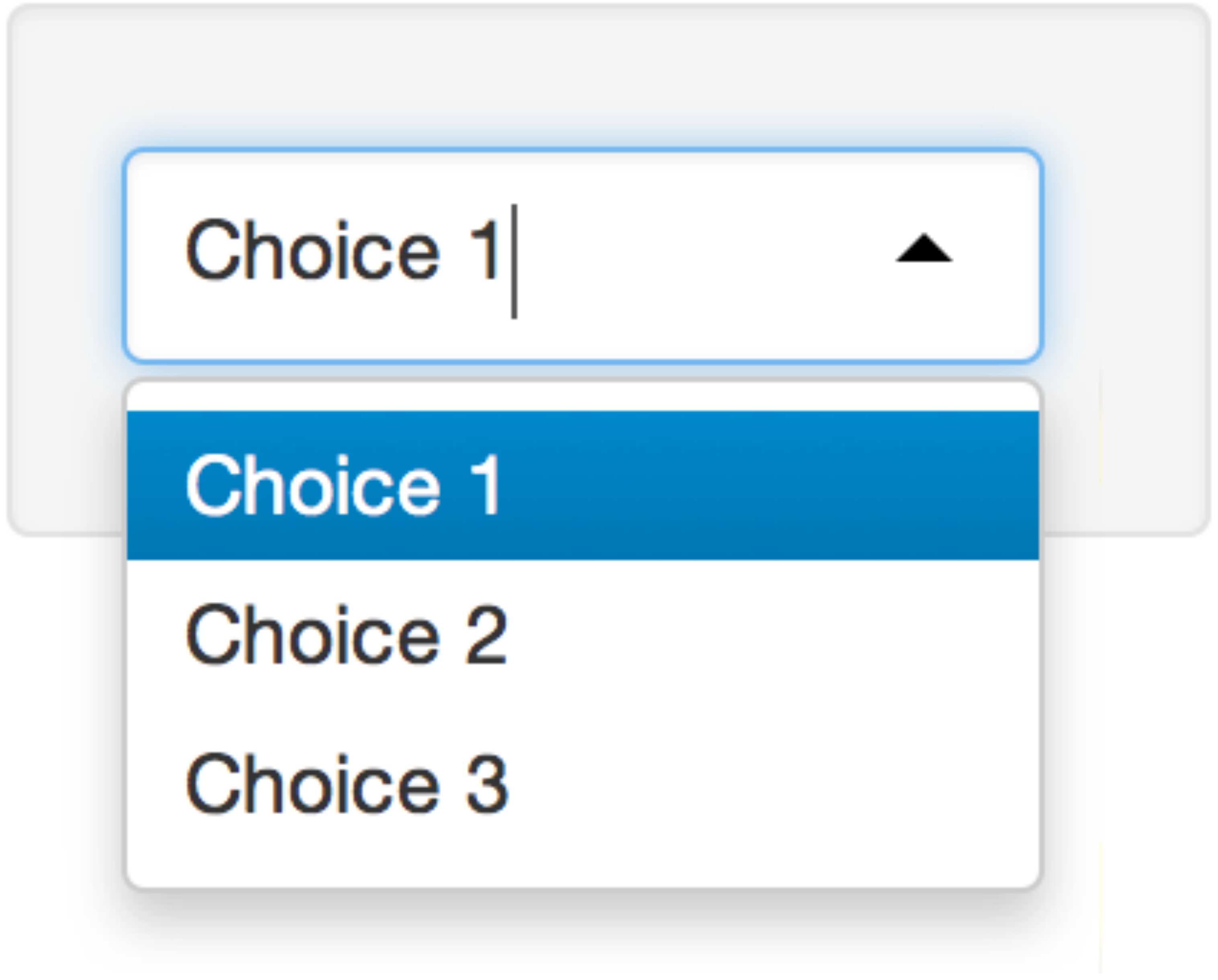
Shiny Interactive Data Analysis

How to build a Shiny App



Dean Young (youngde@reed.edu) and Chester Ismay (cismay@reed.edu)

Slides available at http://tinyurl.com/shinyslides

Learn R

R basics

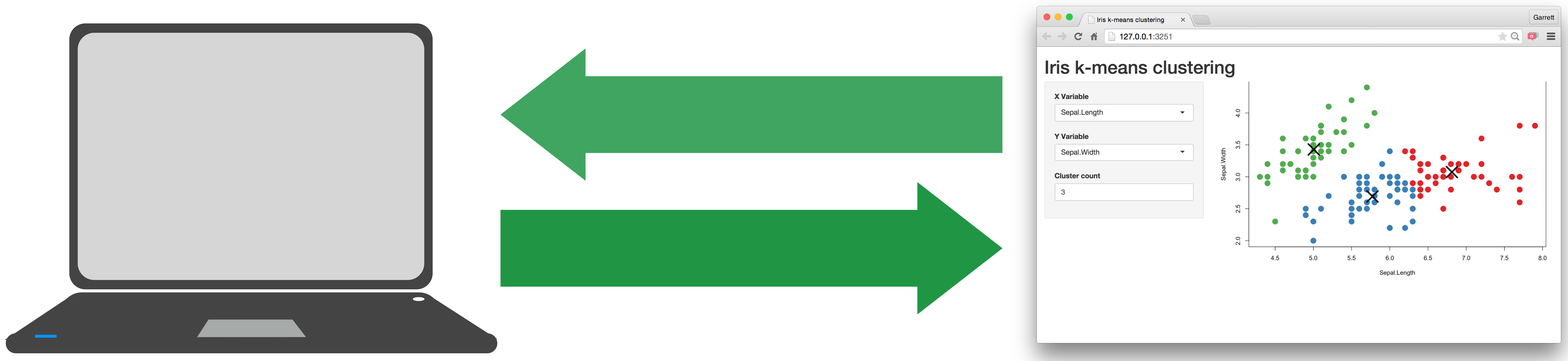
```
# Assignment of values to objects
num_rows <- 10
name <- “Chester”
temp <- c(0, 10, 52, 100)
vec <- rnorm(100)

# Simple function call
mean(temp)
[1] 40.5
```

Writing functions

```
# Function definition
cube.it <- function(x) {
    cube <- x * x * x
    return(cube)
}

# Function call
cube.it(7)
[1] 343
```

Understand the architecture

Every Shiny app is maintained by a computer running R



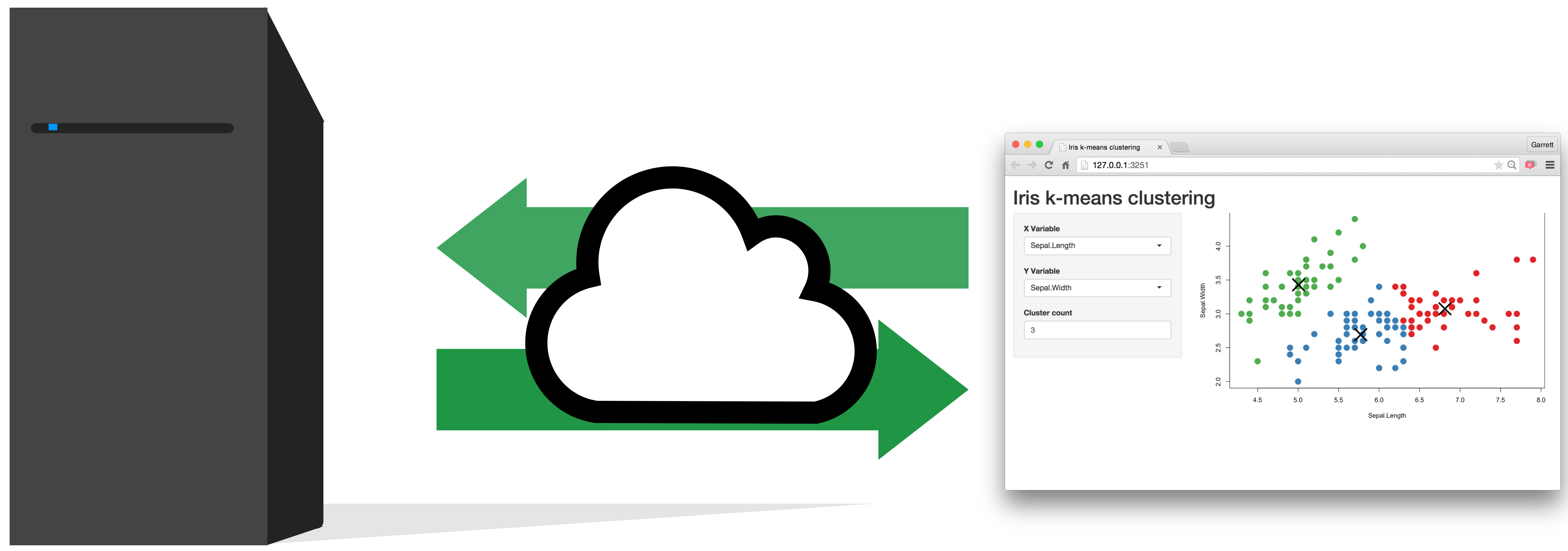
Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

Every Shiny app is maintained by a computer running R

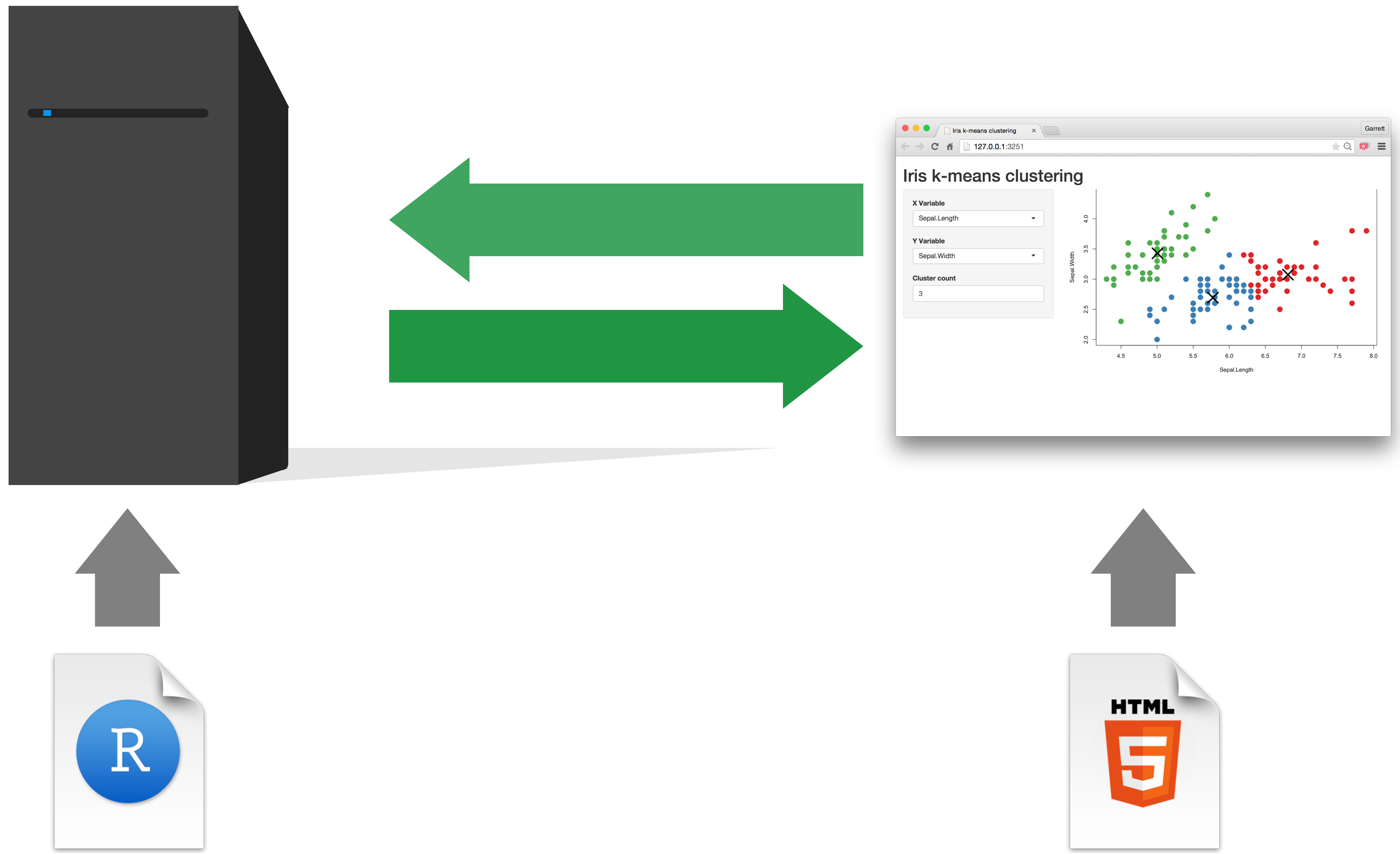


Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3



Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

Server Instructions

User Interface (UI)

Use the template

RStudio

App template

The shortest viable shiny app

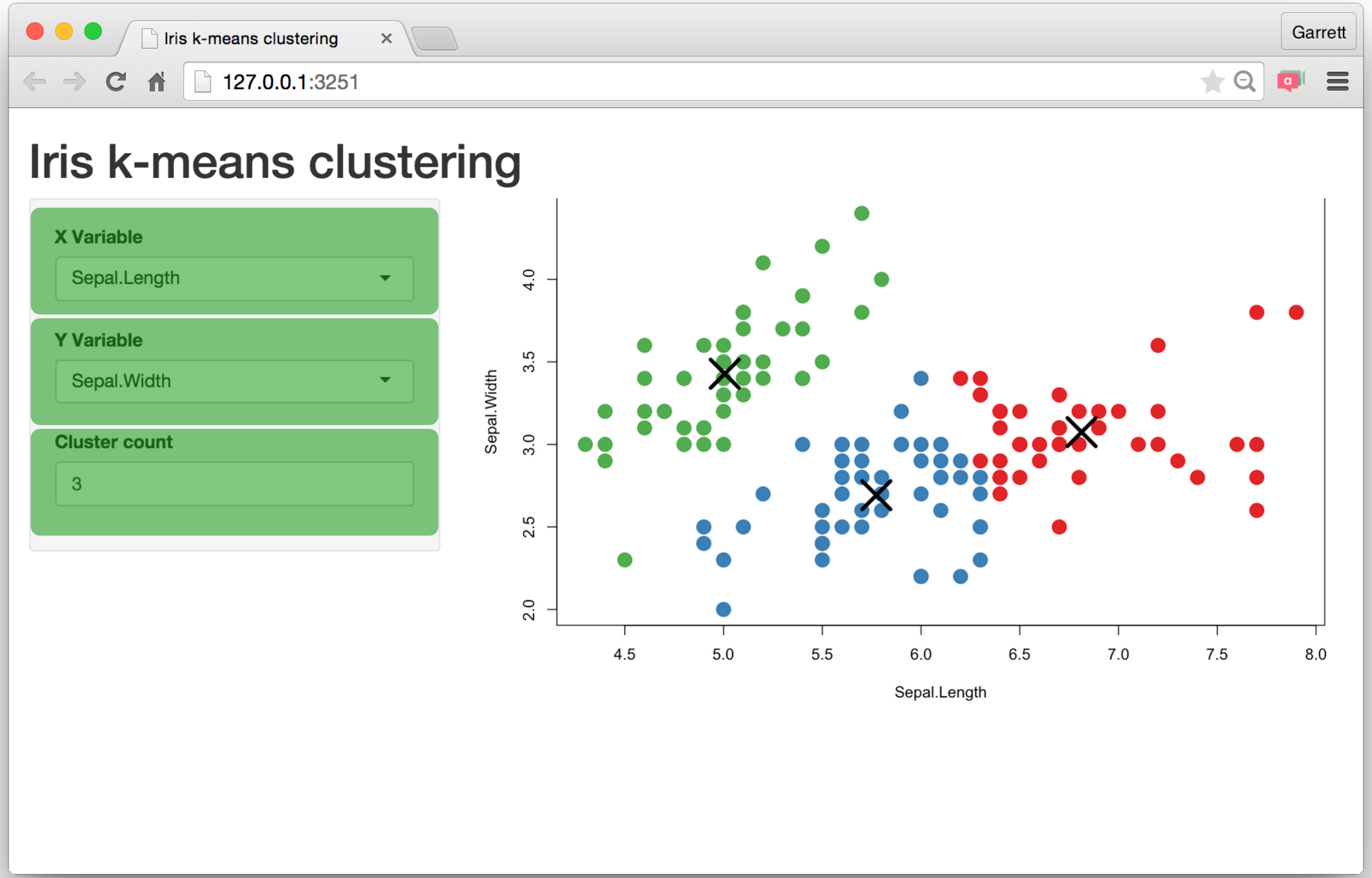
```
library(shiny)
ui <- fluidPage()

server <- function(input, output) {}

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

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

```
library(shiny)
ui <- fluidPage("Hello World")

server <- function(input, output) {}

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

Inputs and Outputs

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



Iris k-means clustering

X Variable

Sepal.Length

Y Variable

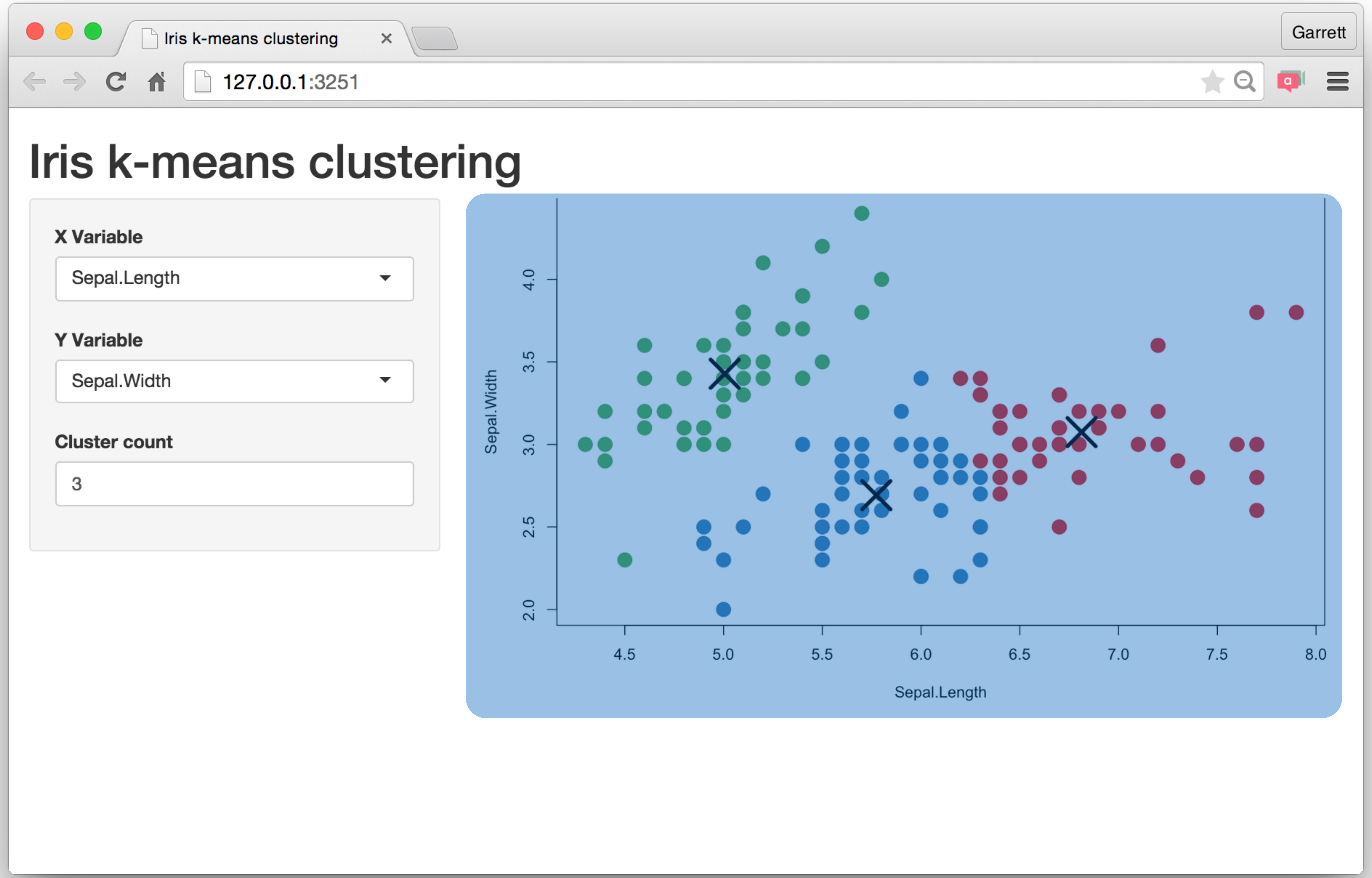
Sepal.Width

Cluster count

3

RStudio

Build your app around inputs and outputs


Iris k-means clustering

127.0.0.1:3251

Garrett

Iris k-means clustering

X Variable
Sepal.Length

Y Variable
Sepal.Width

Cluster count
3

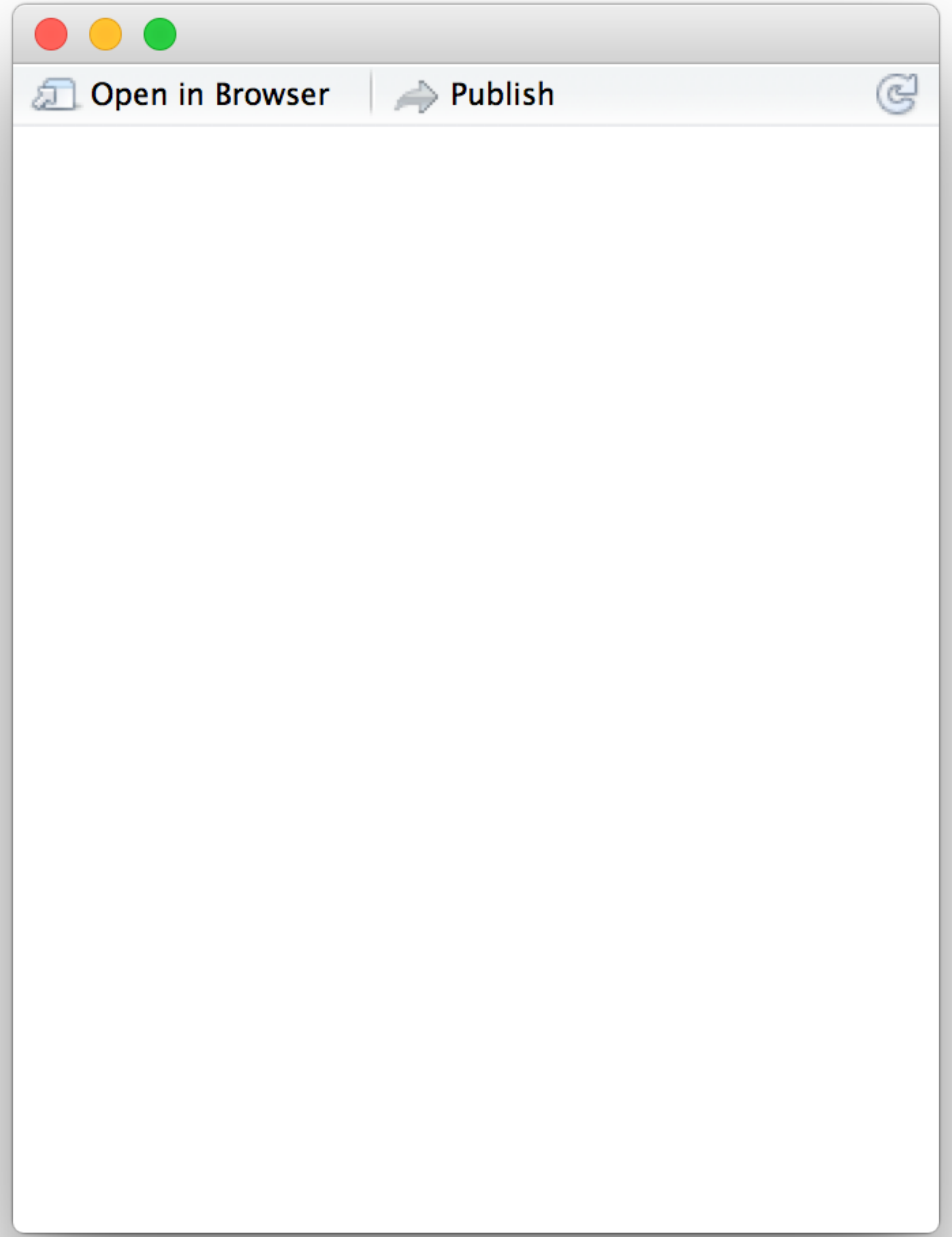
Add elements to your app as arguments to `fluidPage()`

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

Inputs

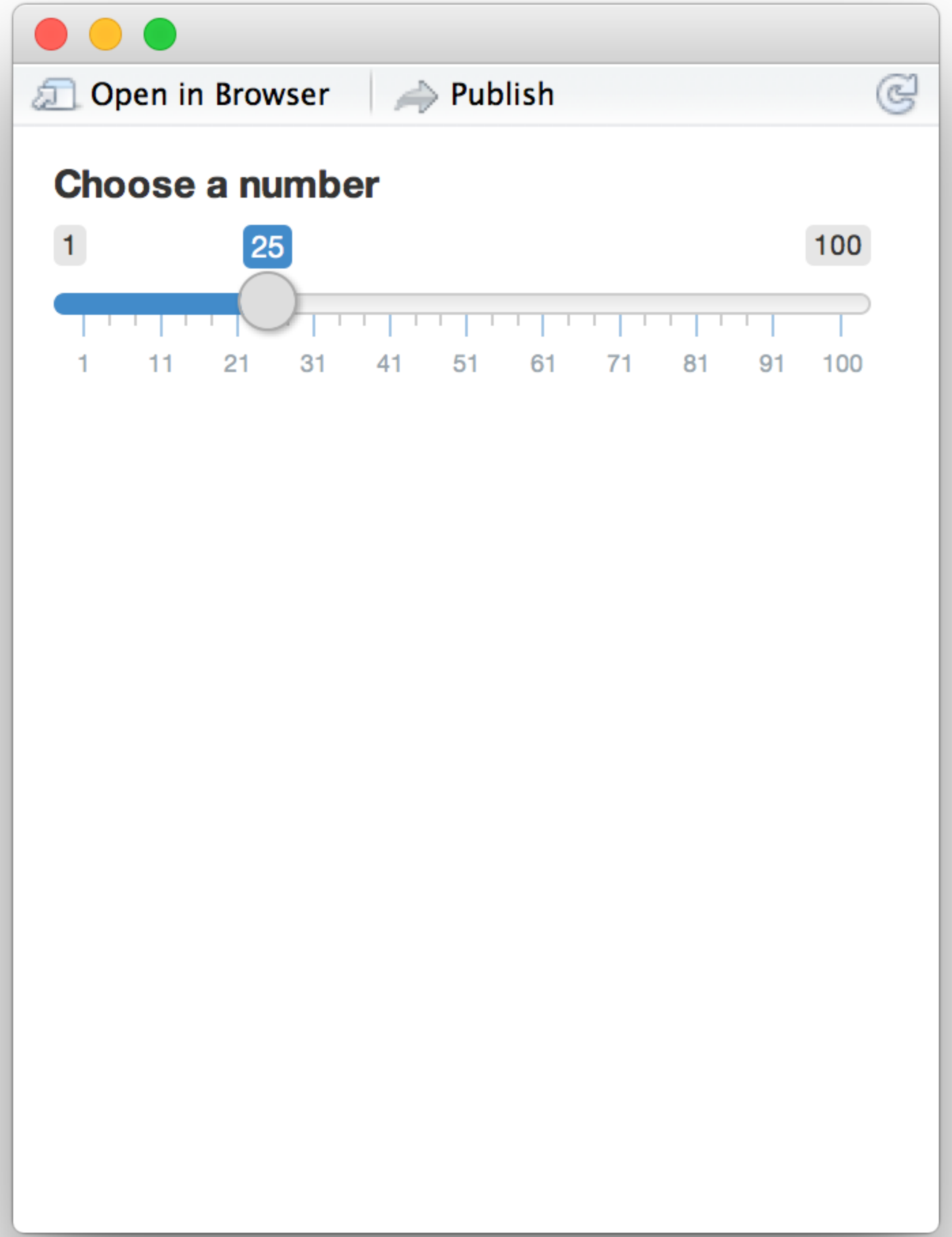
Create an input with an input function.

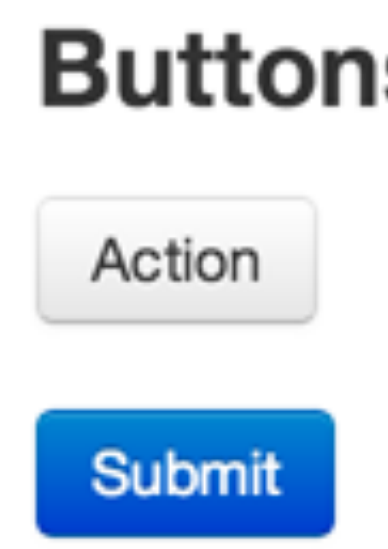
```
library(shiny)
ui <- fluidPage(



)

server <- function(input, output) {}

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



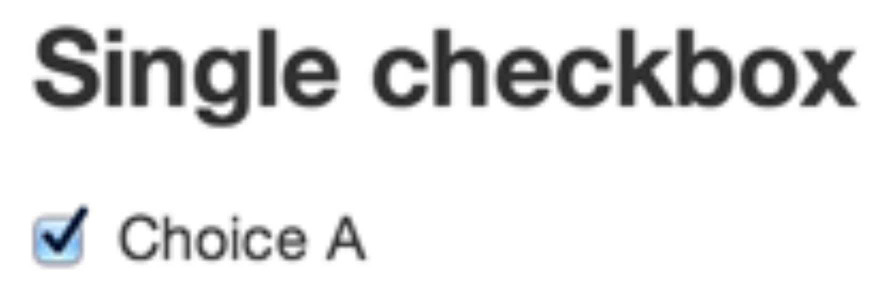
Create an input with an input function.

```
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)
```

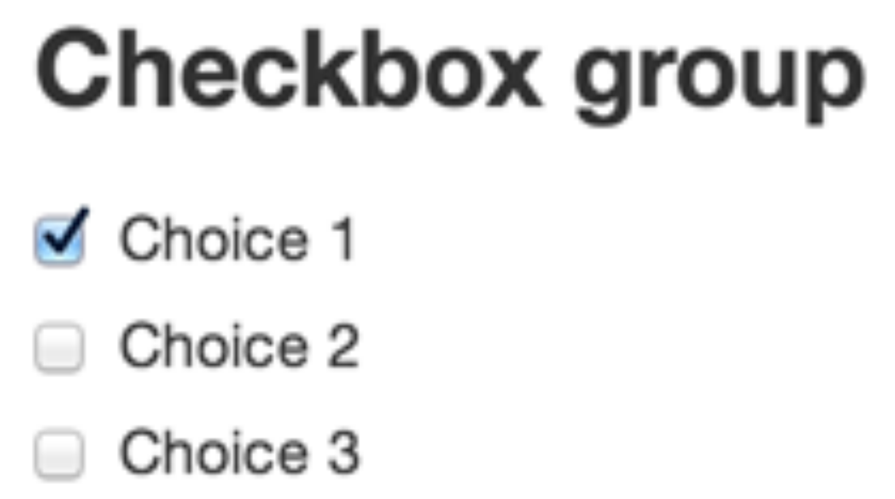




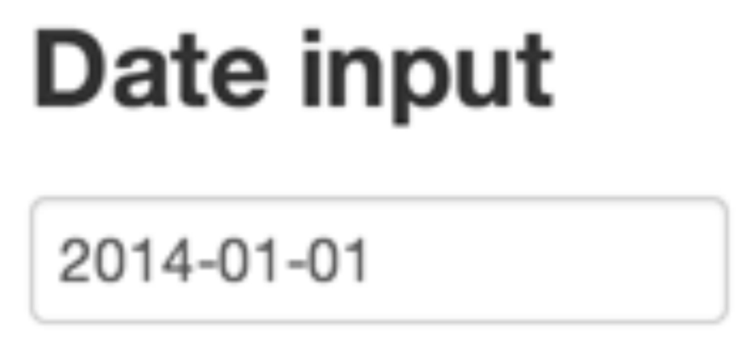
`actionButton()`
`submitButton()`



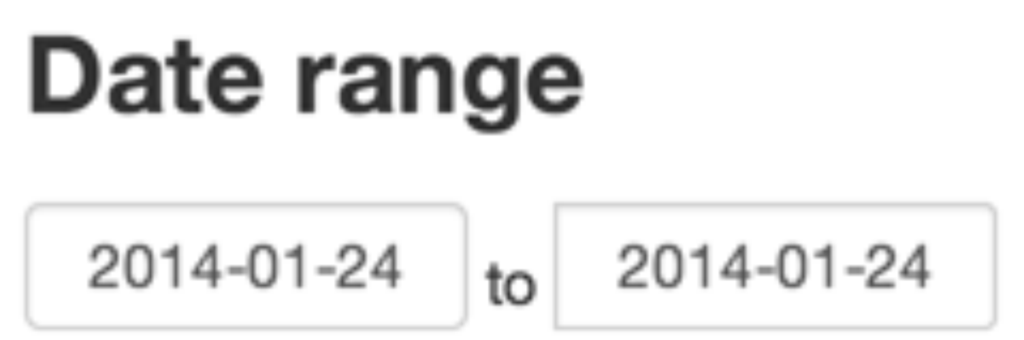
`checkboxInput()`



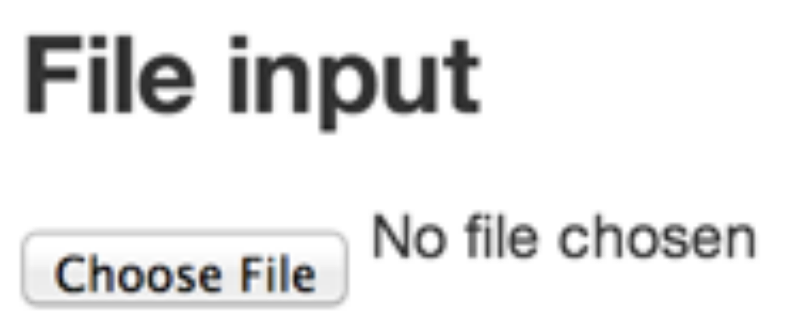
`checkboxGroupInput()`



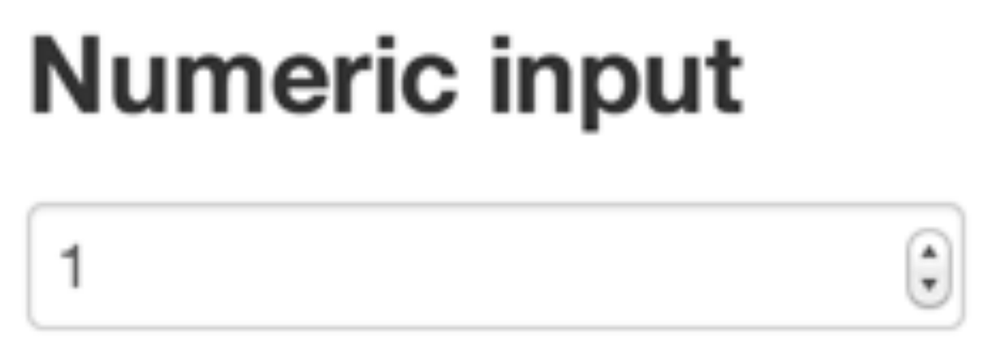
`dateInput()`



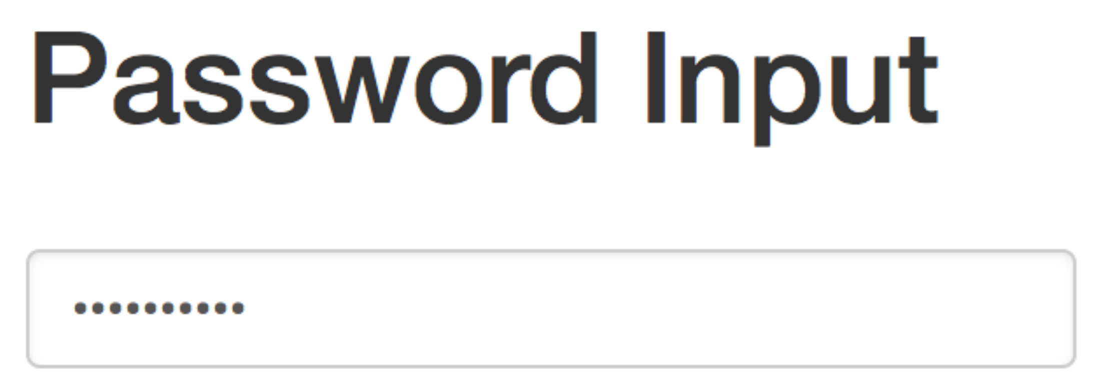
`dateRangeInput()`



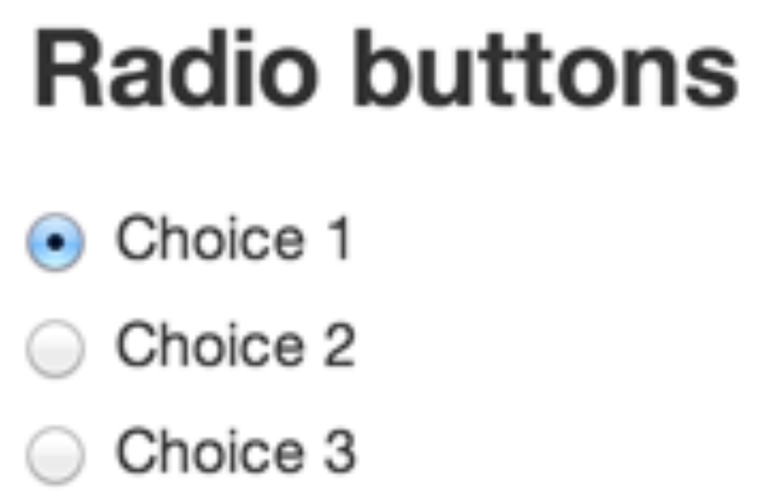
`fileInput()`



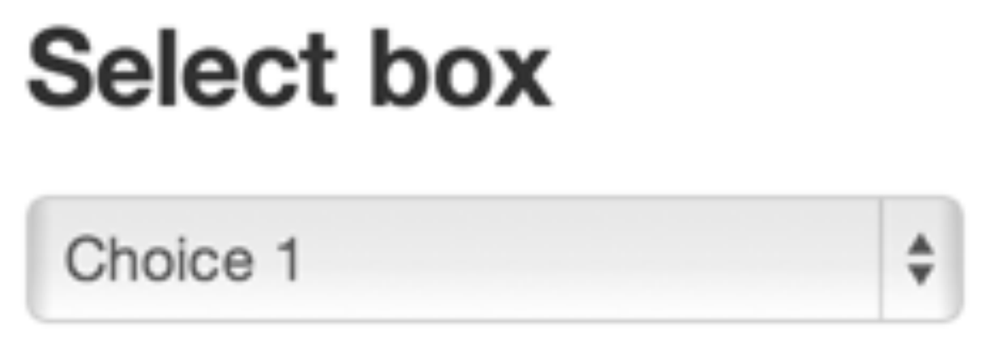
`numericInput()`



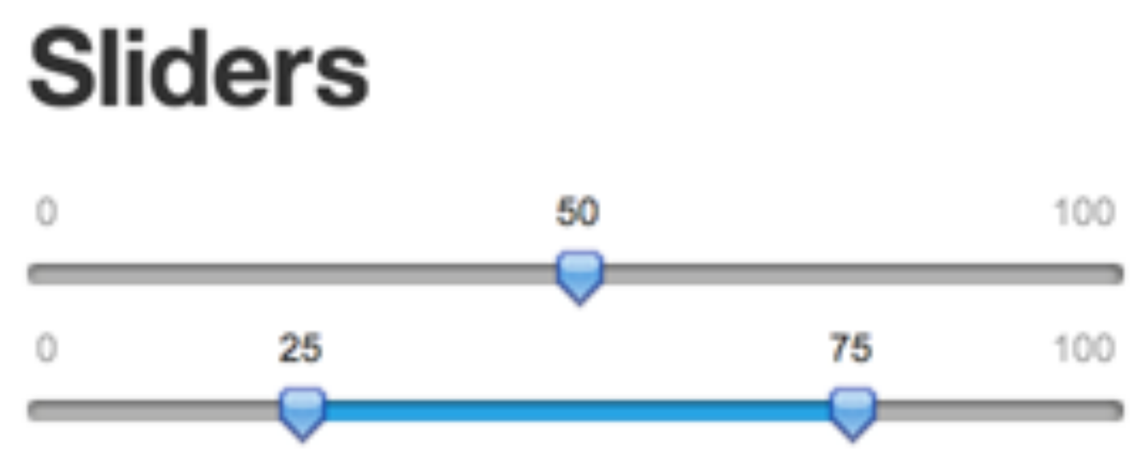
`passwordInput()`



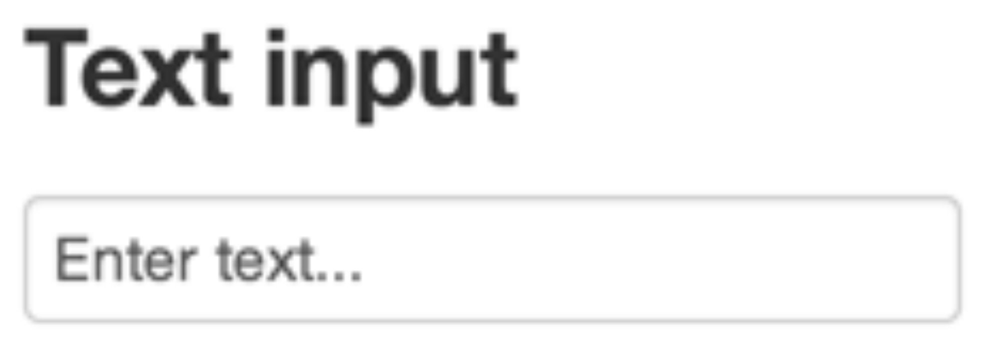
`radioButtons()`



`selectInput()`

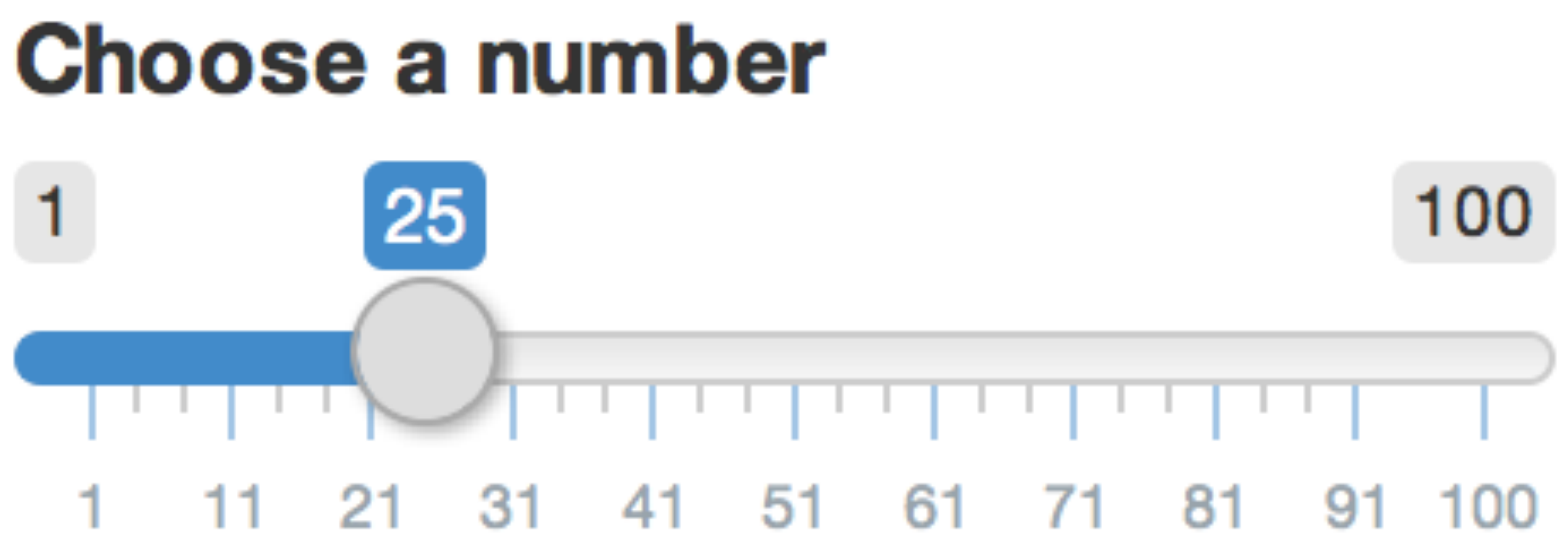


`sliderInput()`



`textInput()`

Syntax



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

input name (for internal use)

Notice: Id not ID

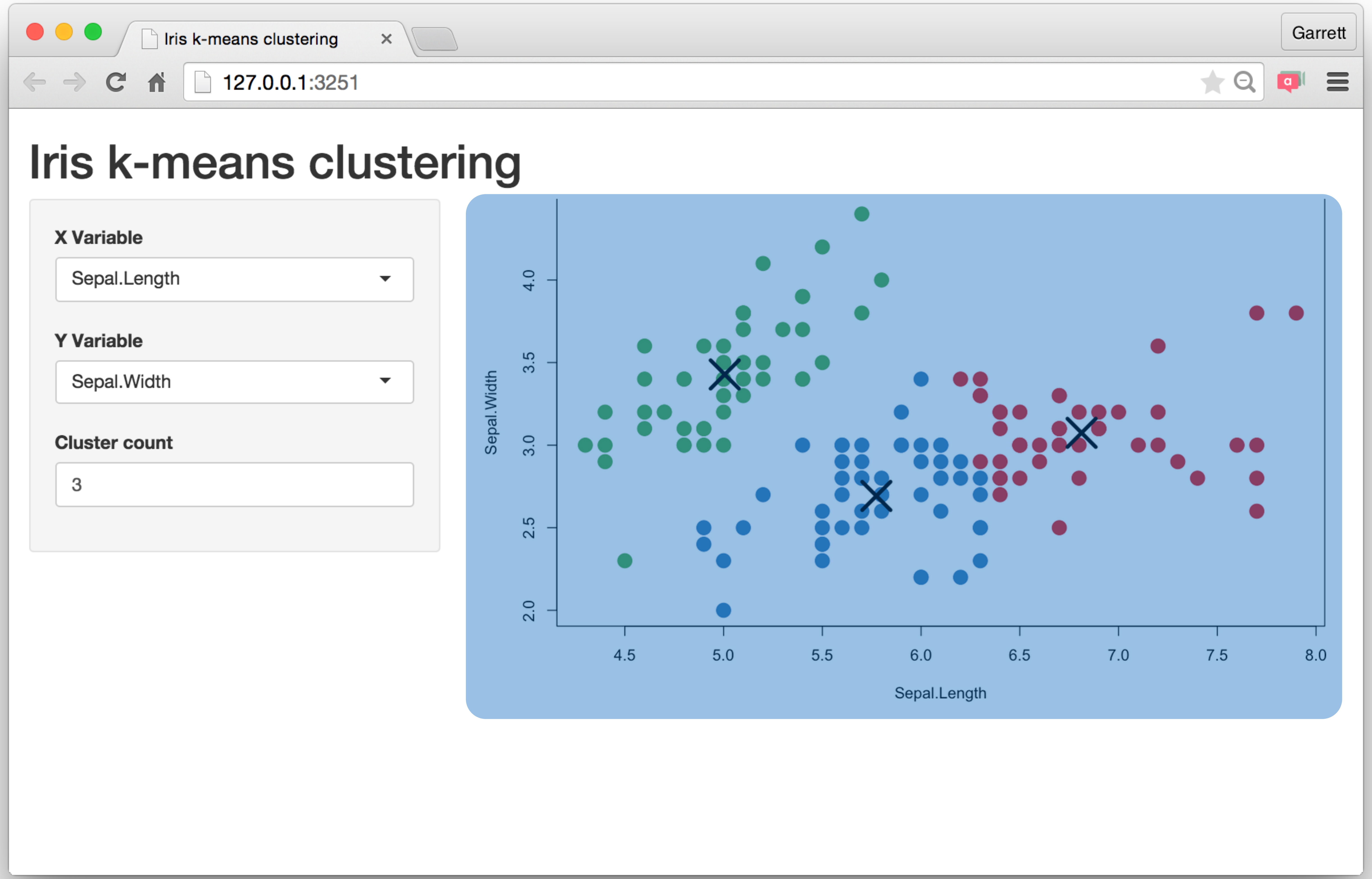
label to display

input specific arguments

`?sliderInput`

Outputs

Build your app around inputs and outputs



Iris k-means clustering

127.0.0.1:3251

Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3

RStudio

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(outputId = "hist")
```

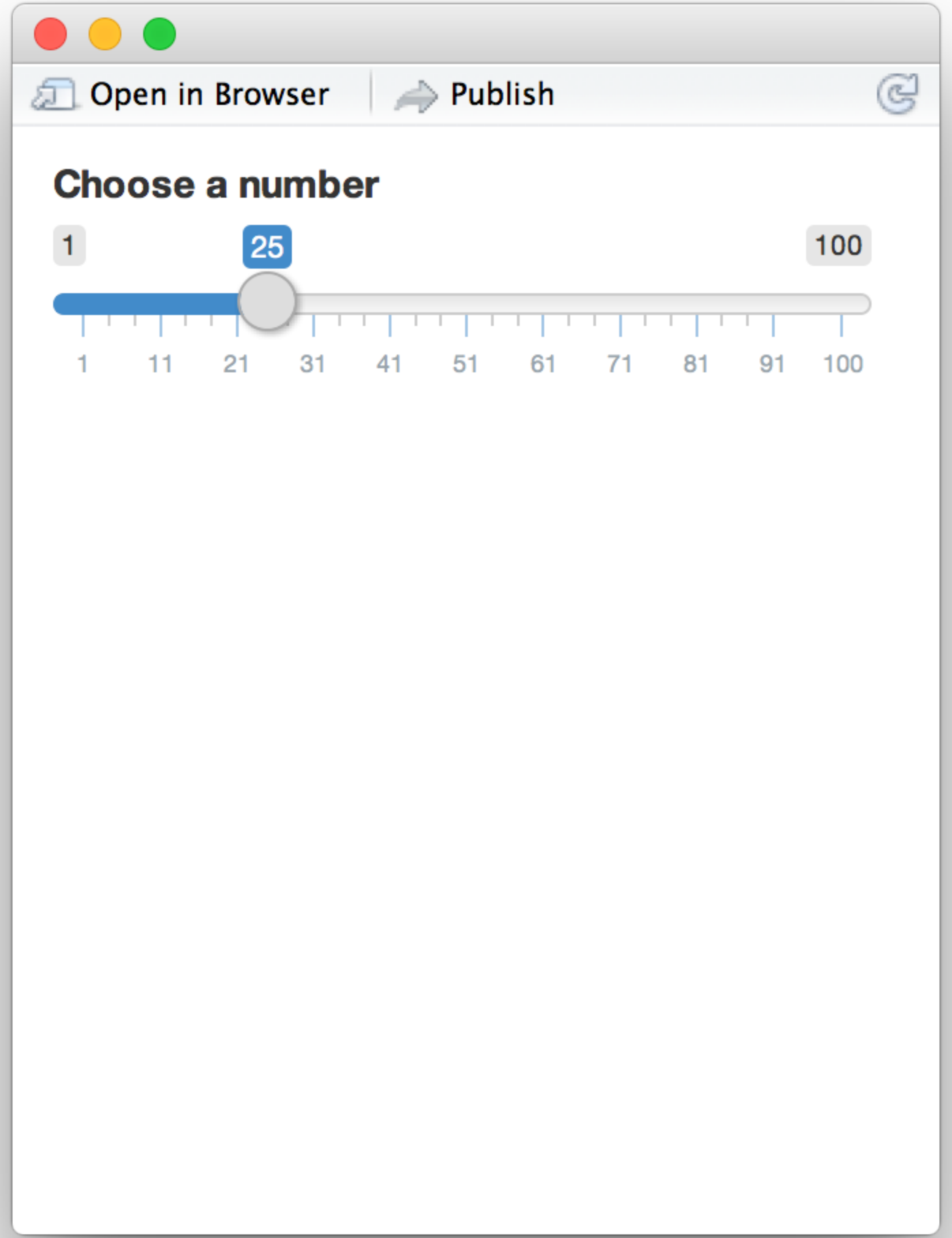
the type of output to display

name to give to the output object

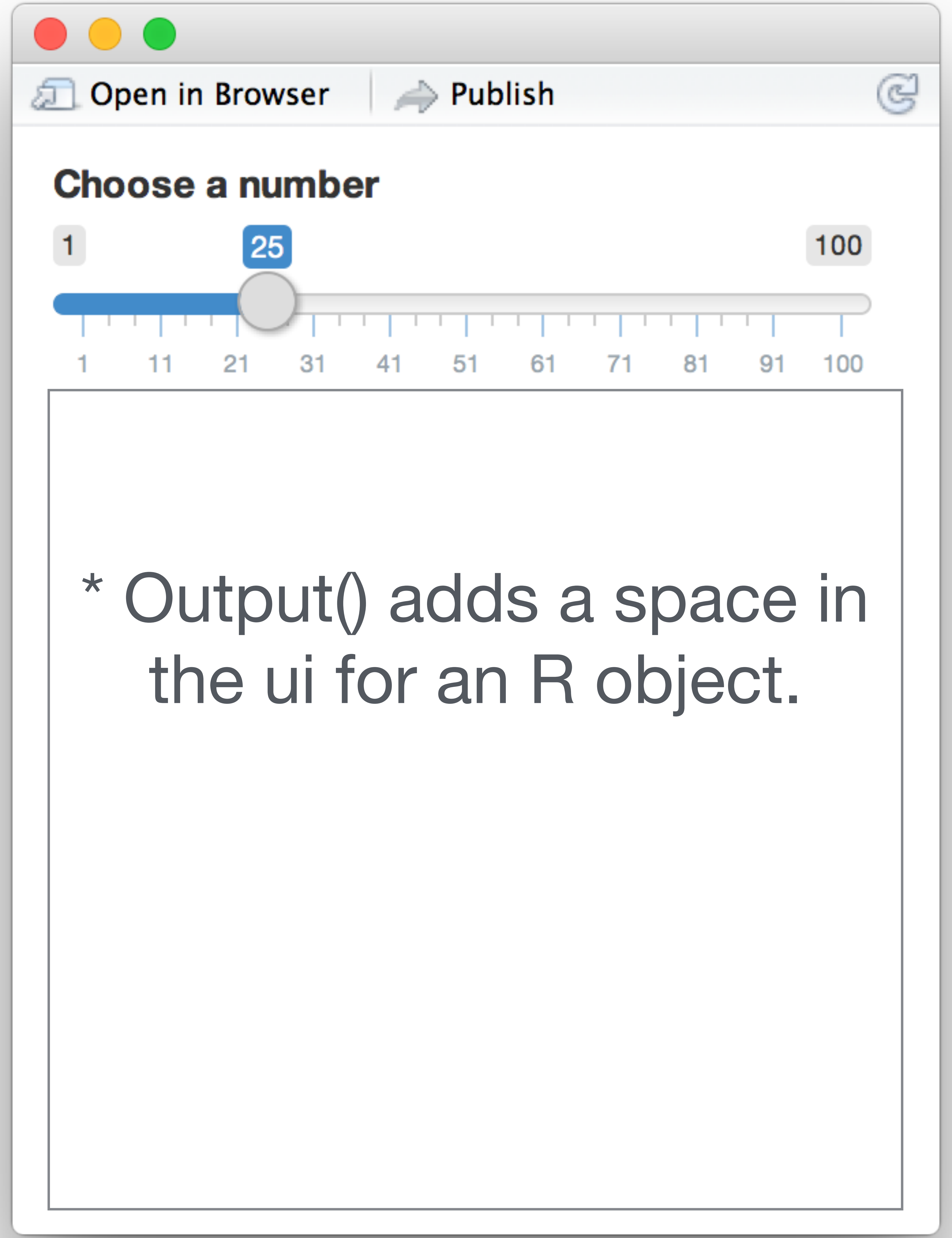
```r
library(shiny)

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

```r
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)
```

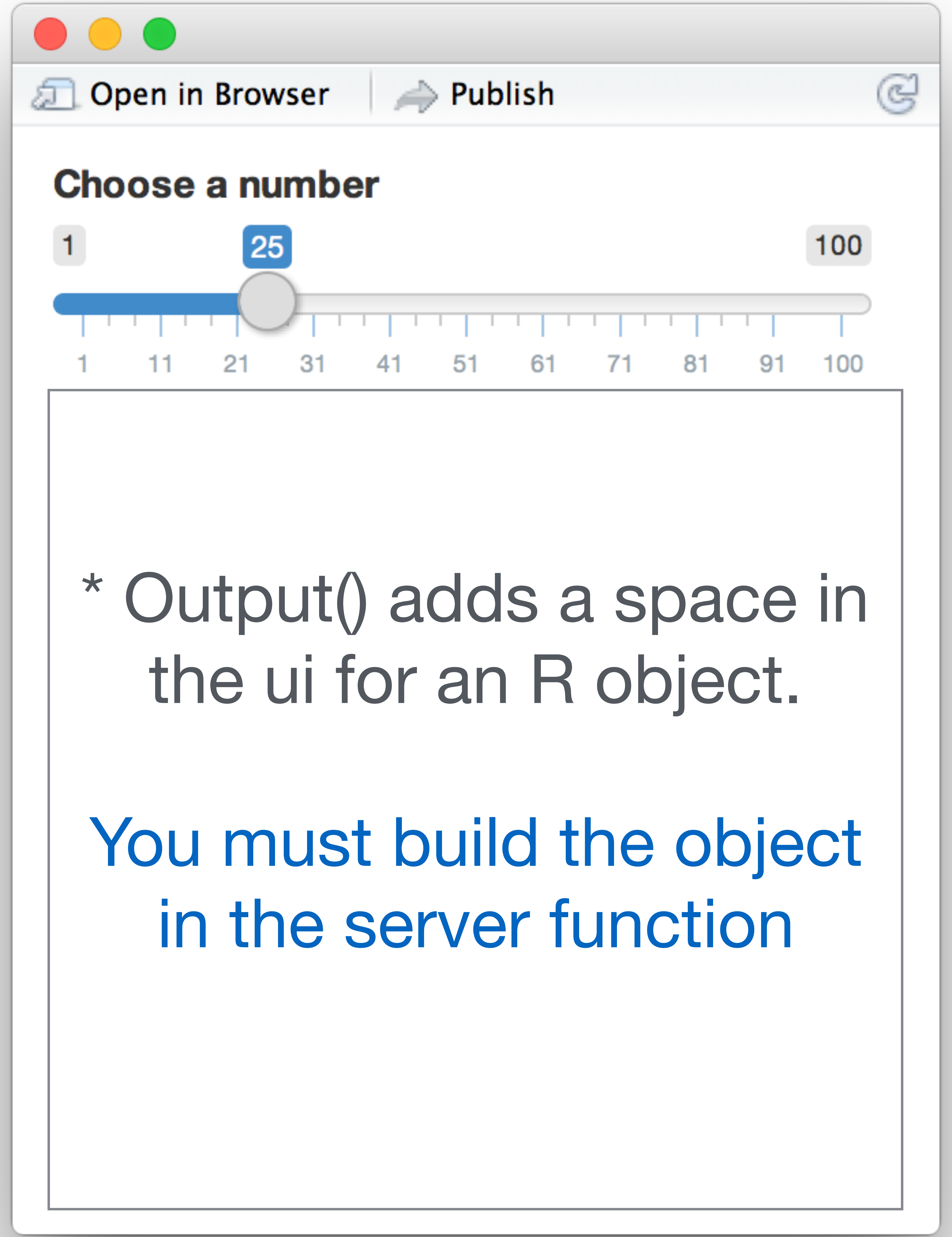


```r
library(shiny)

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.

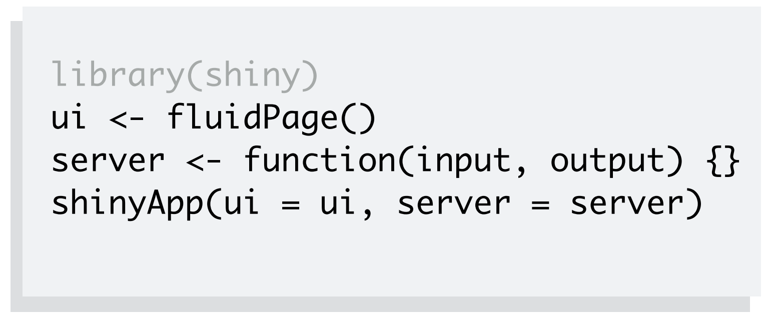
```r
library(shiny)

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

Recap

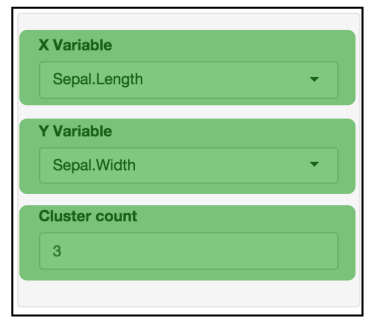


```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

Begin each app with the template

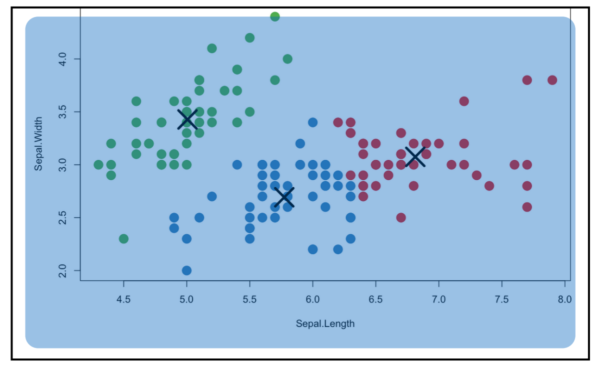
Hello World

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

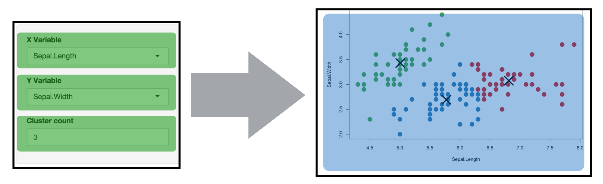


X Variable
Sepal.Length
Y Variable
Sepal.Width
Cluster count
3

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



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



Use the server function to assemble inputs into outputs

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$

```r
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 (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(vec) })
```

type of object to build

code block that builds the object

RStudio

2

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

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

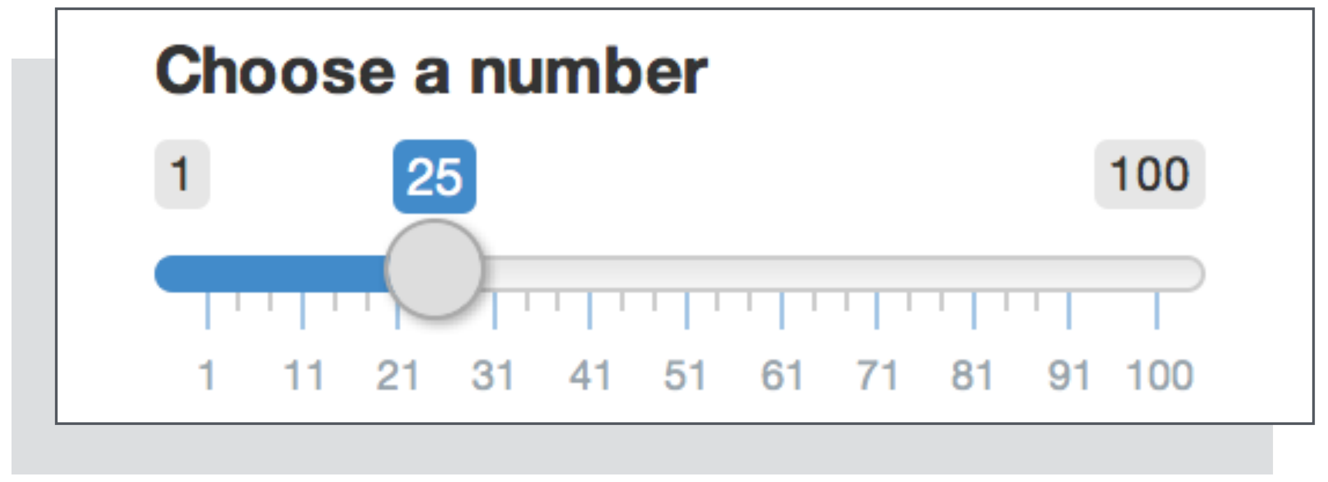
3

Use **input** values with input$

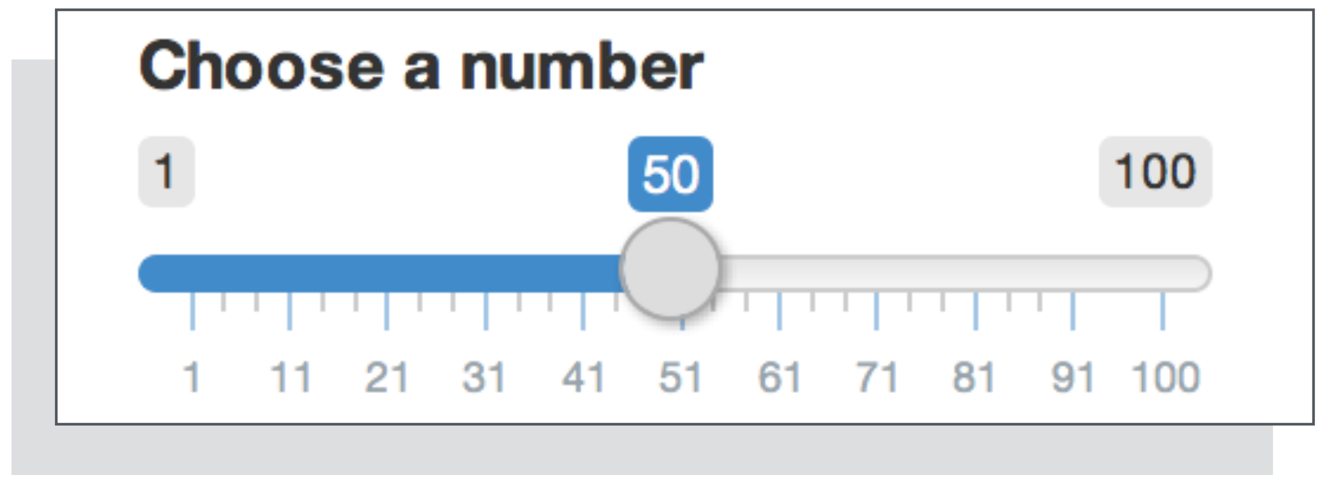
```
sliderInput(inputId = "num",…)

                 input$num
```

Input values

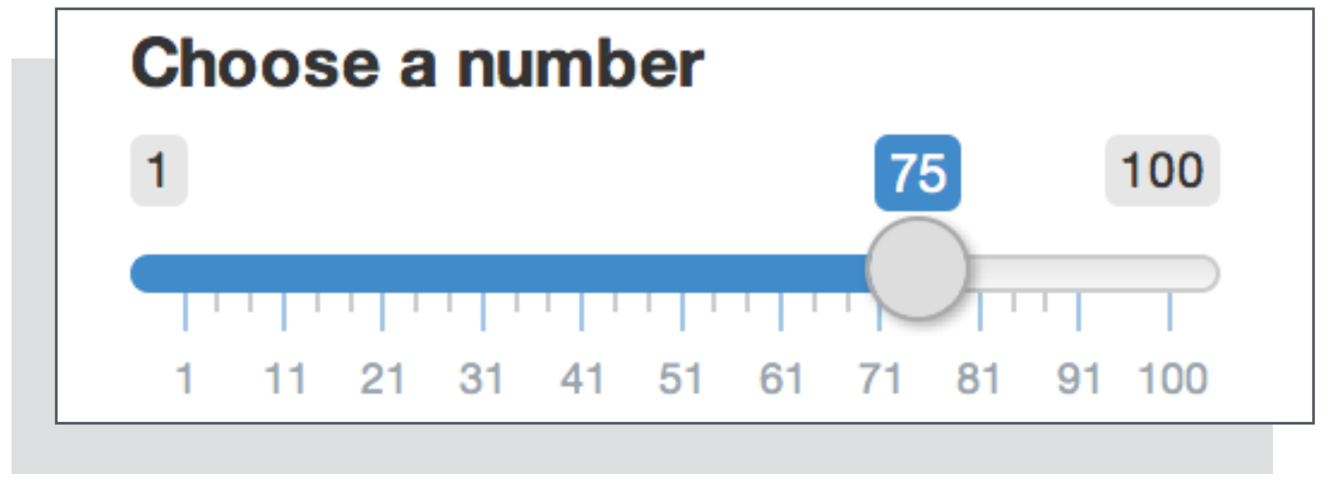
The input value changes whenever a user changes the input.



```
input$num = 25
```



```
input$num = 50
```



```
input$num = 75
```

3

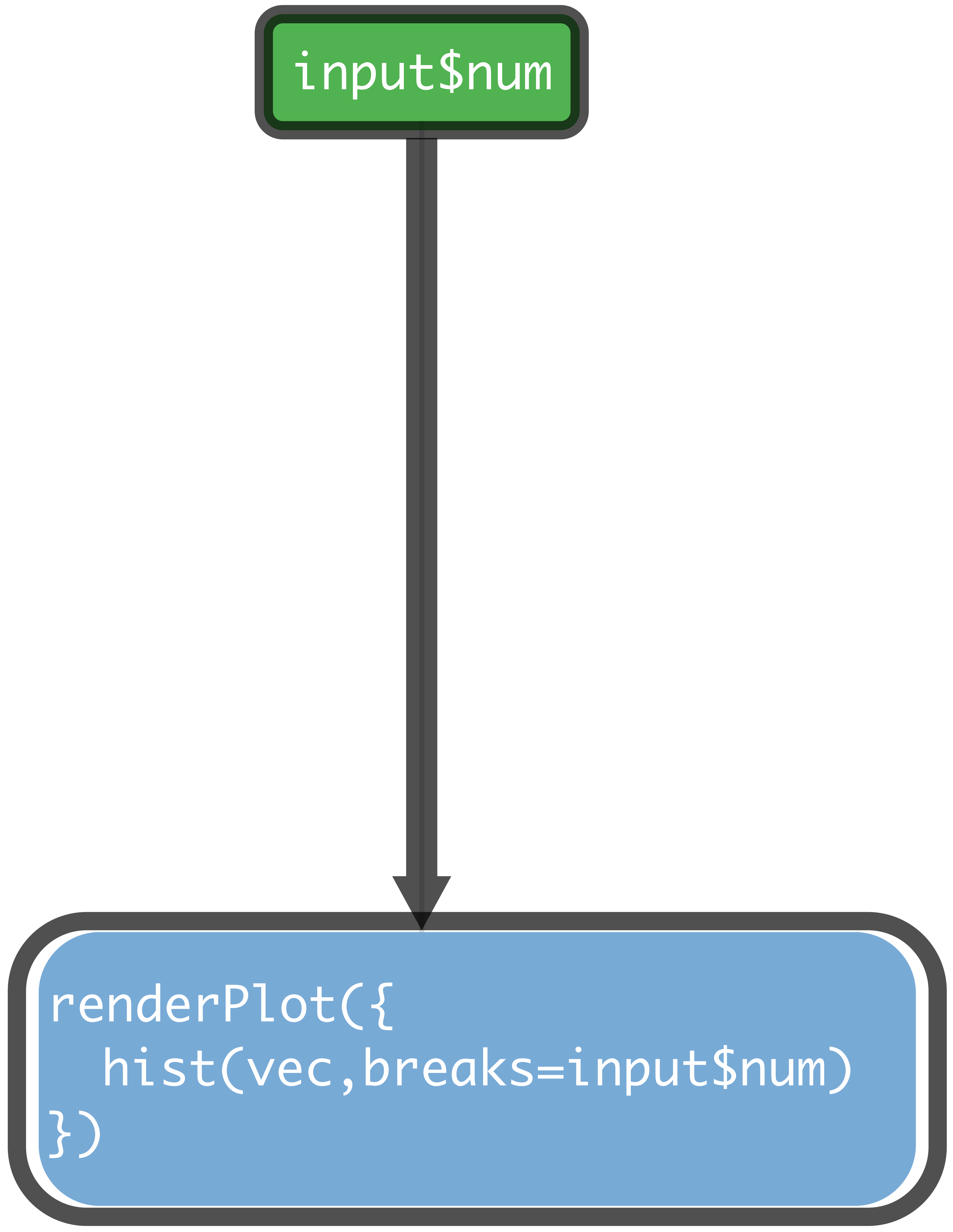
Use input values with input$

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

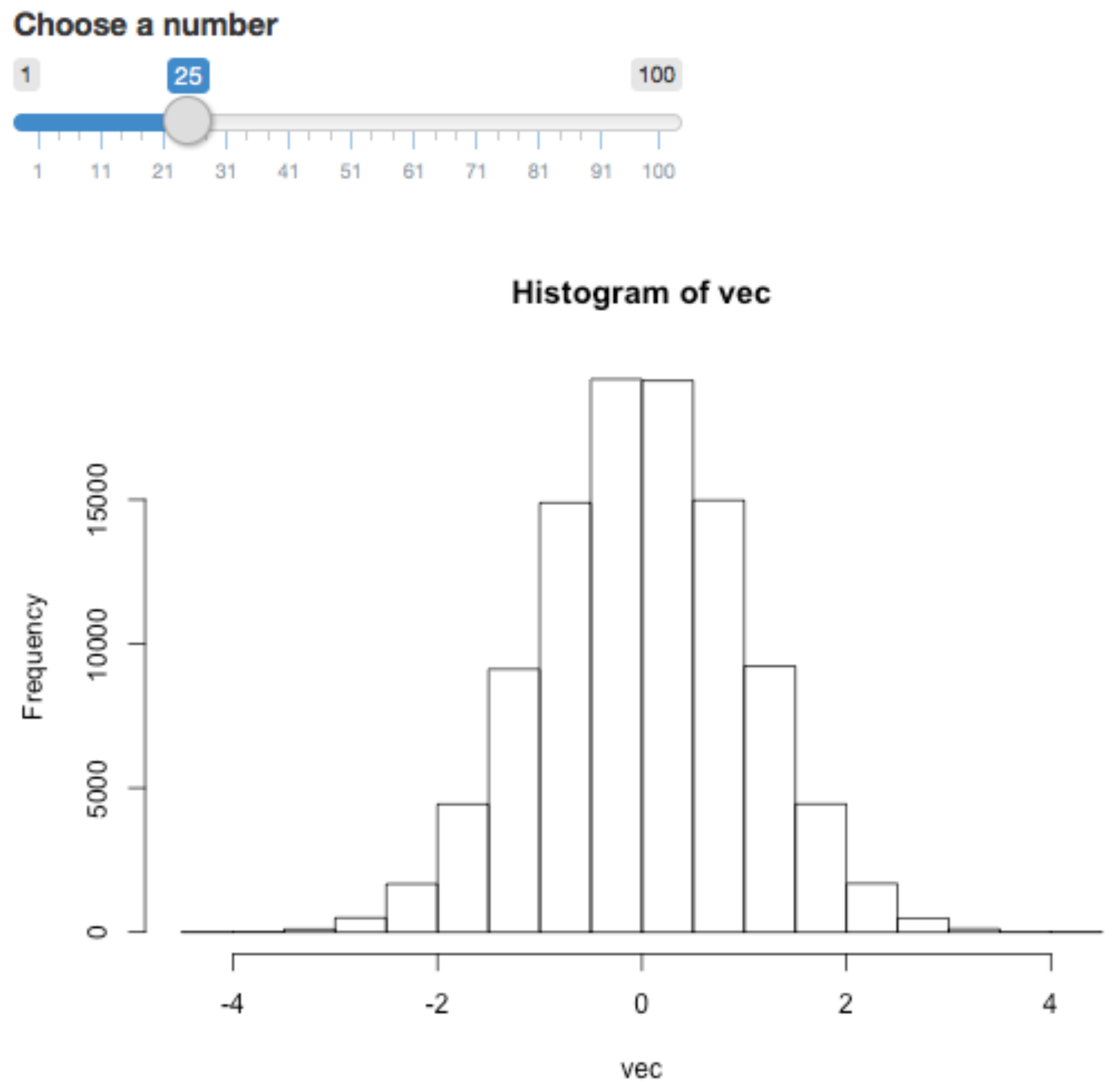
Reactivity 101

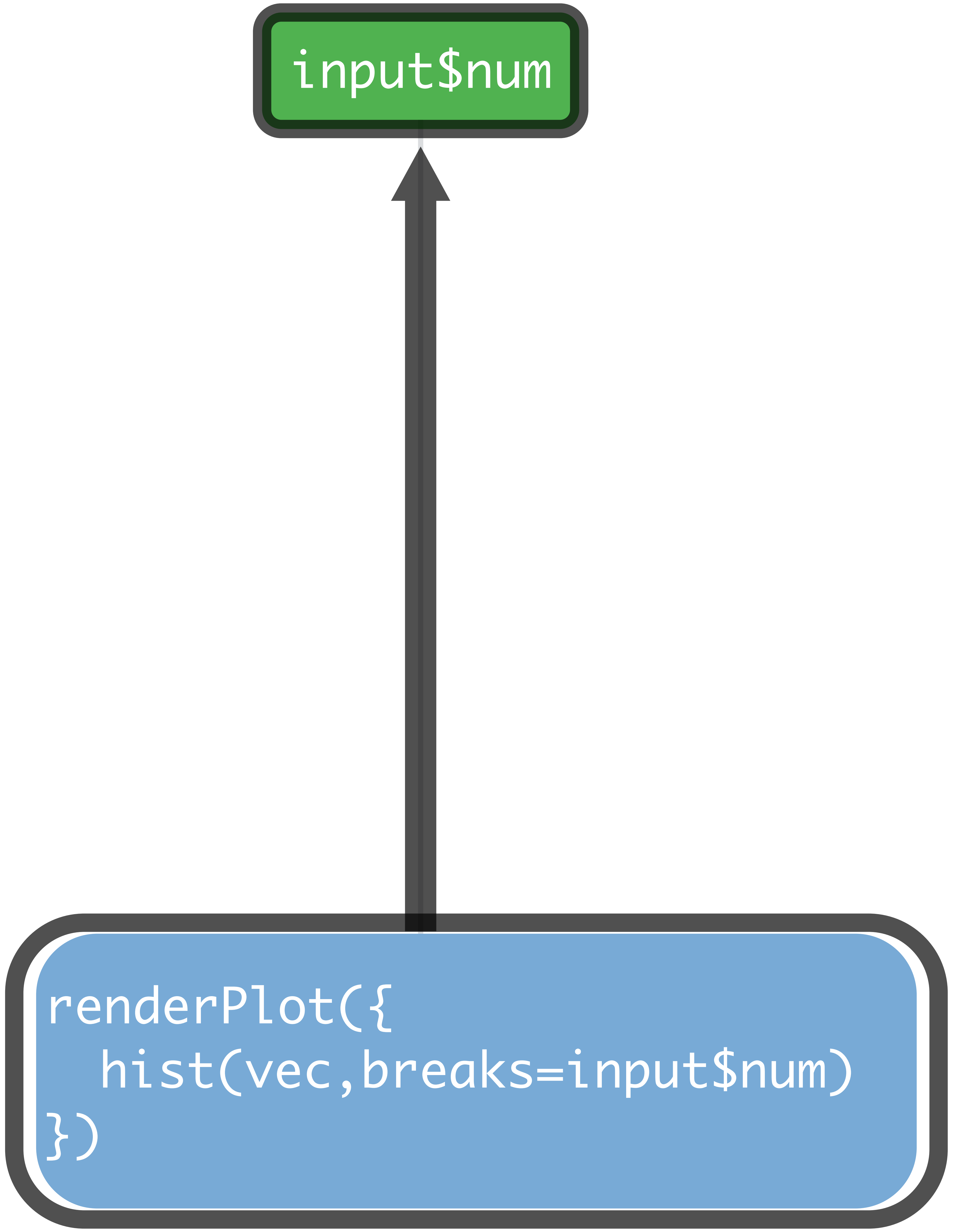
Reactivity automatically occurs whenever you use an input value to render an output object

```
function(input, output) {
  output$hist <- renderPlot({
    hist(vec, breaks = input$num)
  })
})
```

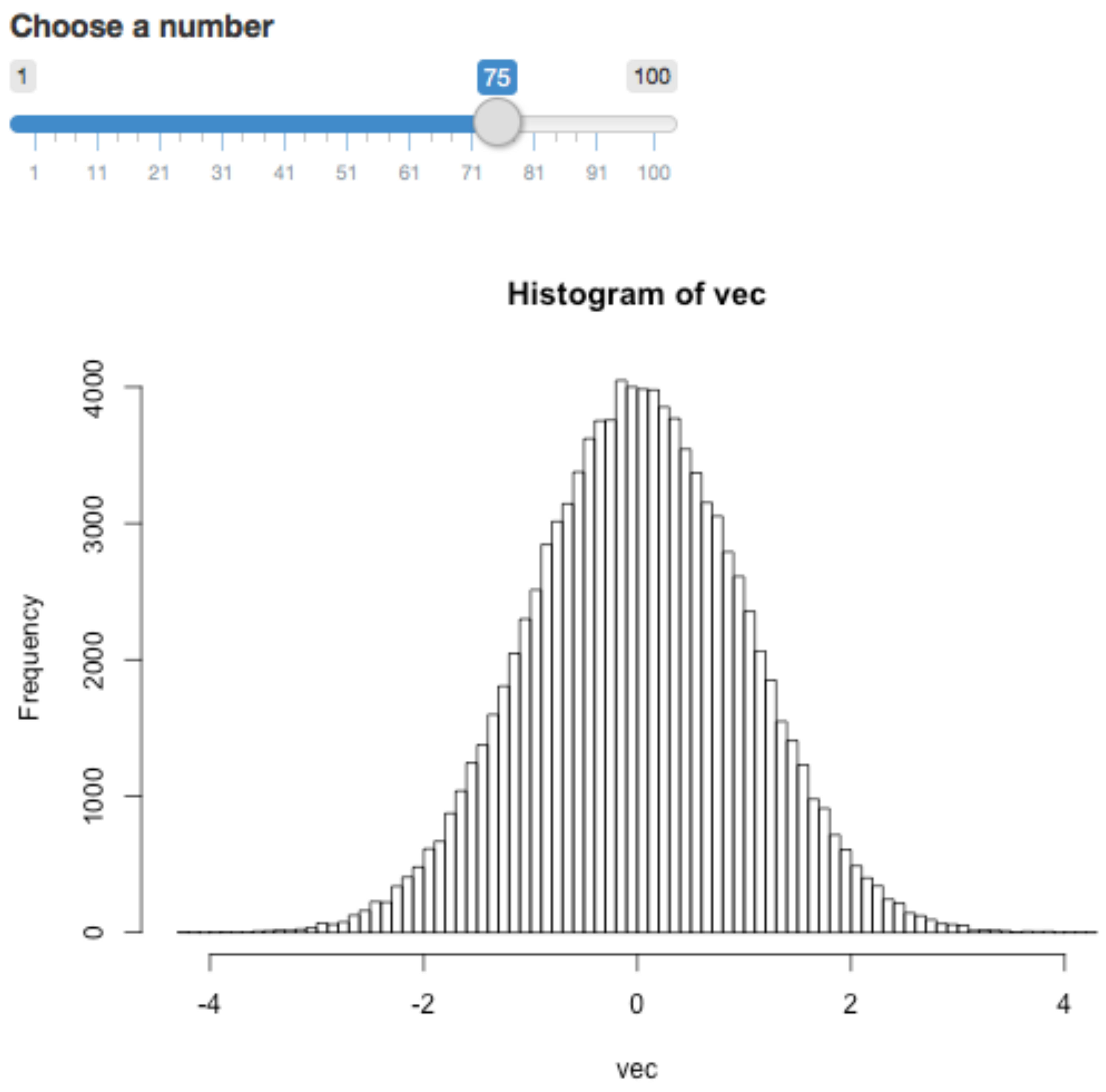


```
renderPlot({
  hist(vec,breaks=input$num)
})
```





```
renderPlot({
  hist(vec,breaks=input$num)
})
```



Recap: Server

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

`output$hist <-`

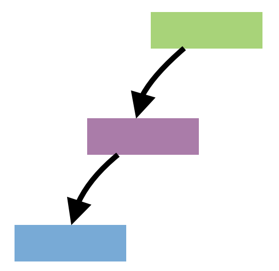
1. Save the output that you build to **output$**

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

2. Build the output with a **render*()** function

`input$num`

3. Access input values with **input$**



Create reactivity by using **Inputs** to build **rendered Outputs**

Practice

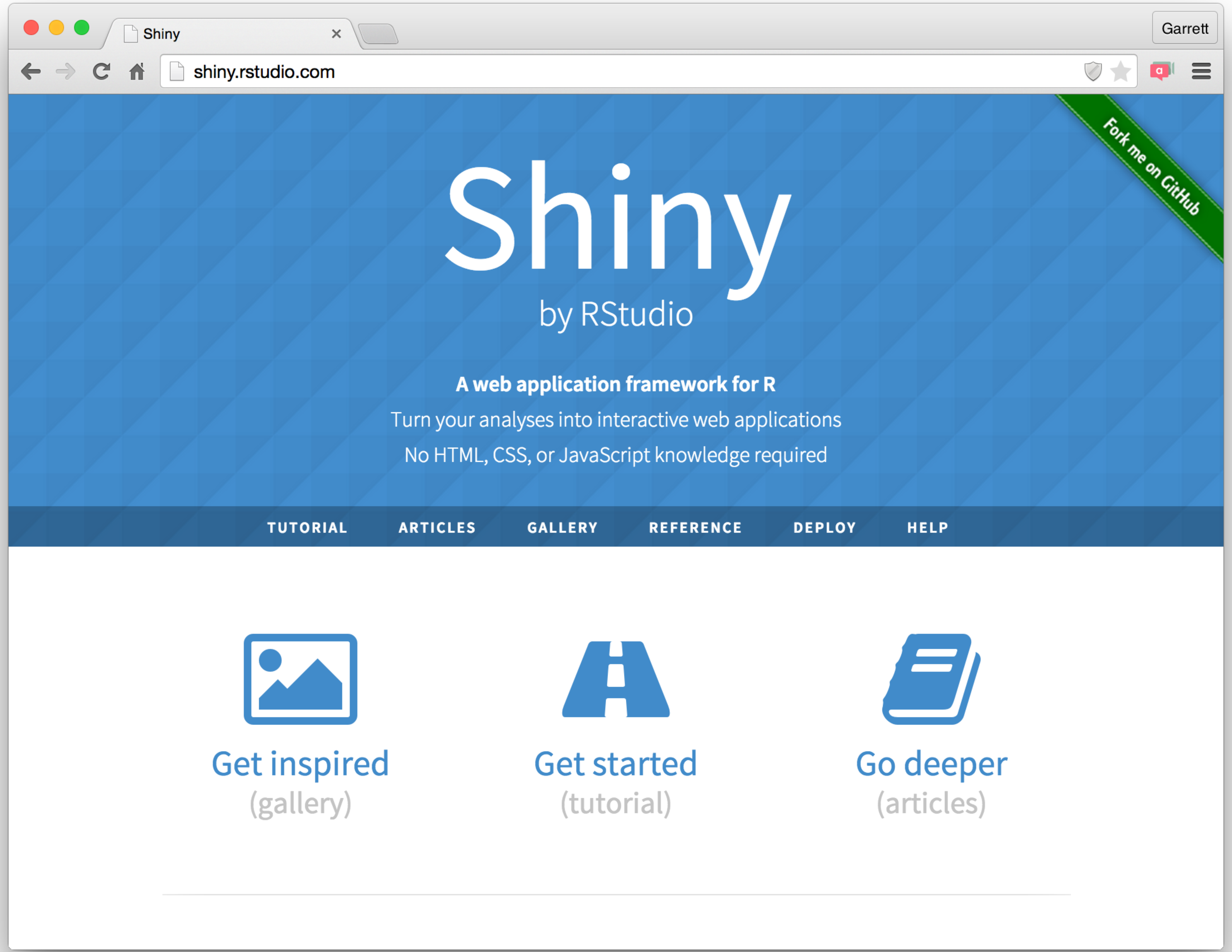
ZIP file containing R code and slides:

http://tinyurl.com/2016shiny

Learn more

The Shiny Development Center

shiny.rstudio.com



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)

Special thanks to Garrett Grolemund with RStudio for many of these slides