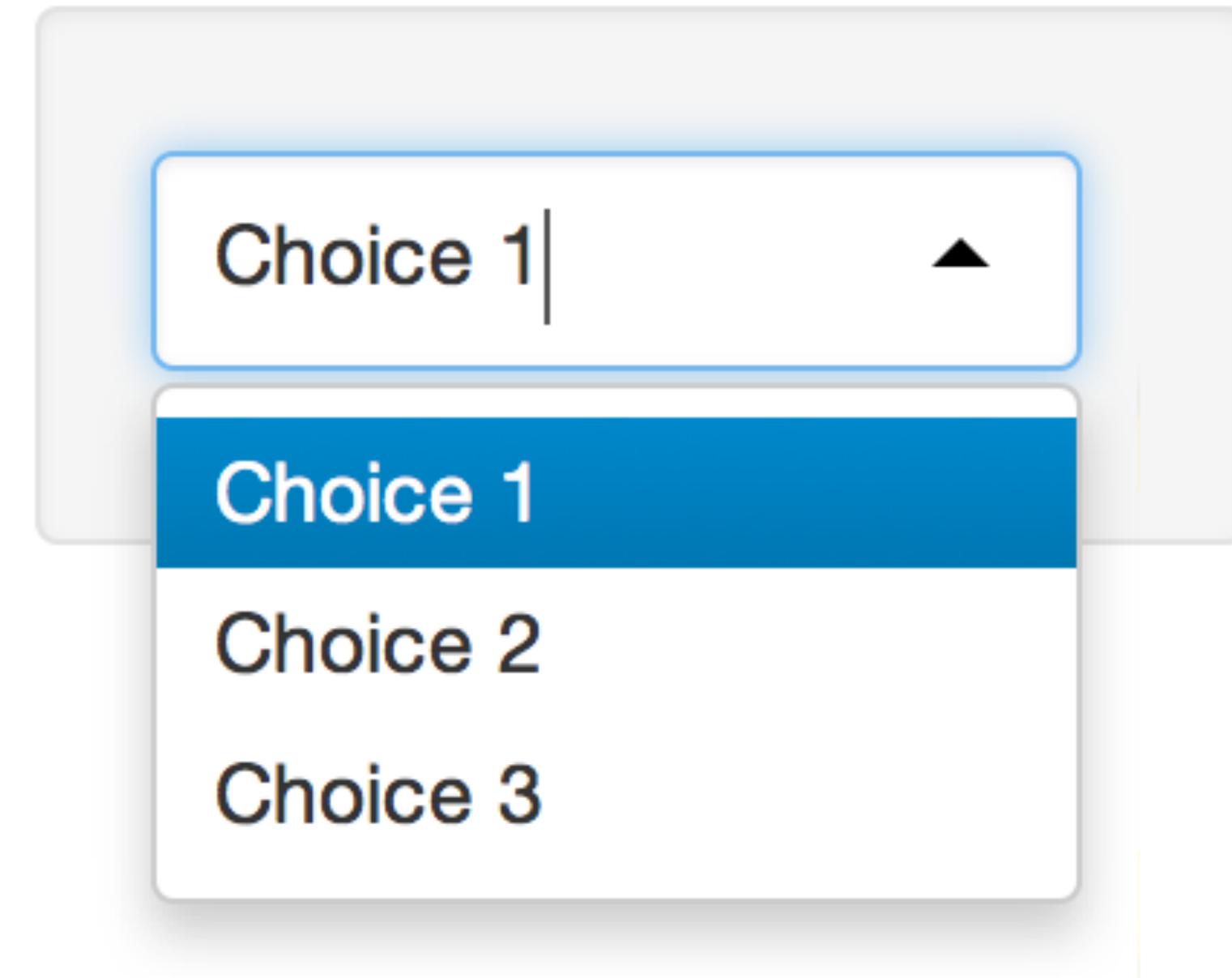


# Shiny Interactive Data Analysis

How to build a Shiny App



Dean Young ([youngde@reed.edu](mailto:youngde@reed.edu)) and Chester Ismay ([cismay@reed.edu](mailto: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(vector)
```

```
[1] 40.5
```

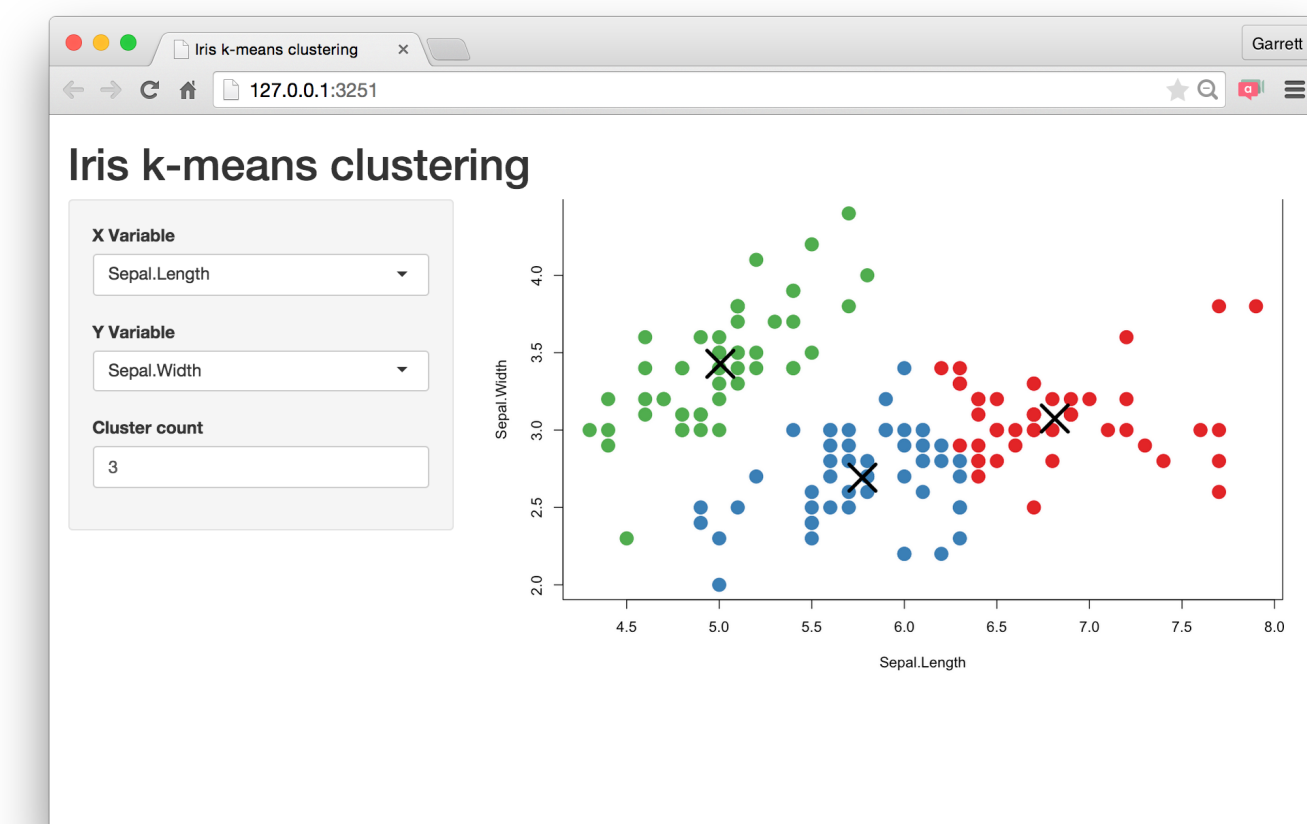
# Writing functions

```
# Function definition
cube.it <- function(x) {
  cube <- x * x * 2
  return(cube)
}
```

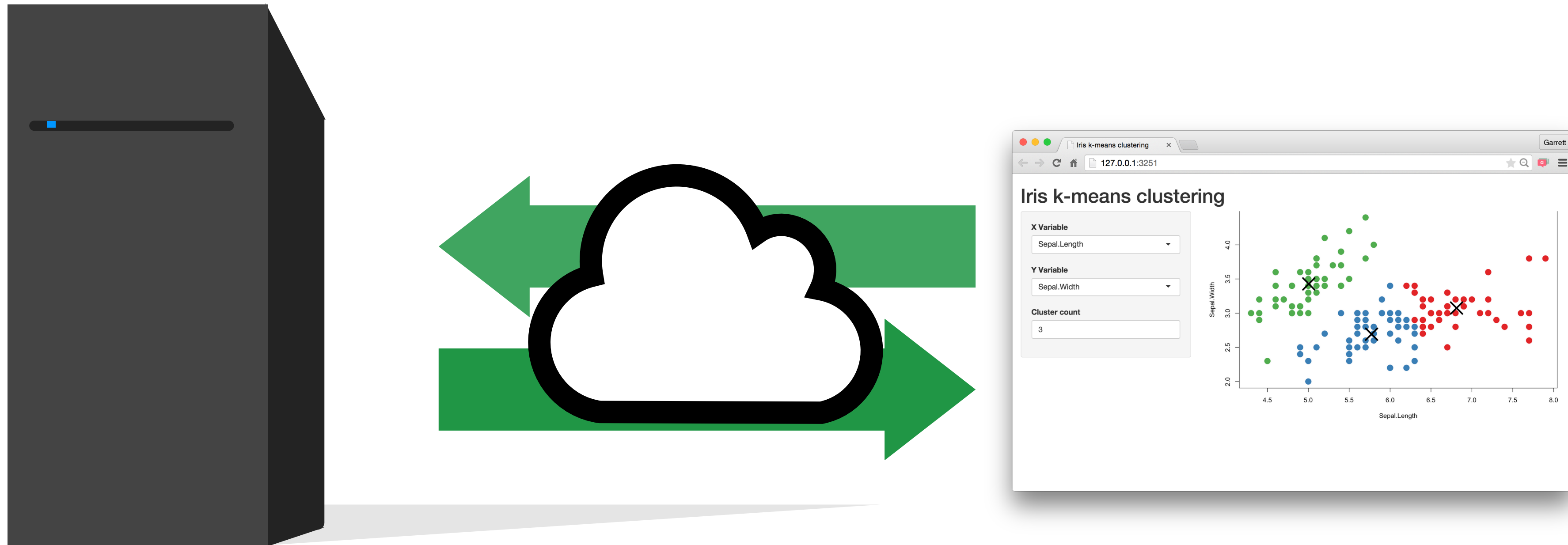
```
# Function call
cube(7)
[1] 343
```

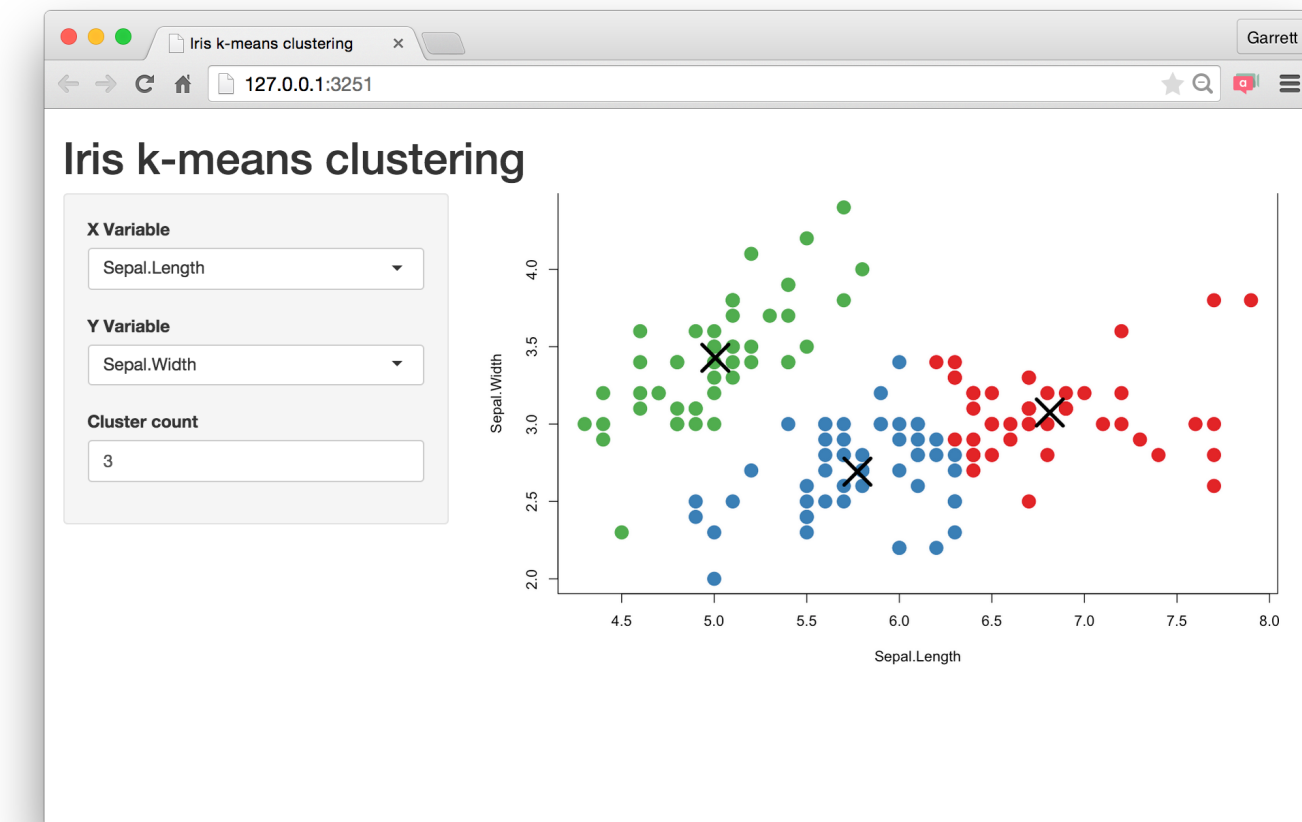
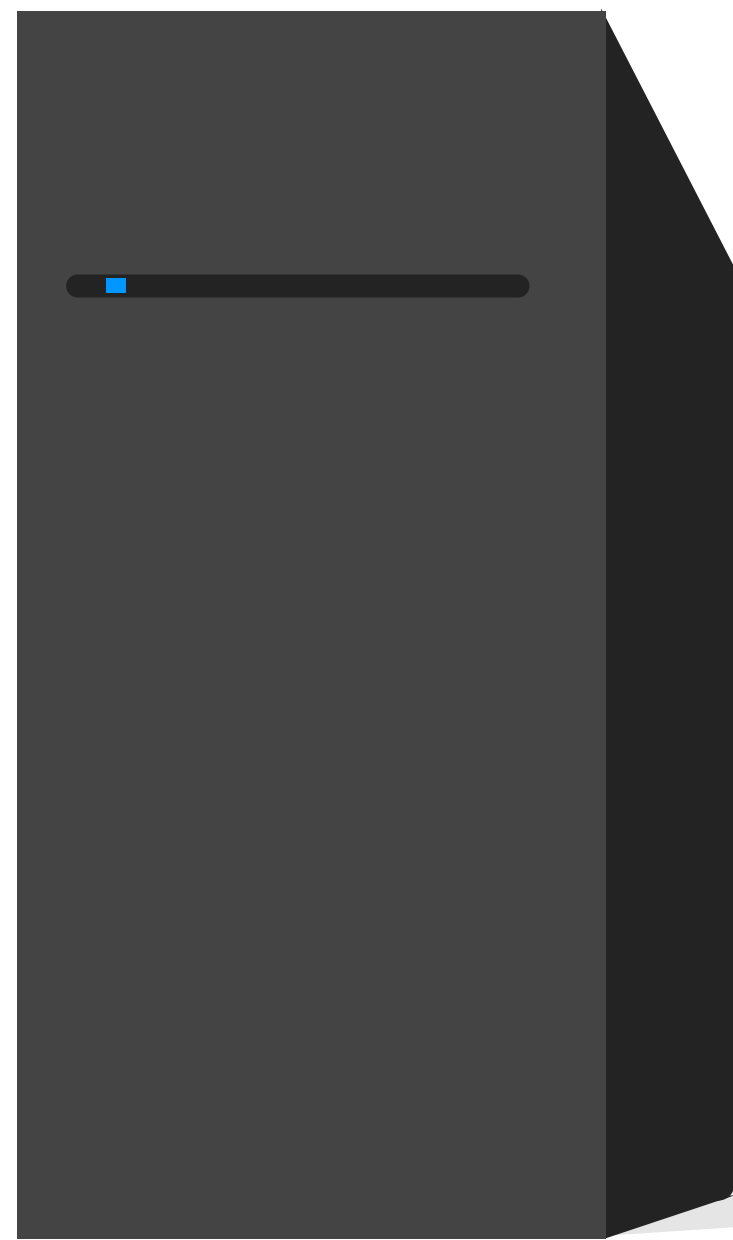
**Understand the  
architecture**

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)
```

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

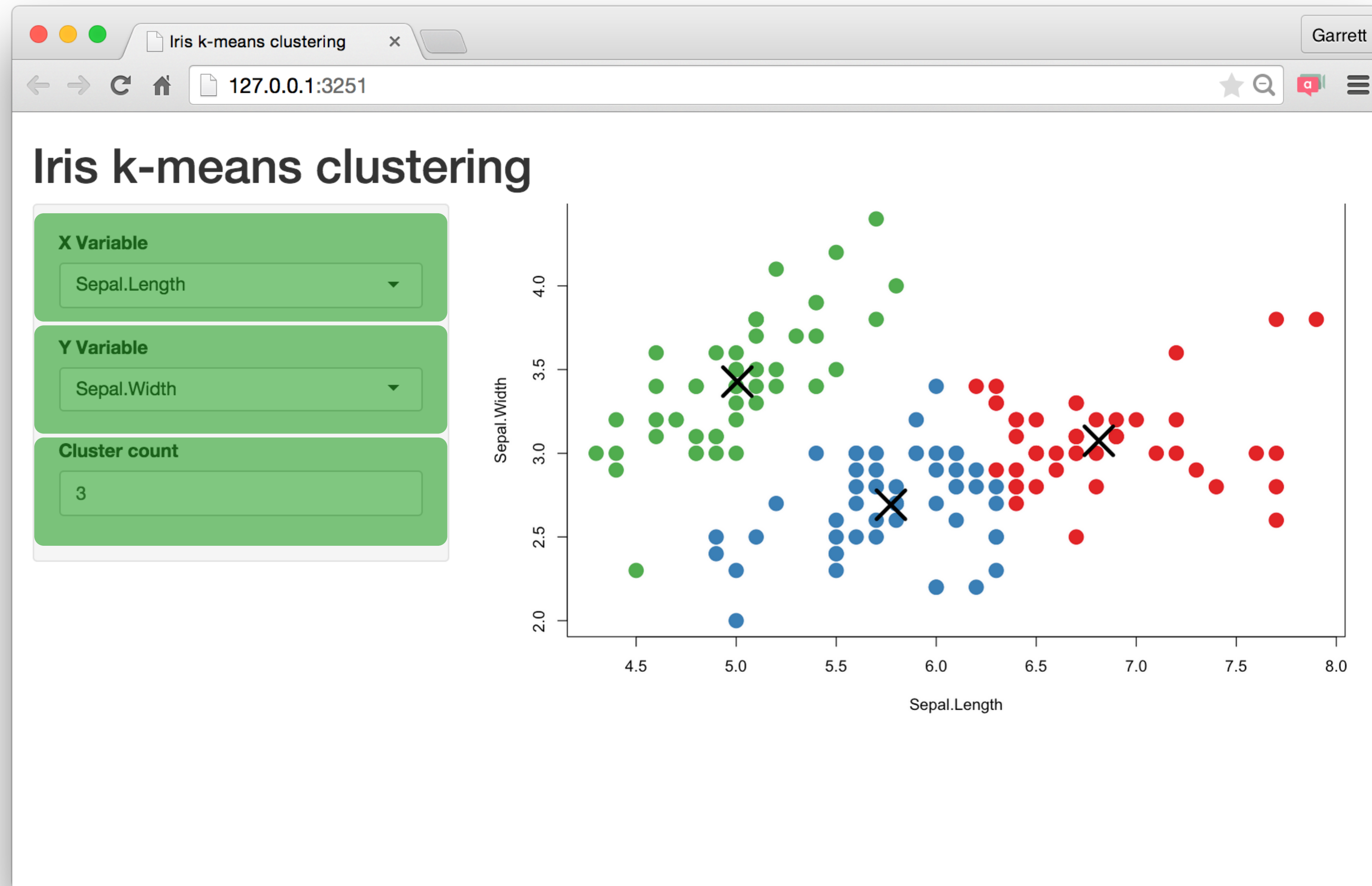
```
library(shiny)
ui <- fluidPage("Hello World")

server <- function(input, output) {}

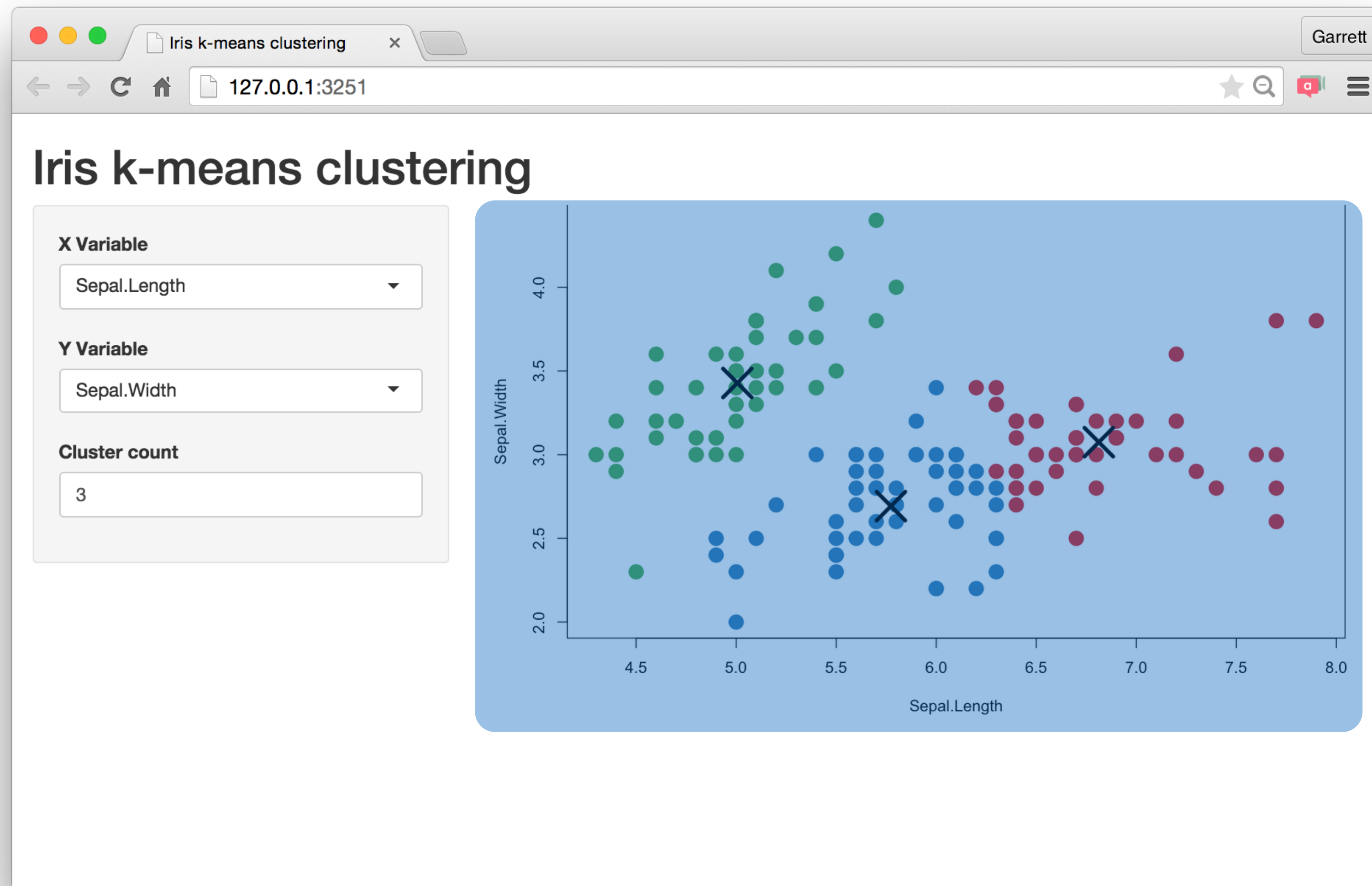
shinyApp(ui = ui, server = server)
```

# **Inputs and Outputs**

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



# 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.

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

)

server <- function(input, output) {}

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

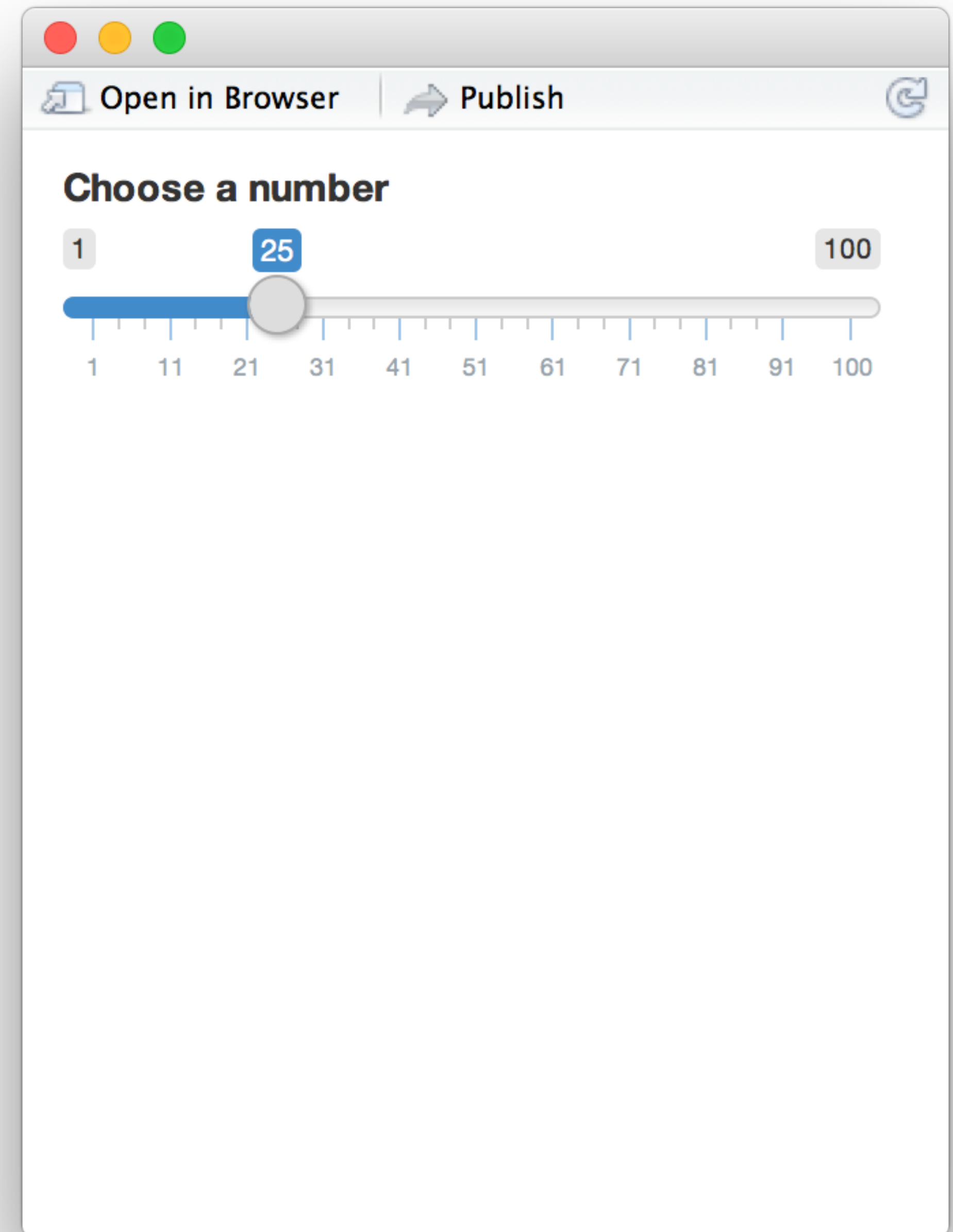


# Create an input with an input function.

```
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()`

## Single checkbox

☒ Choice A

`checkboxInput()`

## Checkbox group

☒ Choice 1

☐ Choice 2

☐ Choice 3

`checkboxGroupInput()`

## Date input

2014-01-01

`dateInput()`

## Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

## File input

Choose File No file chosen

`fileInput()`

## Numeric input

1

`numericInput()`

## Password Input

.....

`passwordInput()`

## Radio buttons

☒ Choice 1

☐ Choice 2

☐ Choice 3

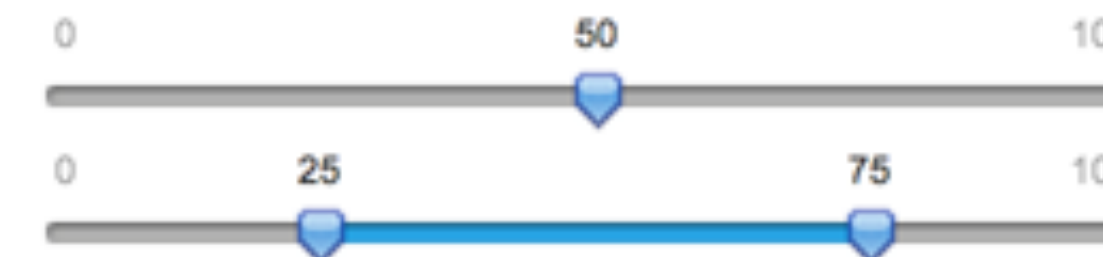
`radioButtons()`

## Select box

Choice 1

`selectInput()`

## Sliders



`sliderInput()`

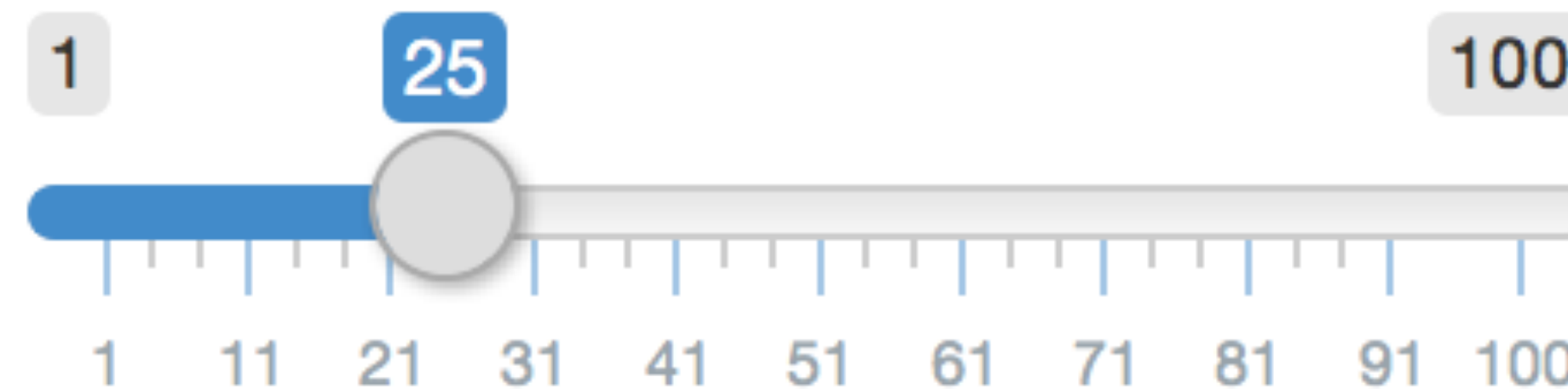
## Text input

Enter text...

`textInput()`

# Syntax

**Choose a number**



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

input name  
(for internal use)

Notice:  
Id not ID

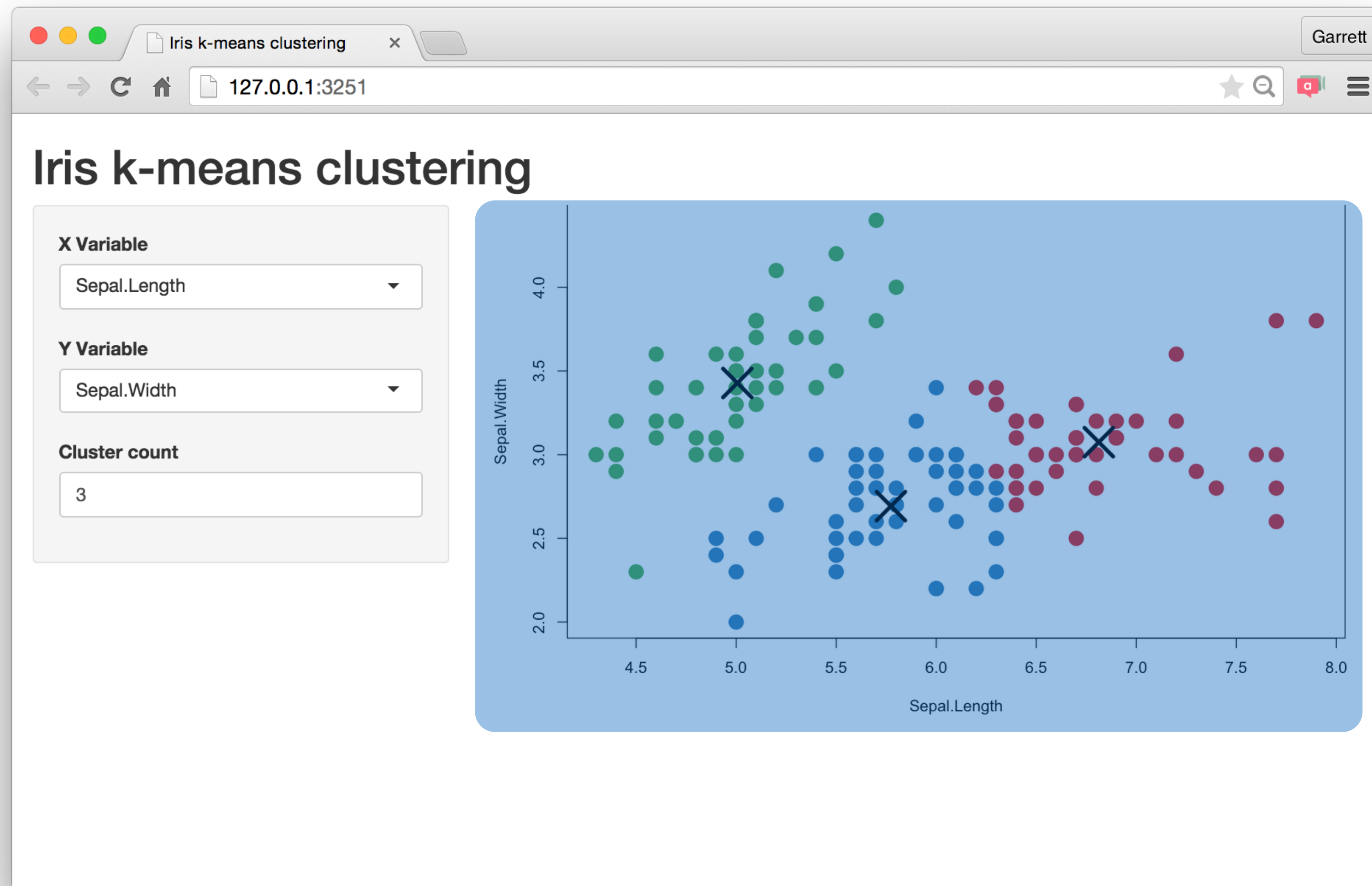
label to  
display

input specific  
arguments

?sliderInput

# Outputs

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



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



```
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)
```

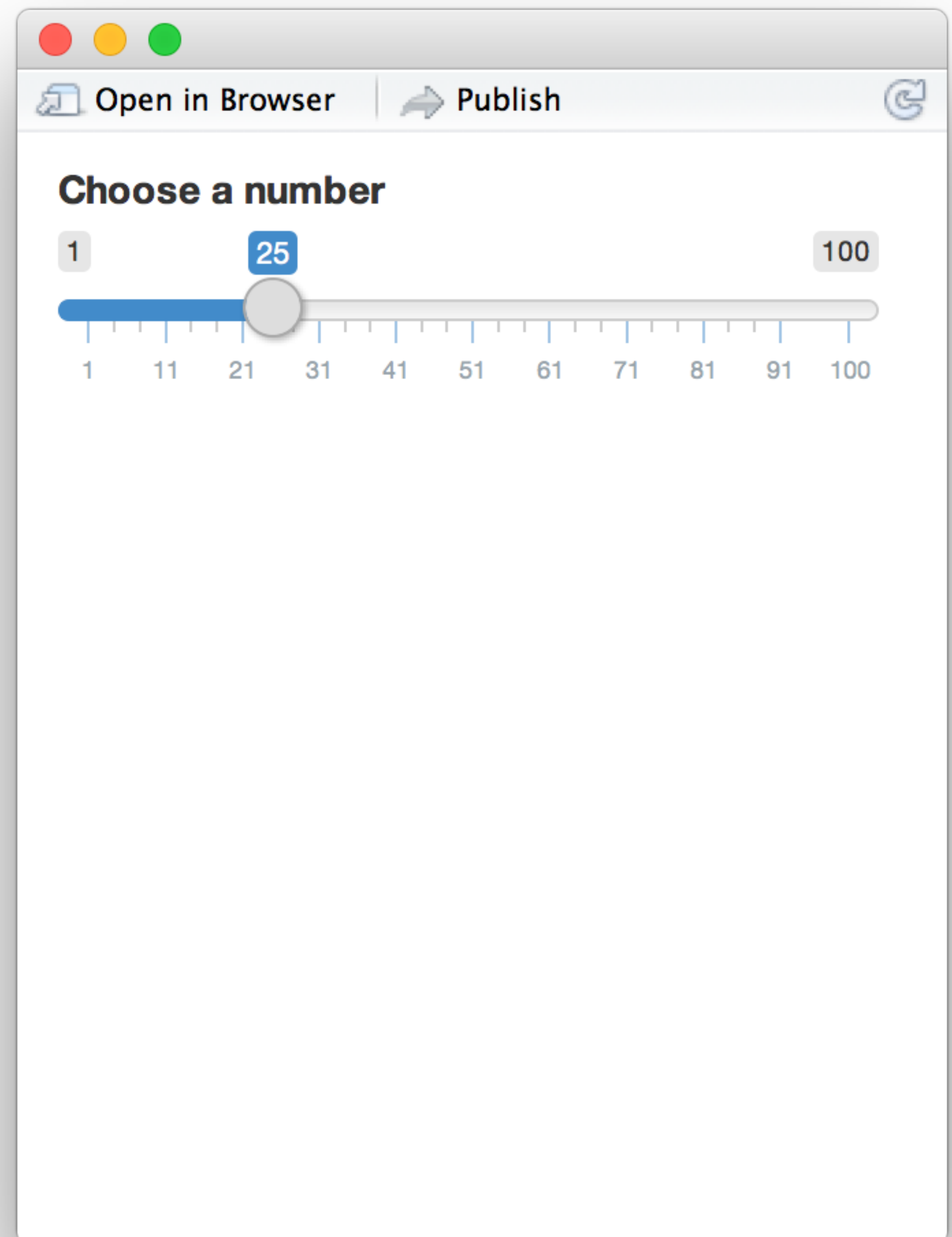
Comma between  
arguments

```
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)
```

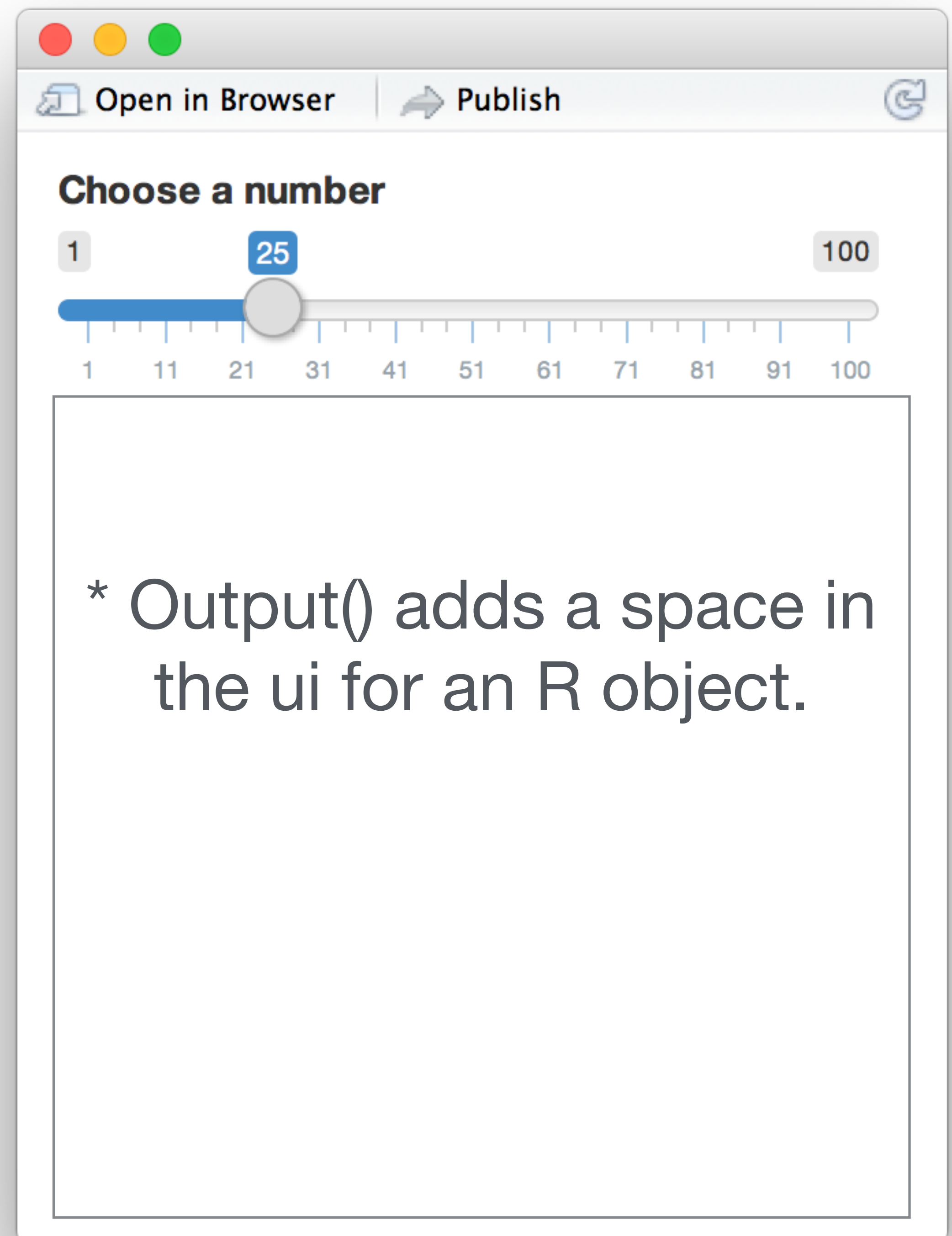


```
library(shiny)

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)
```

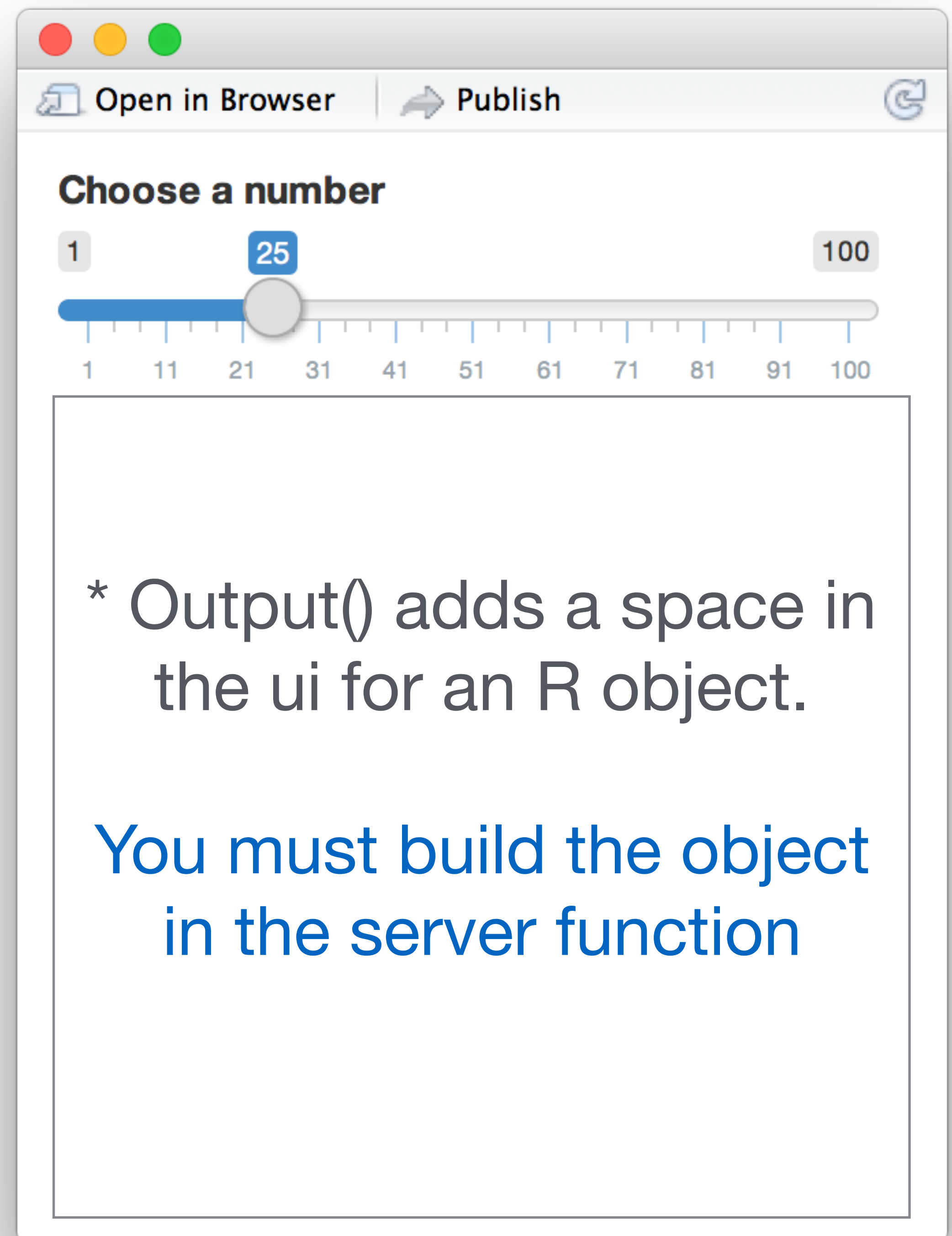


```
library(shiny)

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

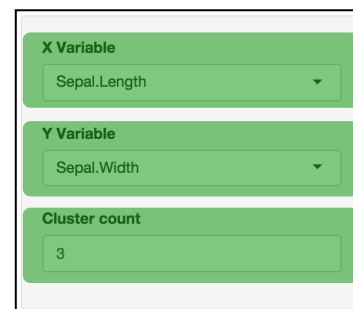
```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```



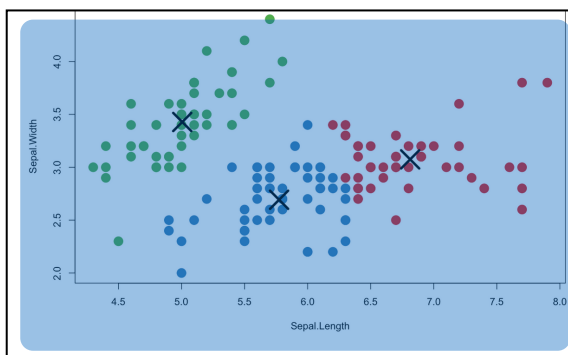
Hello World

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

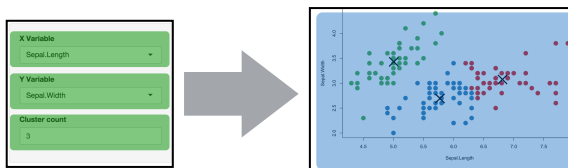
Create reactive inputs with an **\*Input()** function



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**



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

## 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\$

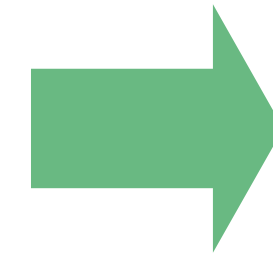
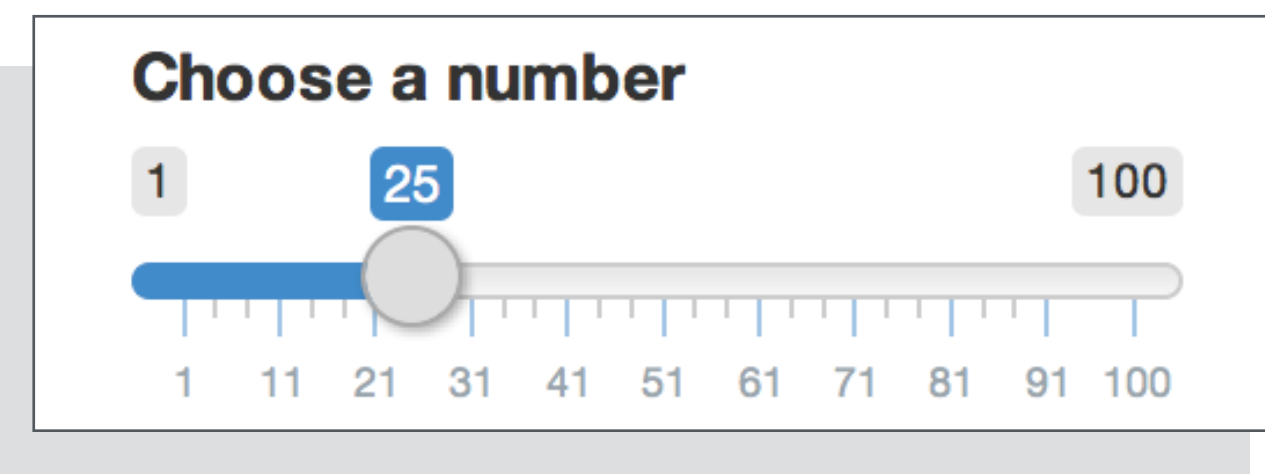
```
sliderInput(inputId = "num", ...)
```



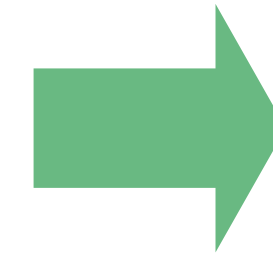
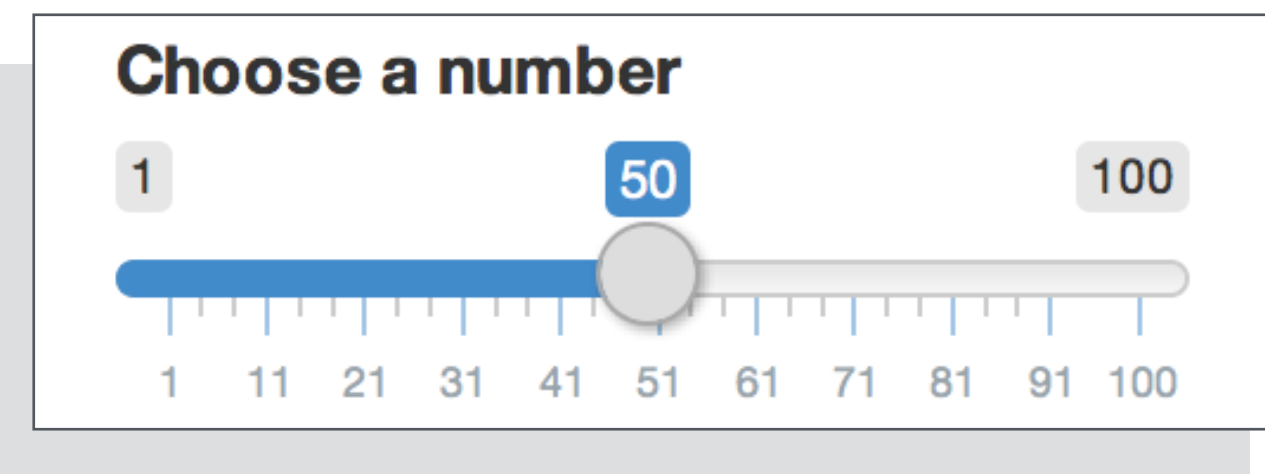
```
input$num
```

# Input values

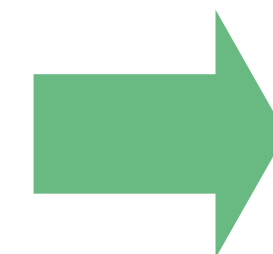
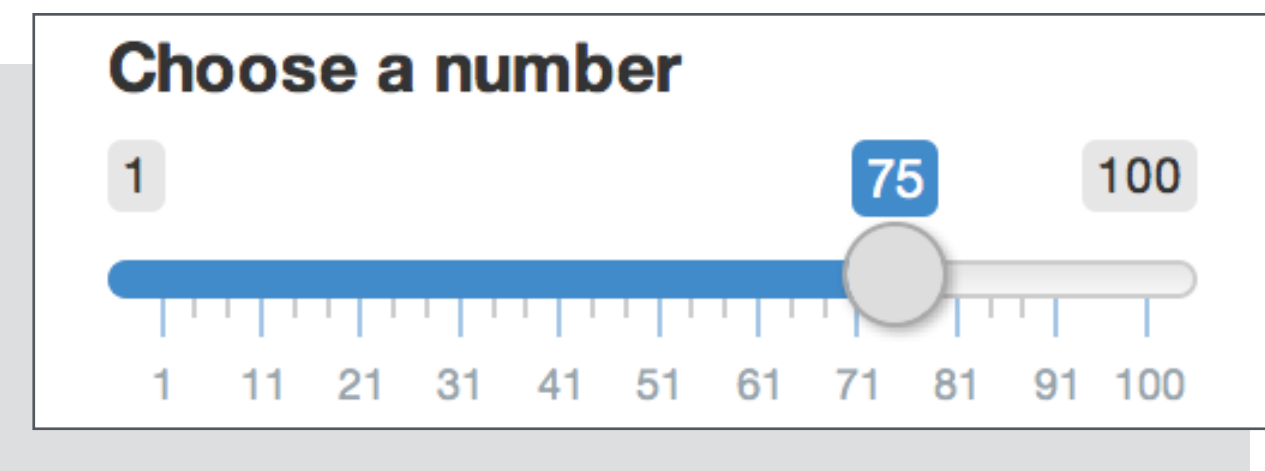
The input value changes whenever a user changes the input.



```
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

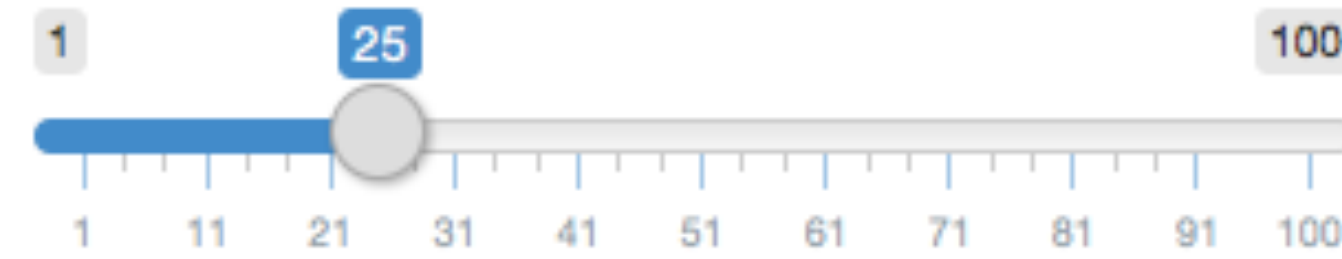
Reactivity automatically occurs whenever you use an input value to render an output object

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

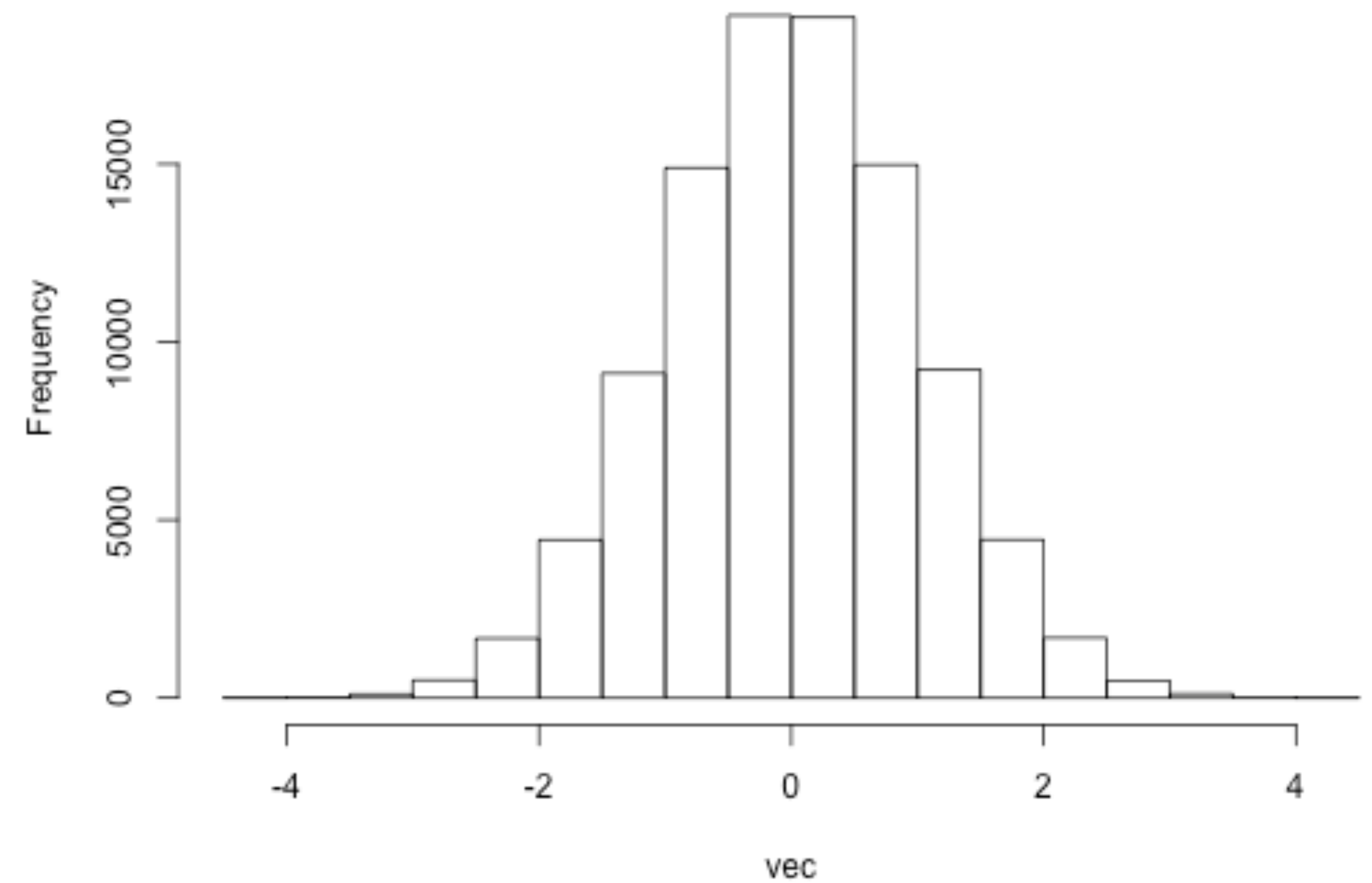
input\$num

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

Choose a number



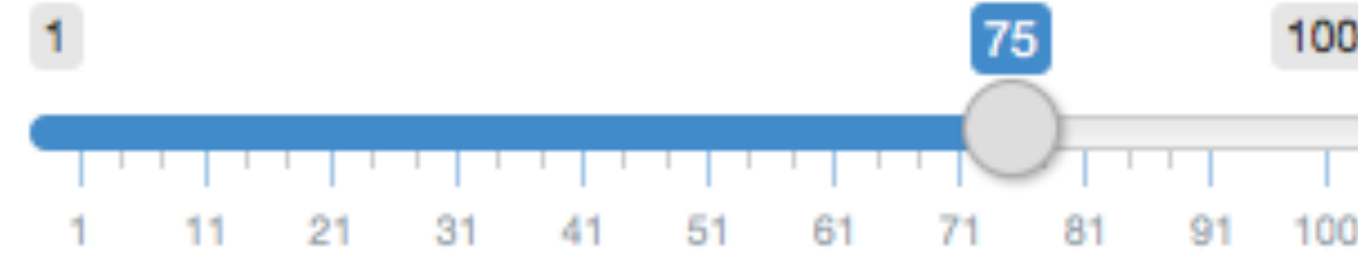
Histogram of vec



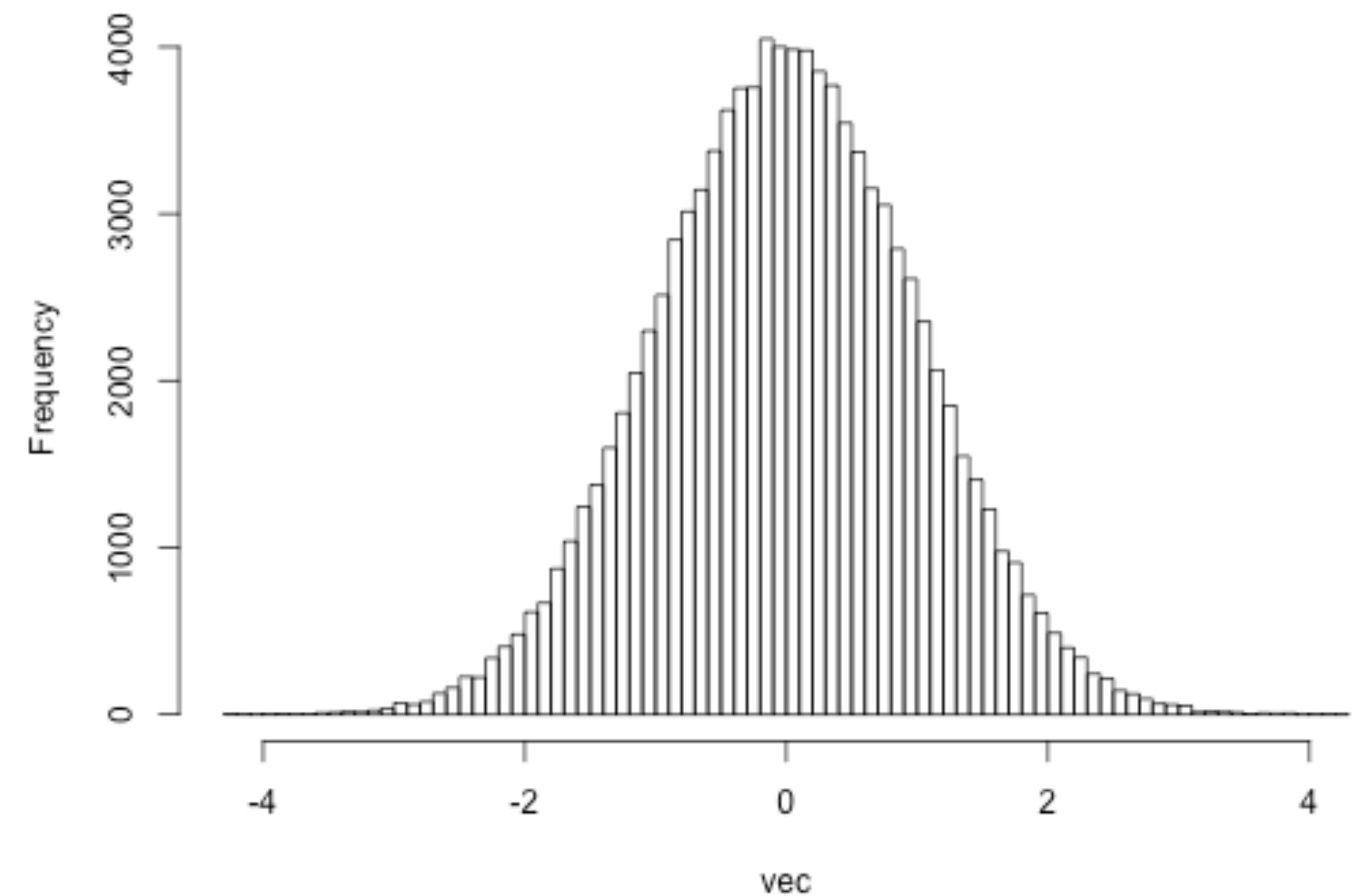
input\$num

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

Choose a number



Histogram of vec



# Recap: Server



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

**output\$hist** ←

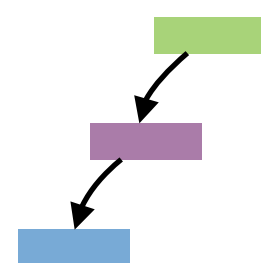
1. Save the output that you build to **output\$**

```
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:

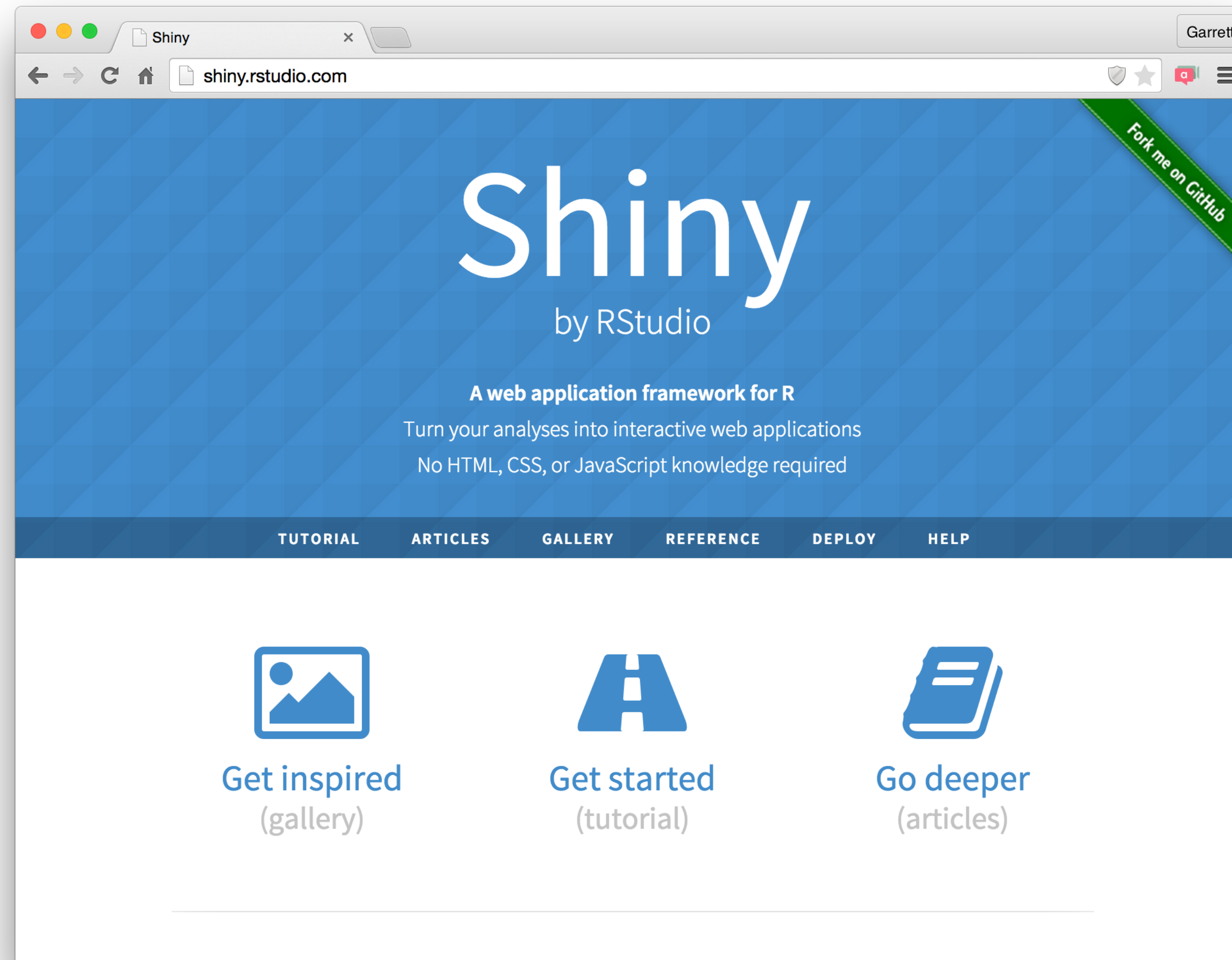
<http://tinyurl.com/2016shiny>

**Learn**

**more**

# The Shiny Development Center

[shiny.rstudio.com](https://shiny.rstudio.com)





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