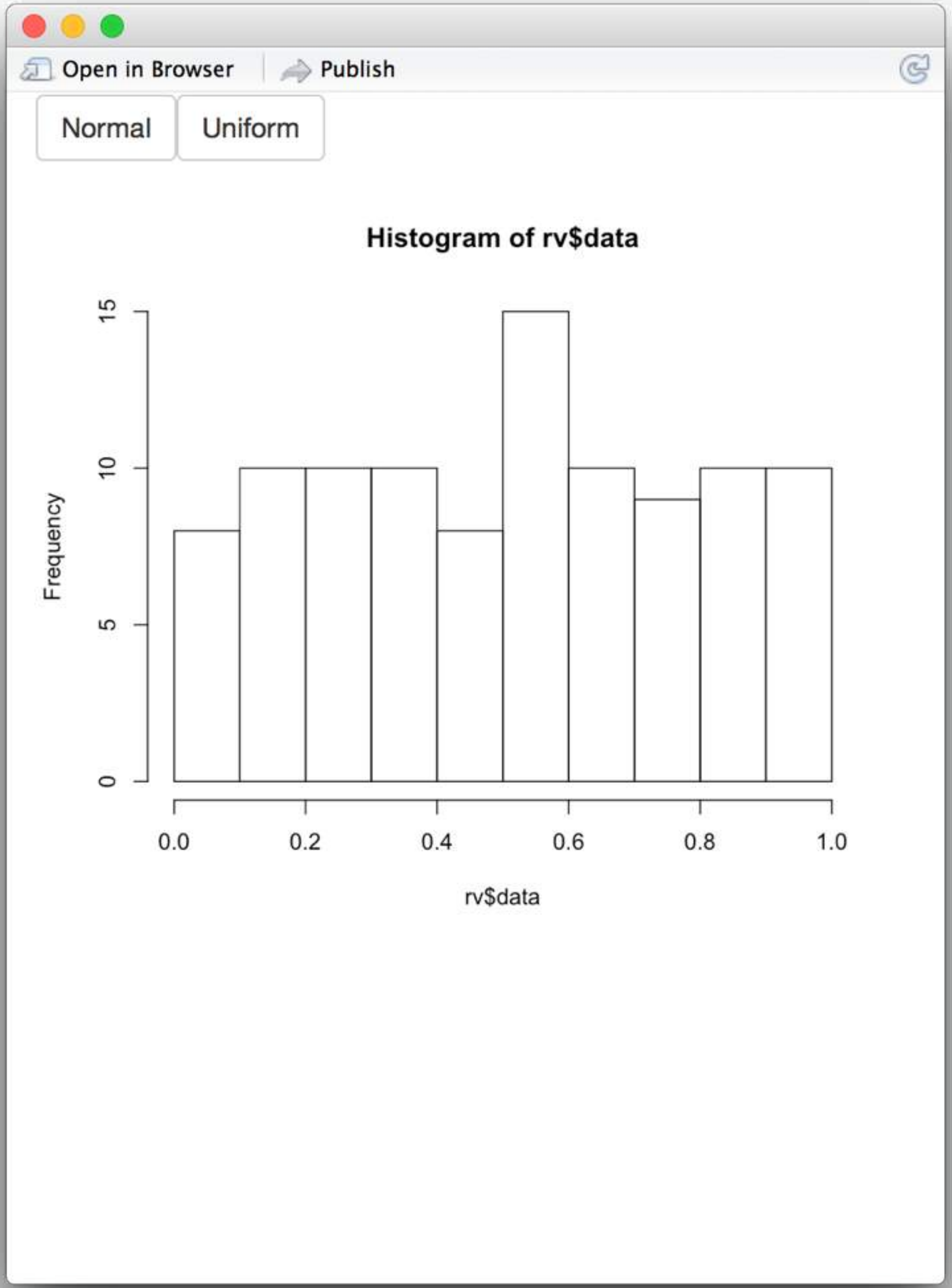


```
output$hist <-
  renderPlot({
    hist(rv$data)
})
```

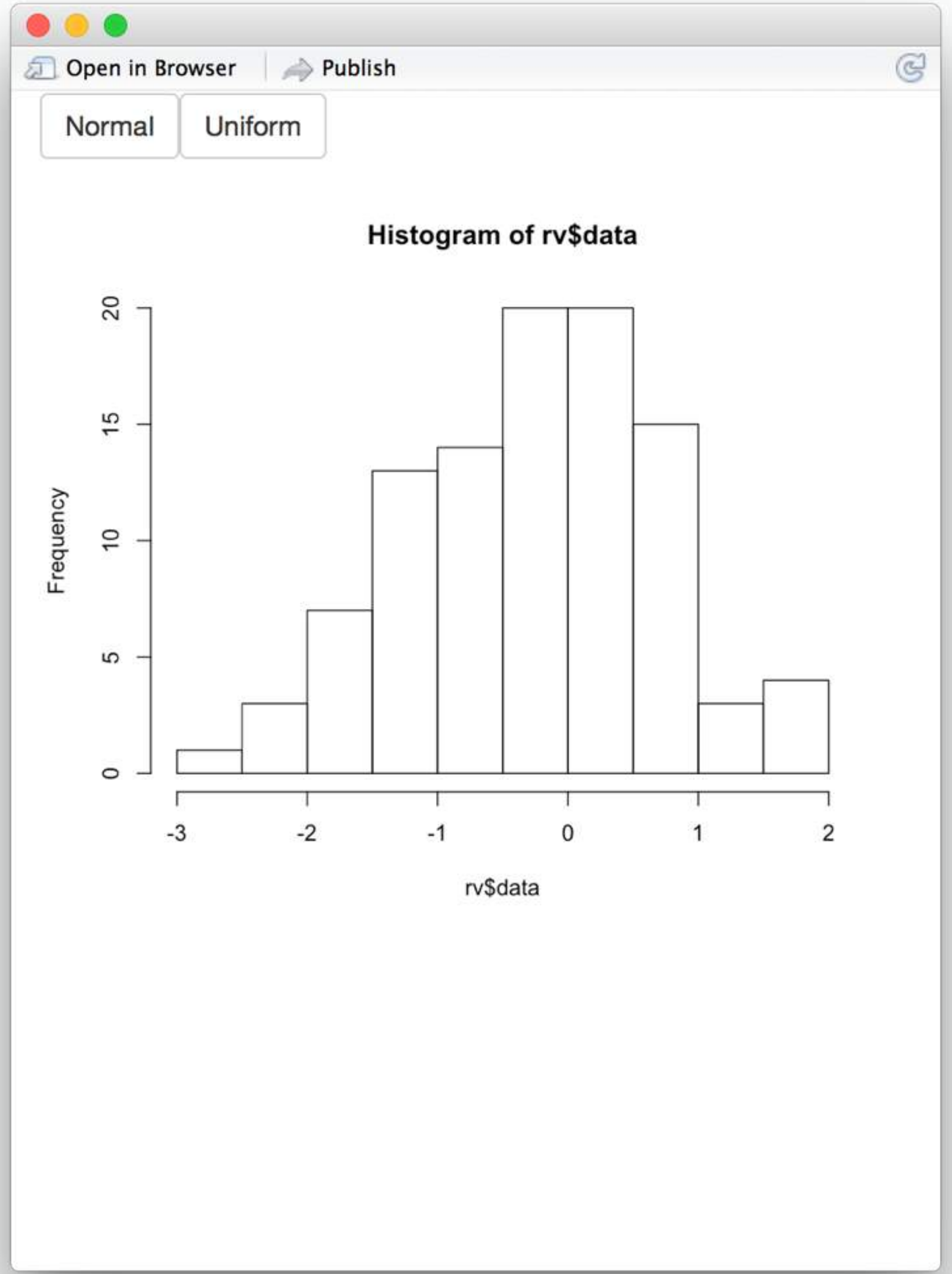




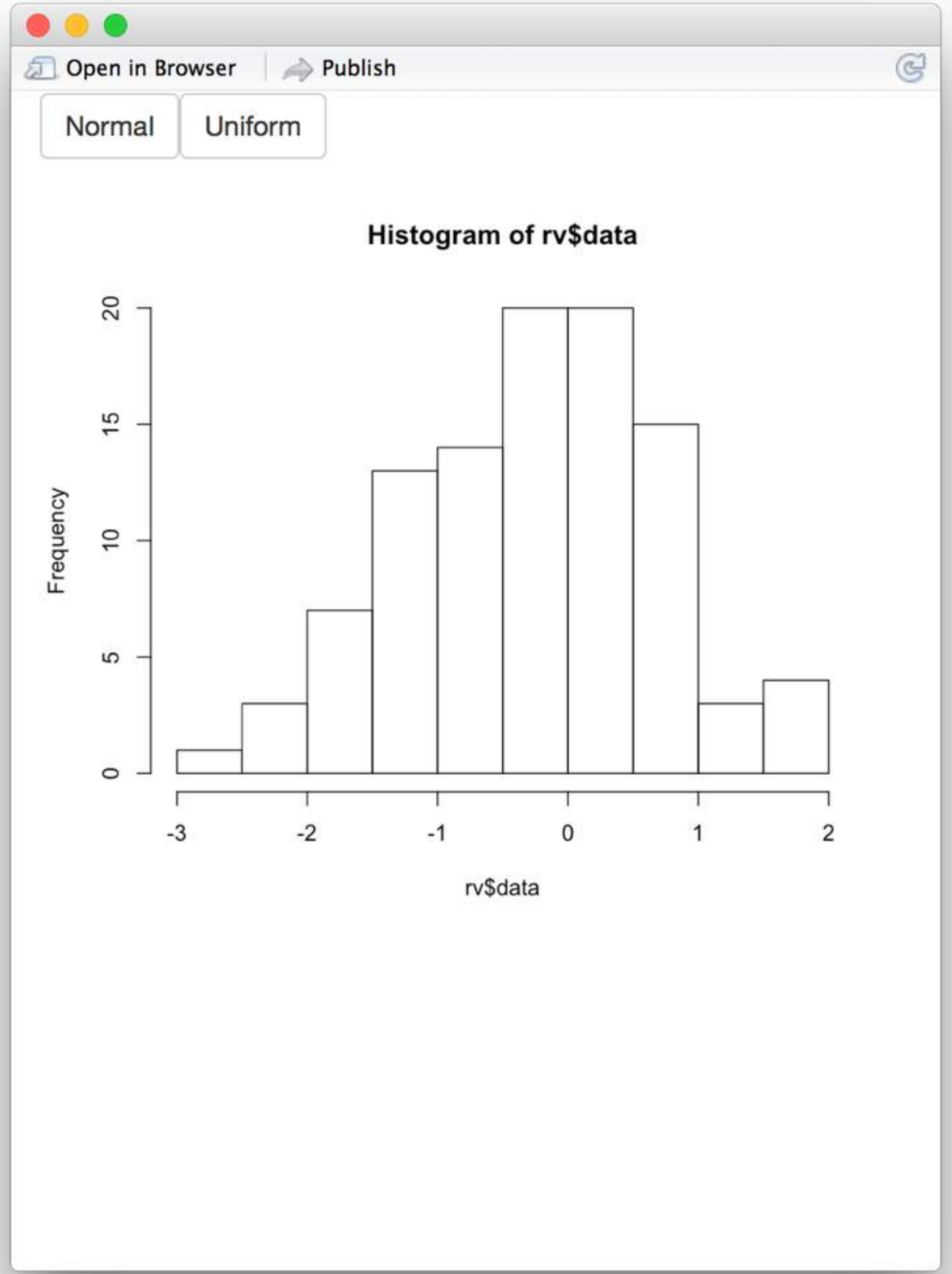
```
output$hist <-
  renderPlot({
    hist(rv$data)
})
```



```r
# 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)
```



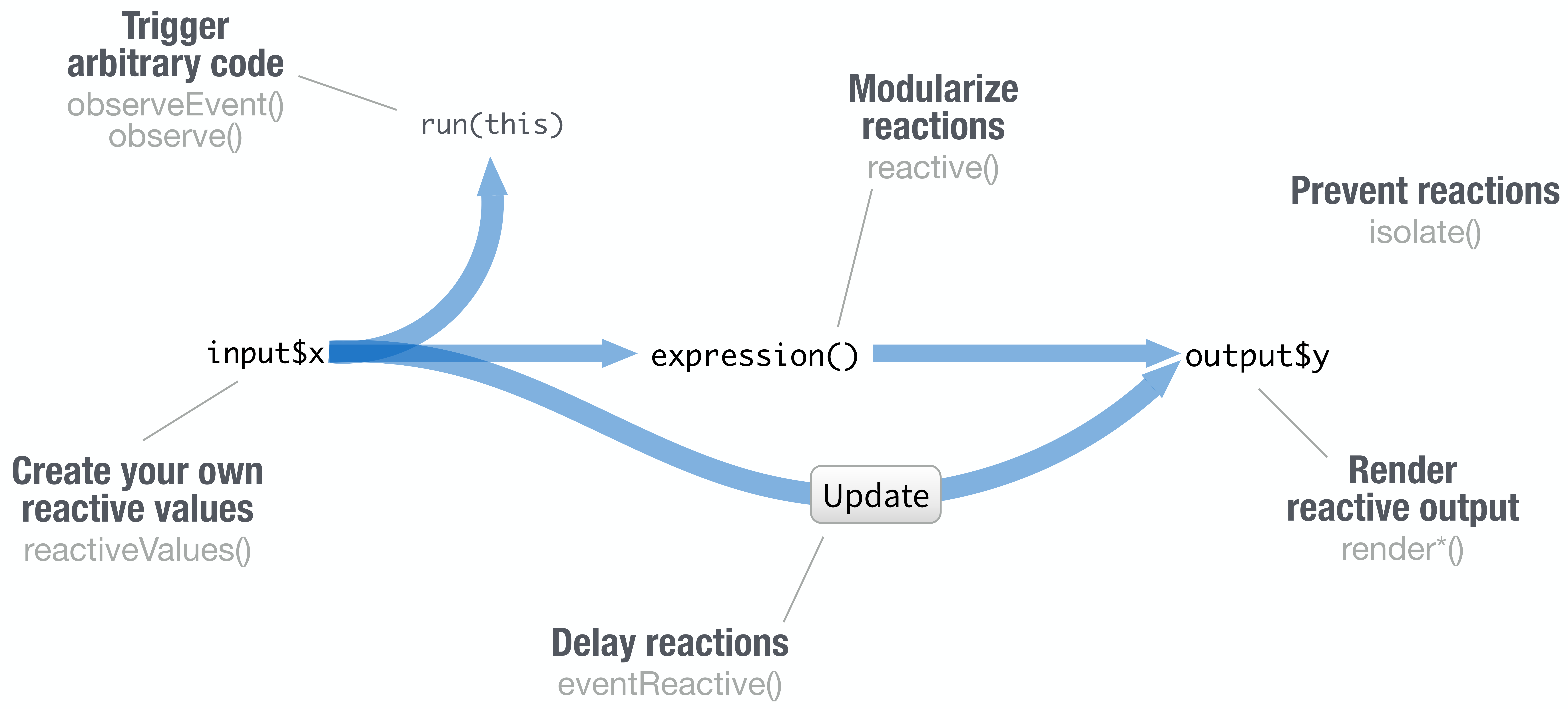
Recap: reactiveValues()

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

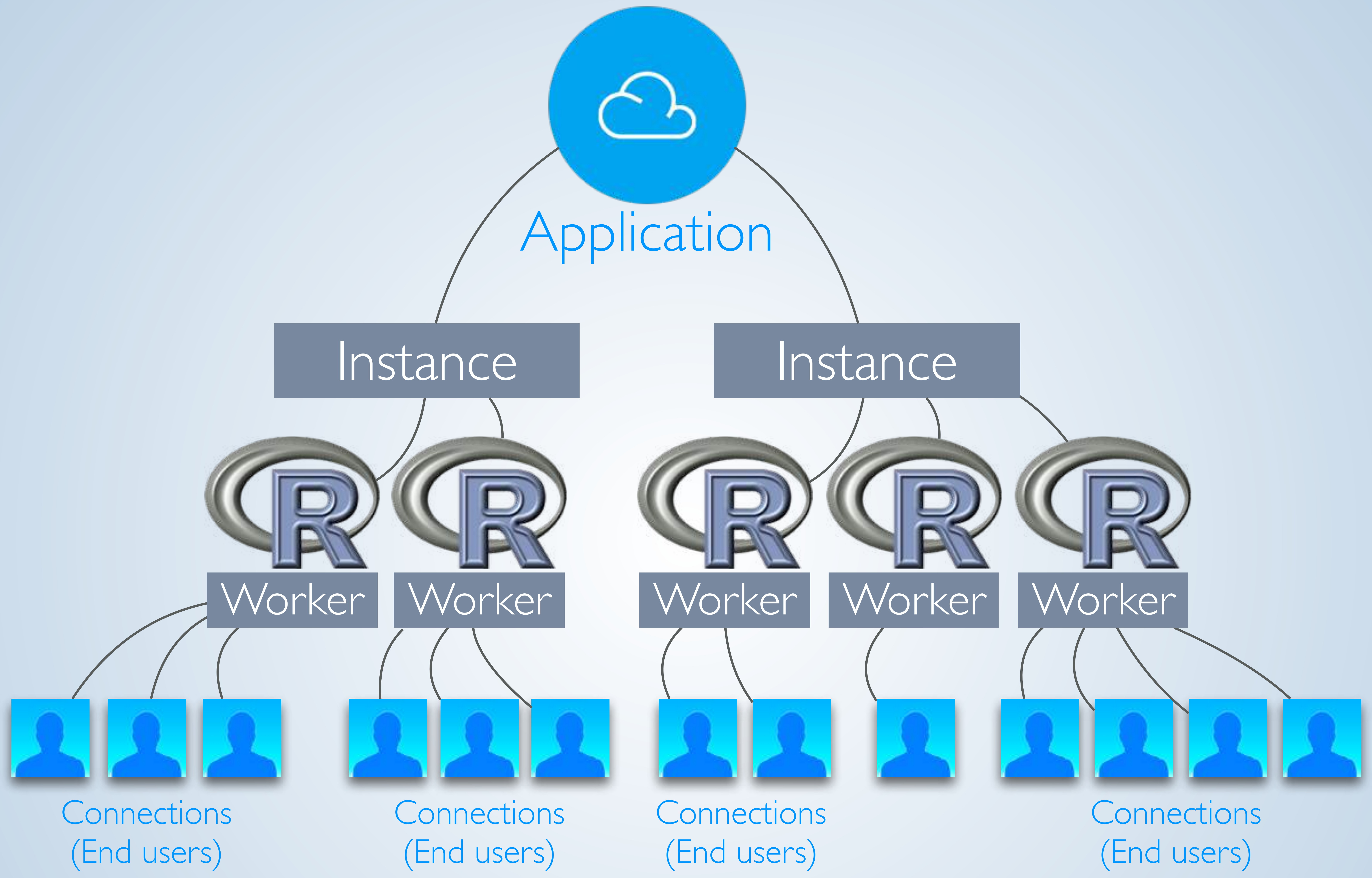
`rv$data <-`

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

You now how to



Parting tips


Application
Instance
Instance
Worker
Worker
Worker
Worker
Worker
Connections (End users)
Connections (End users)
Connections (End users)
Connections (End users)

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)
```

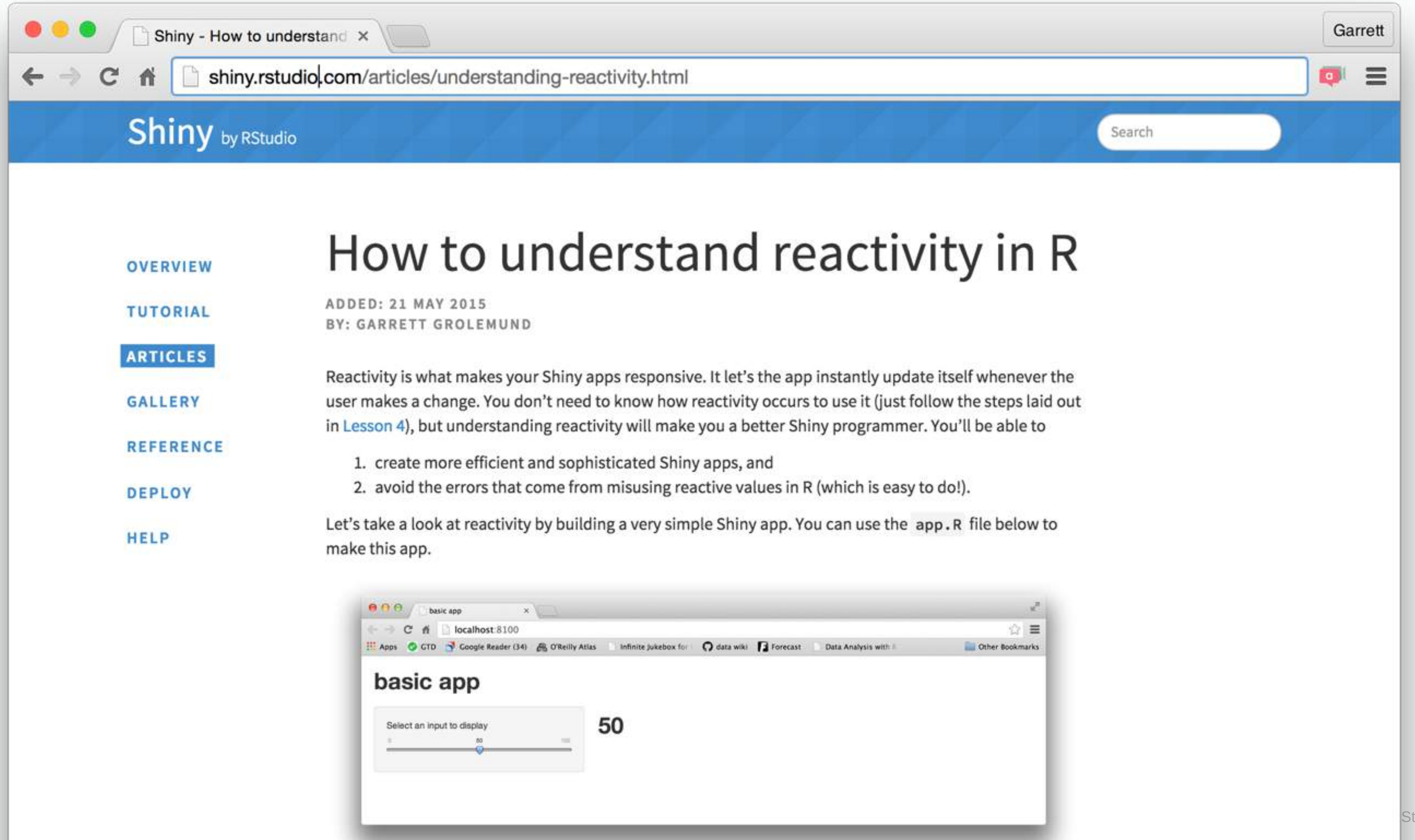
Code outside the server function will be run once per R session (worker)

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


Shiny - How to understand
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 let's 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.

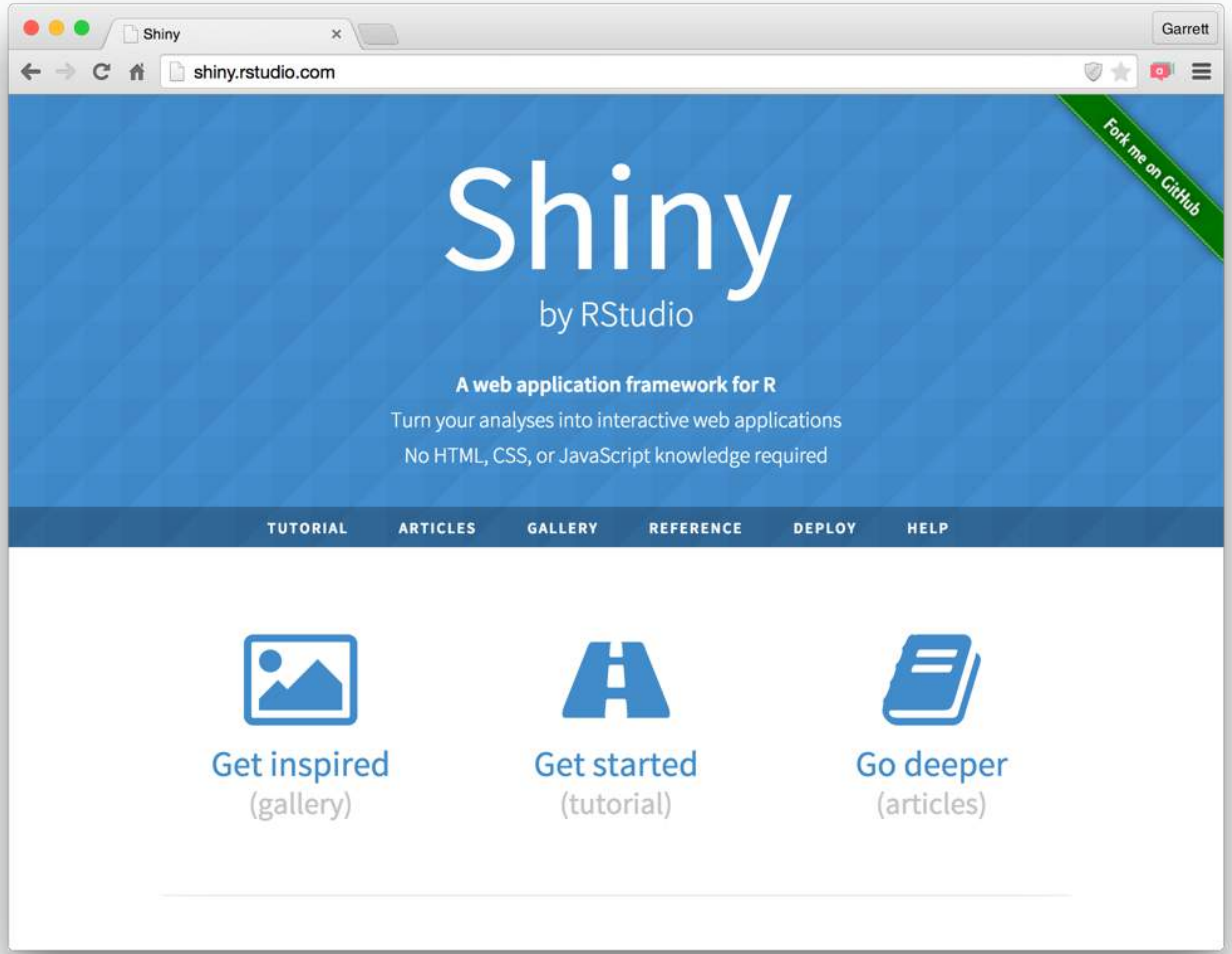
Learn
more

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


Shiny
Garrett
shiny.rstudio.com
Fork me on GitHub
Shiny
by RStudio
A web application framework for R
Turn your analyses into interactive web applications
No HTML, CSS, or JavaScript knowledge required
TUTORIAL
ARTICLES
GALLERY
REFERENCE
DEPLOY
HELP
Get inspired
(gallery)
Get started
(tutorial)
Go deeper
(articles)