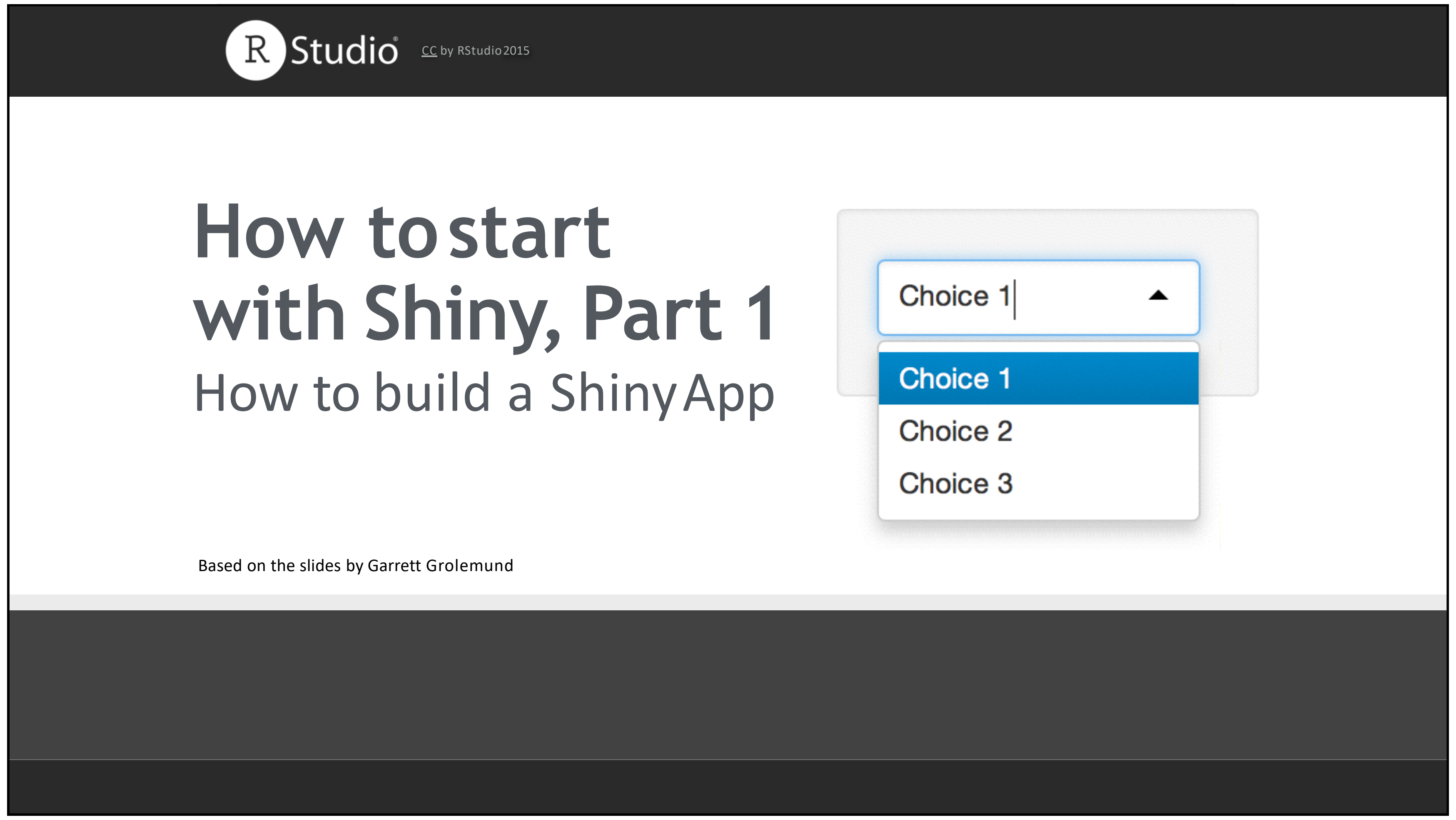
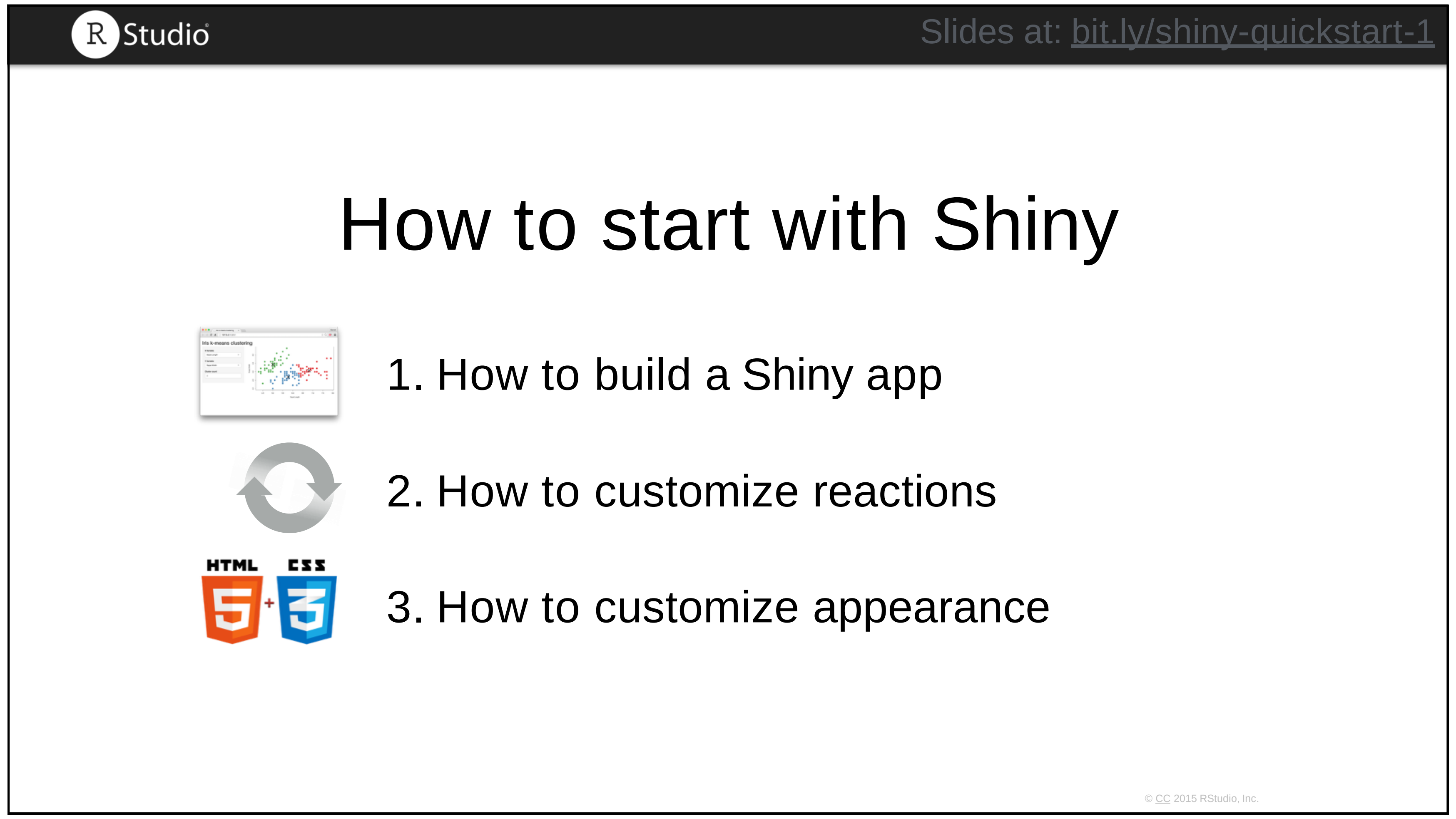


How to start with Shiny, Part 1

How to build a Shiny App

Based on the slides by Garrett Grolemund

1



RStudio Slides at: bit.ly/shiny-quickstart-1

How to start with Shiny

1. How to build a Shiny app
2. How to customize reactions
3. How to customize appearance

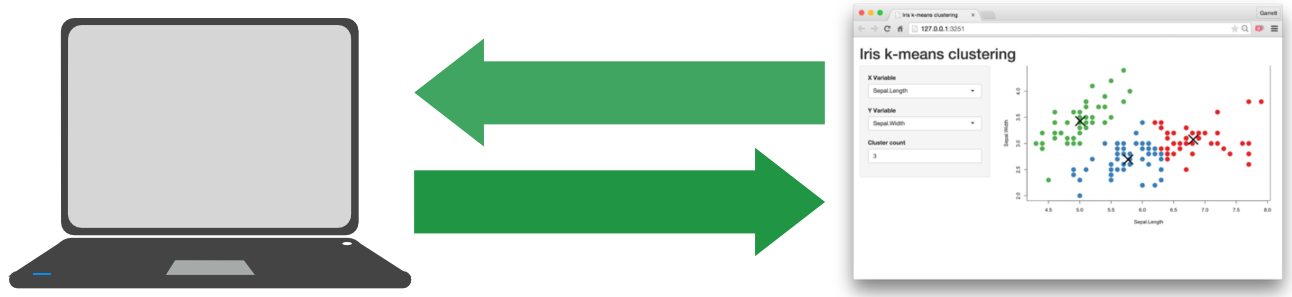
3

Understand the architecture

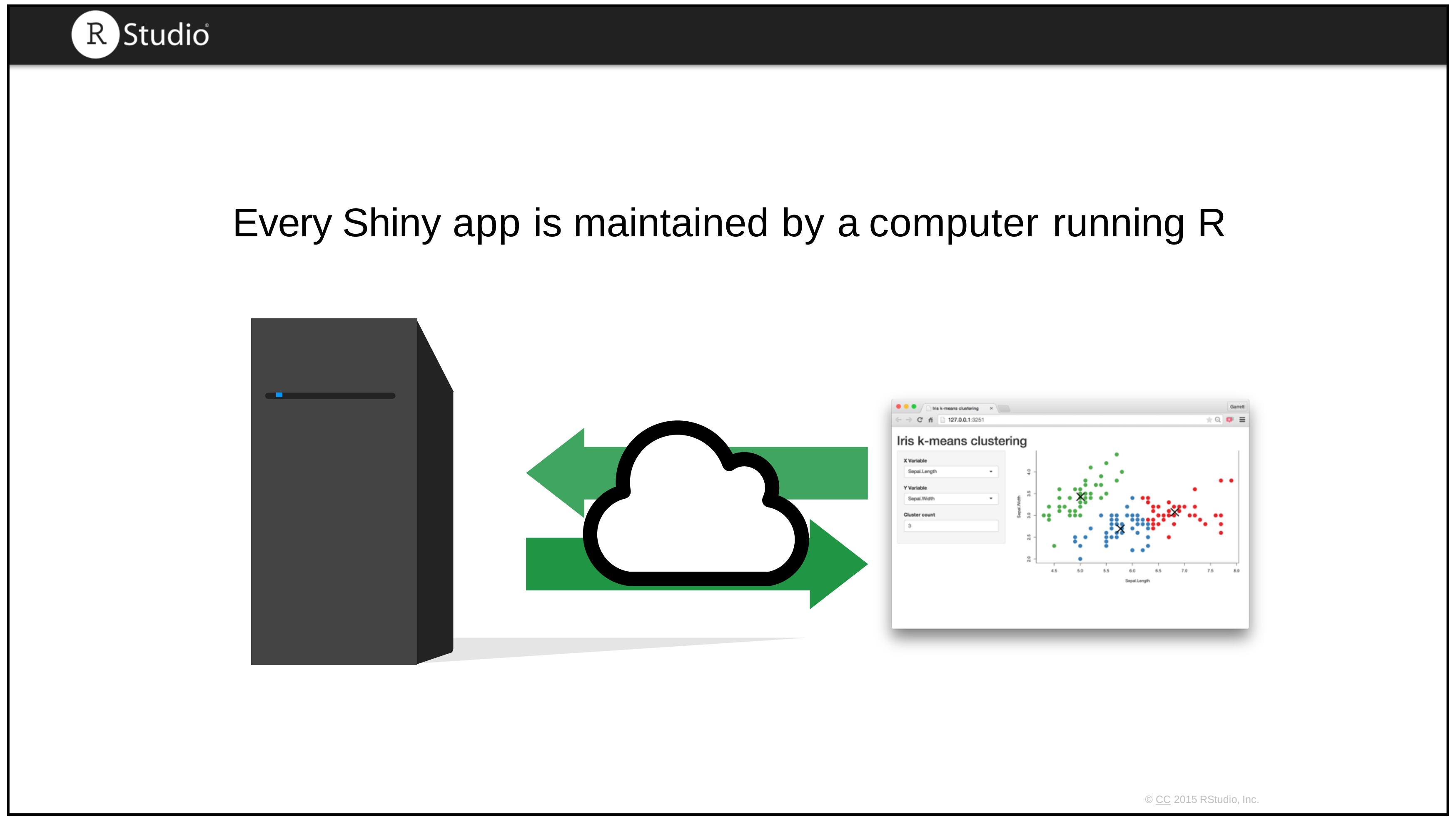
4

RStudio

Every Shiny app is maintained by a computer running R



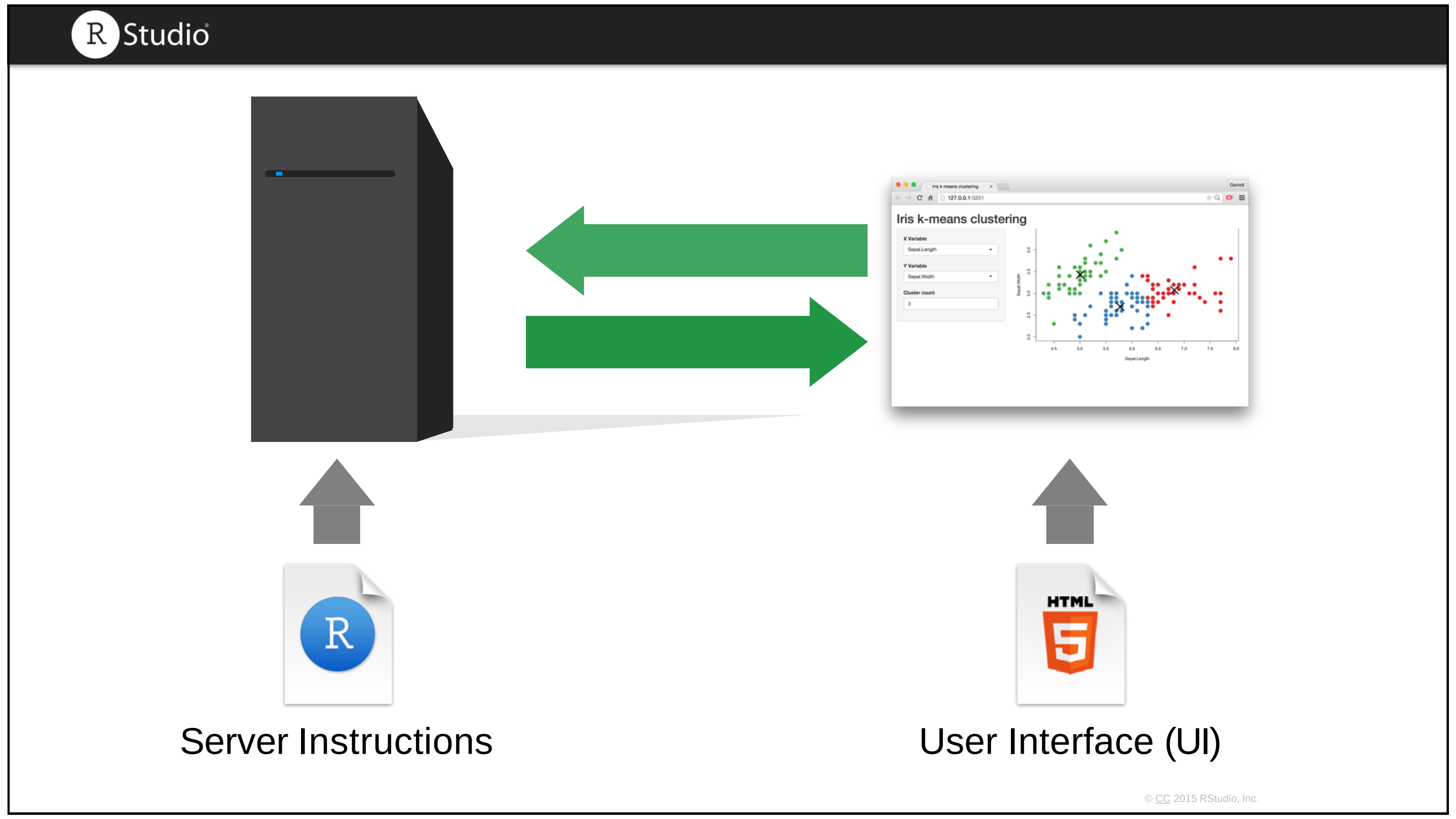
5



R Studio

Every Shiny app is maintained by a computer running R

6

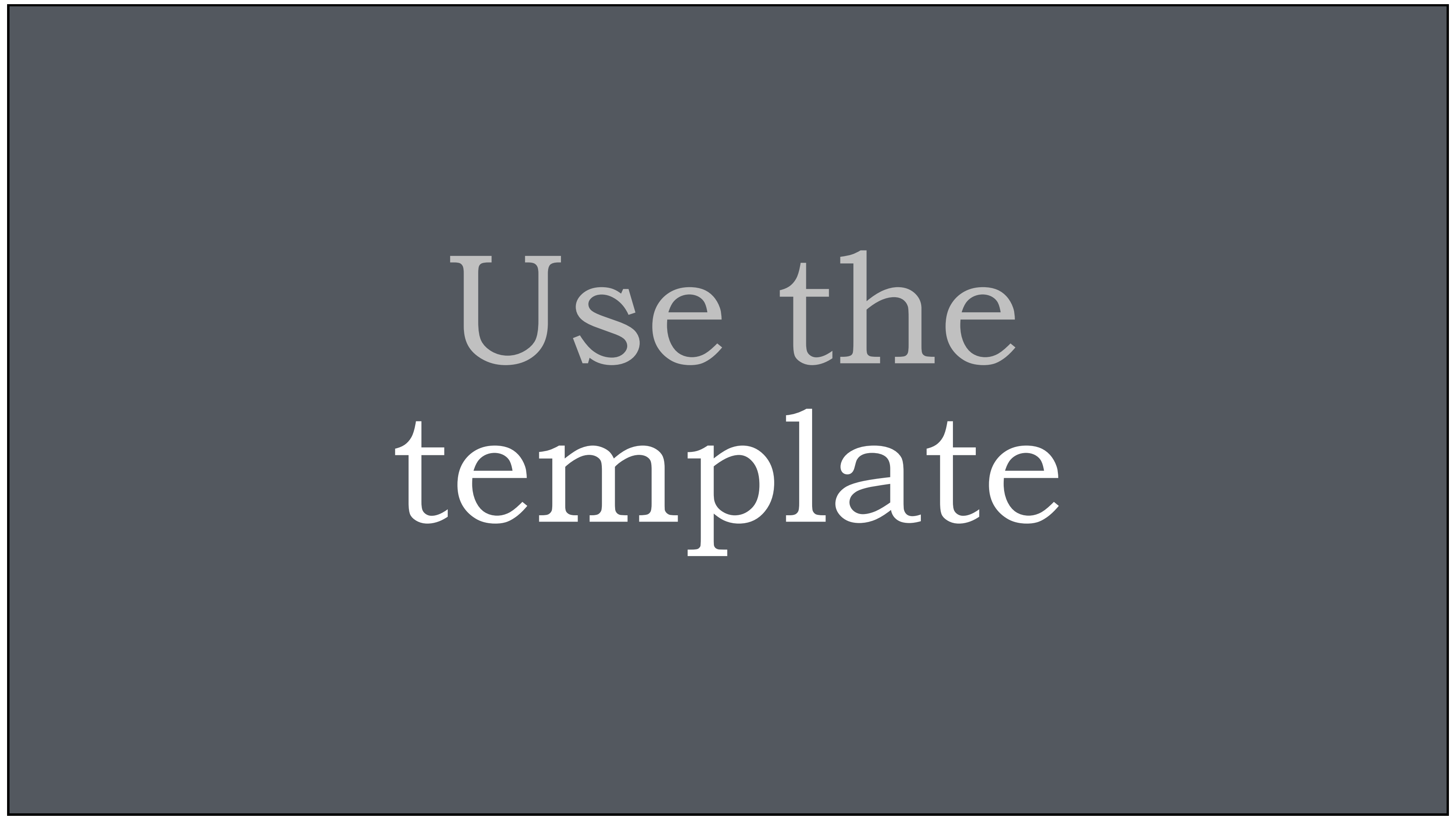


R Studio

Server Instructions

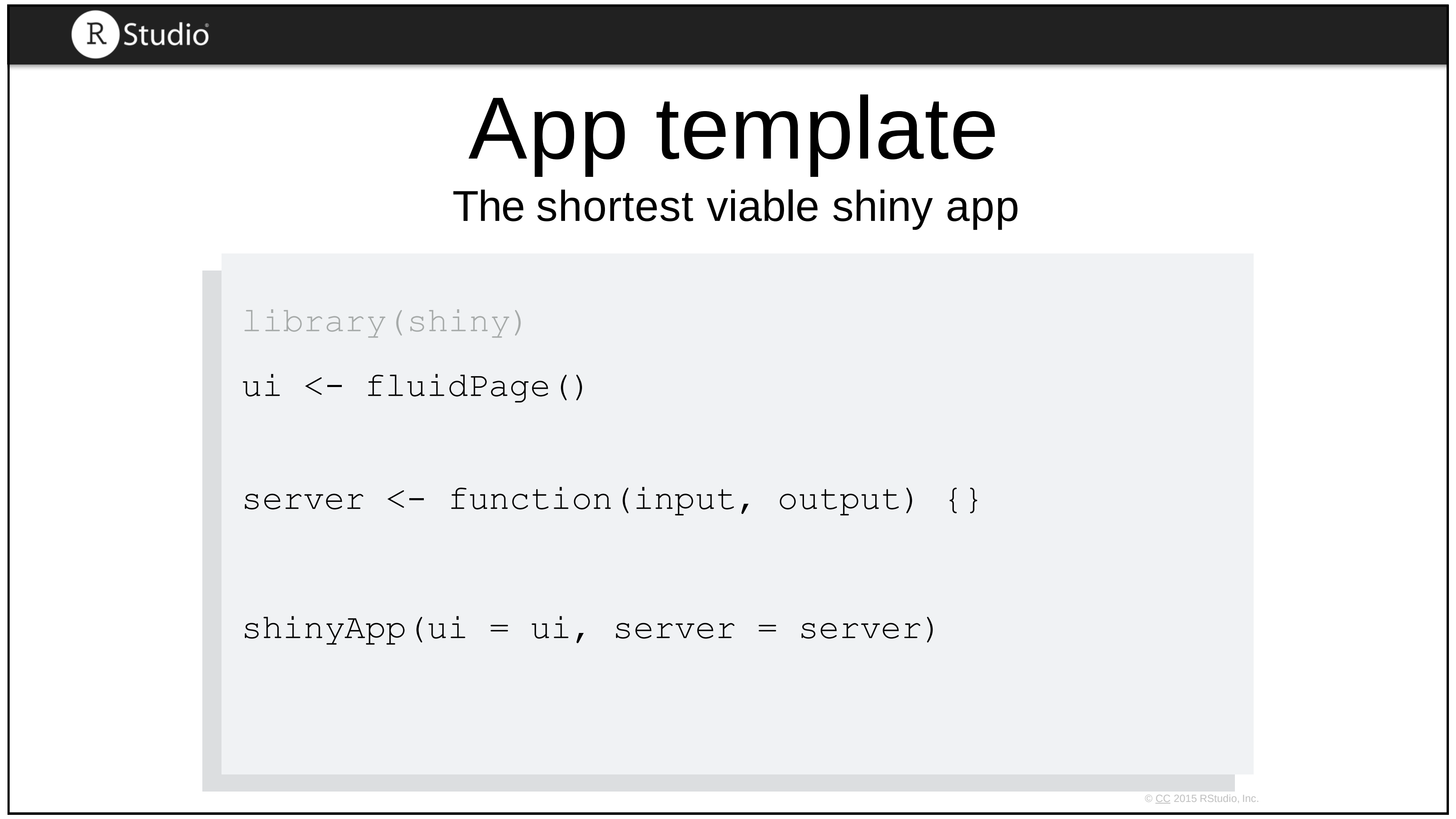
User Interface (UI)

7



Use the template

8



App template

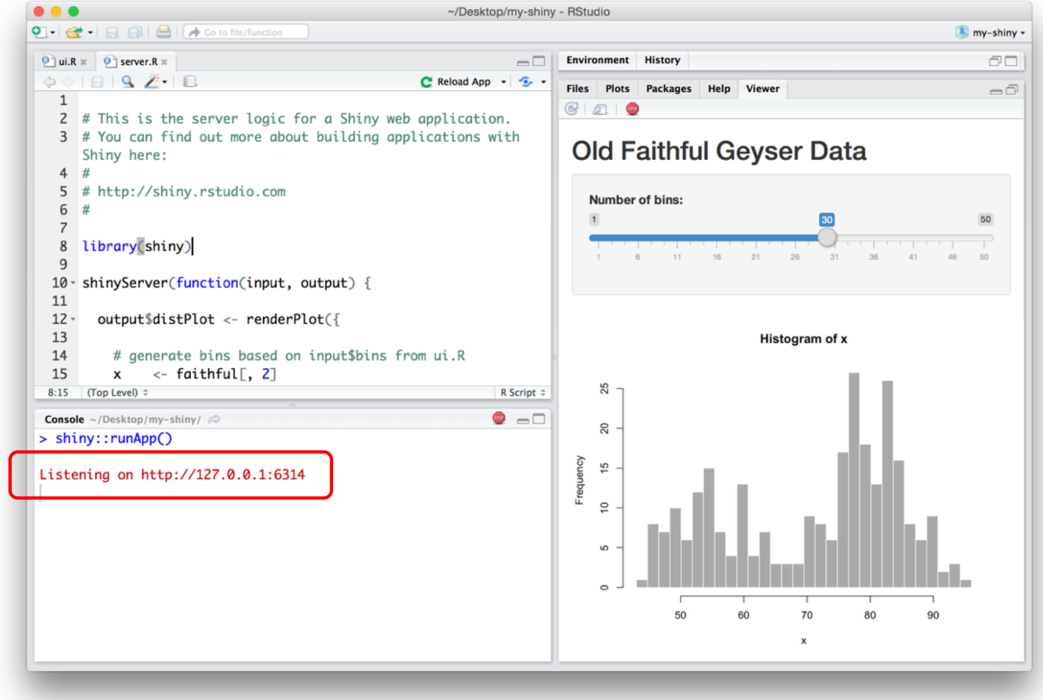
The shortest viable shiny app

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

server <- function(input, output) {}


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

9

R Studio

An app running

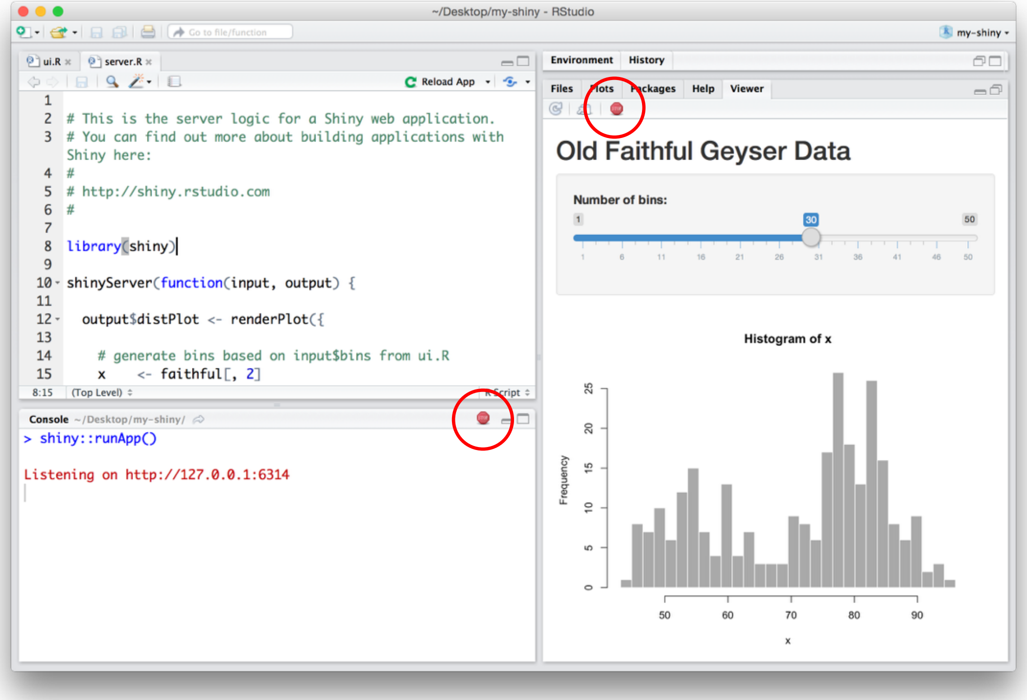


```
# This is the server logic for a Shiny web application.
# You can find out more about building applications with
Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
```

```
> shiny::runApp()

Listening on http://127.0.0.1:6314
```

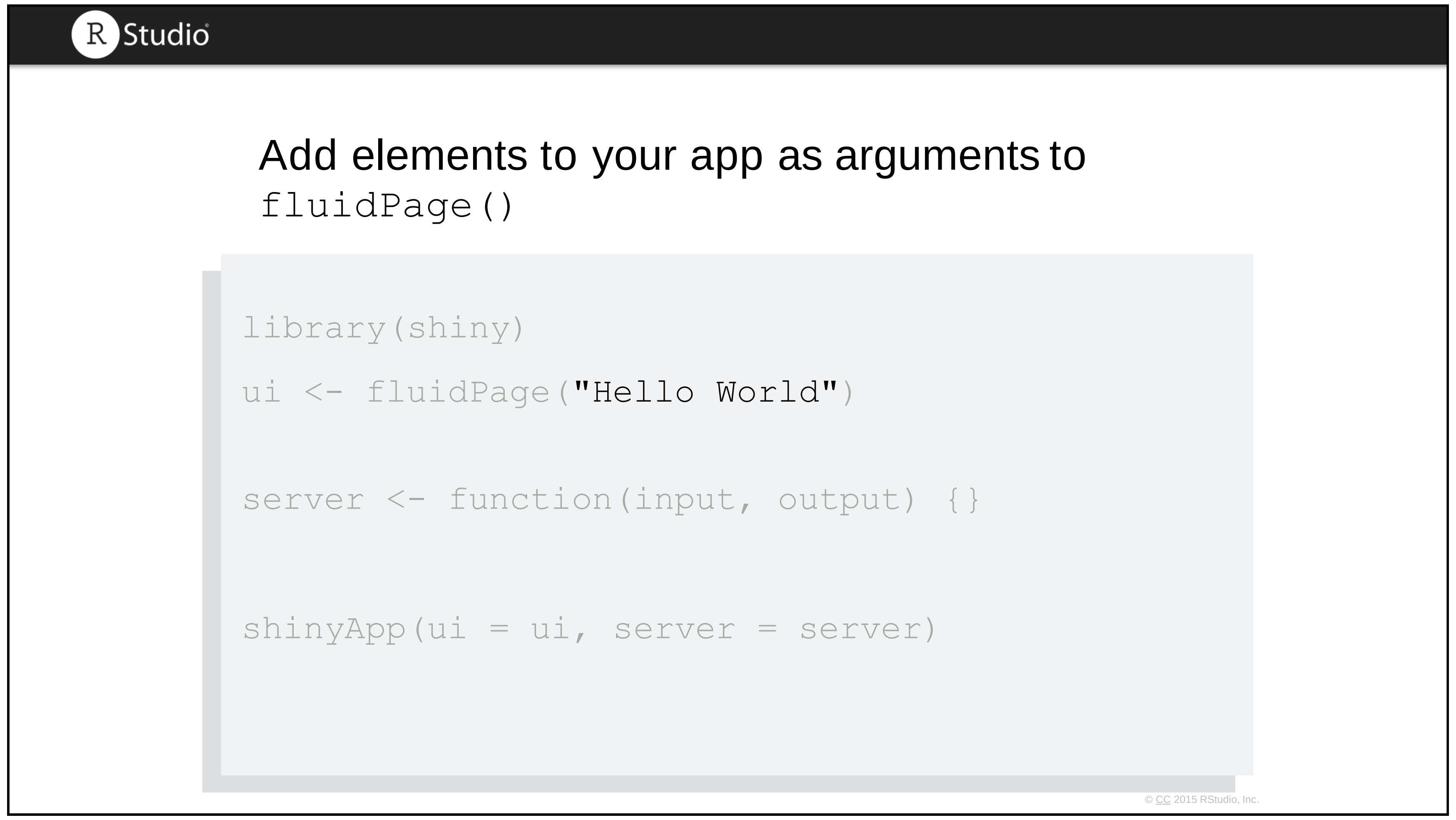
R Studio

Closing an app



```
# This is the server logic for a Shiny web application.
# You can find out more about building applications with
Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
```

```
> shiny::runApp()

Listening on http://127.0.0.1:6314
```

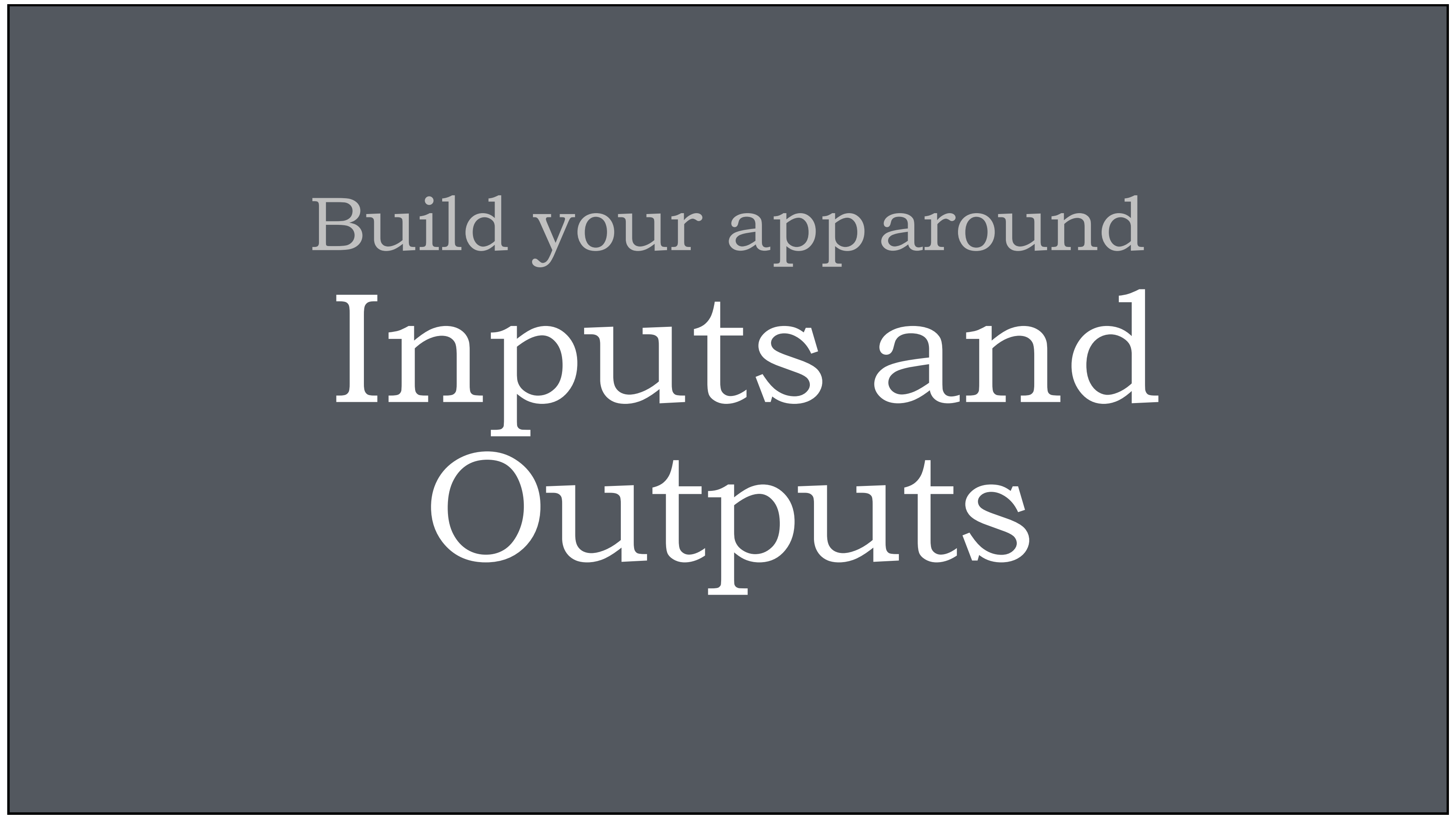


RStudio

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

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

server <- function(input, output) {}


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

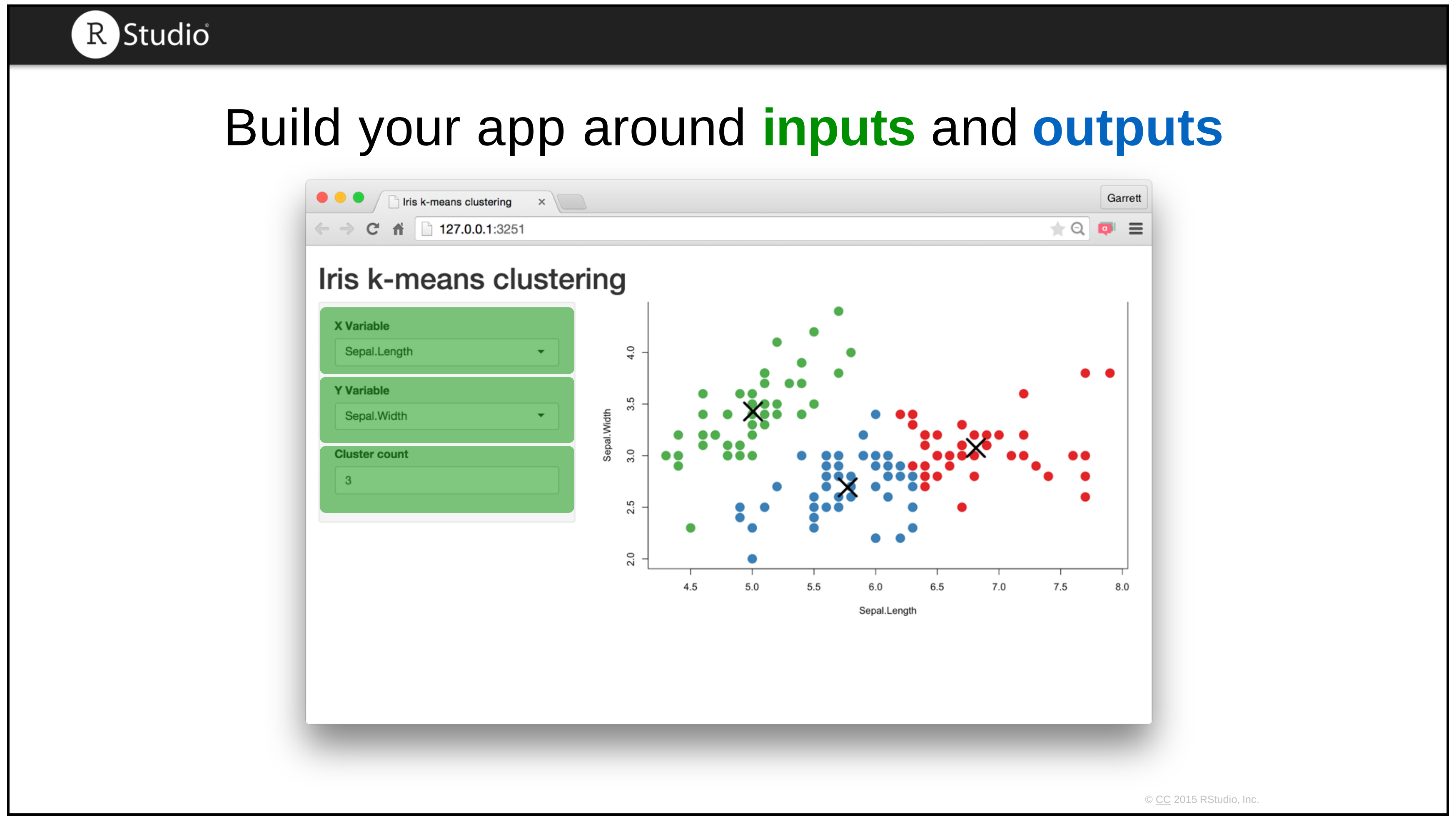
12



Build your app around

Inputs and Outputs

13



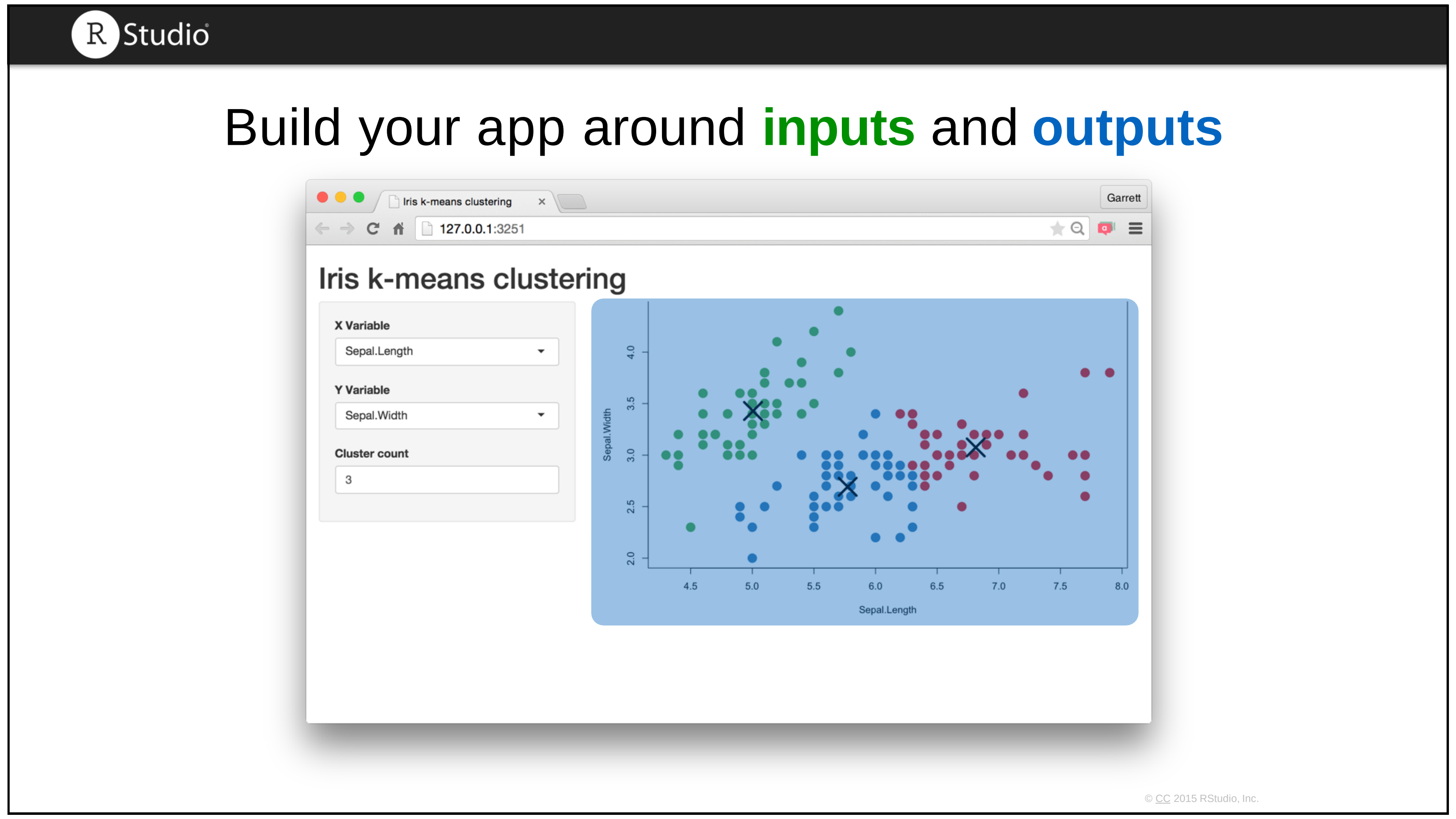
Build your app around **inputs** and **outputs**

Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

Cluster count: 3



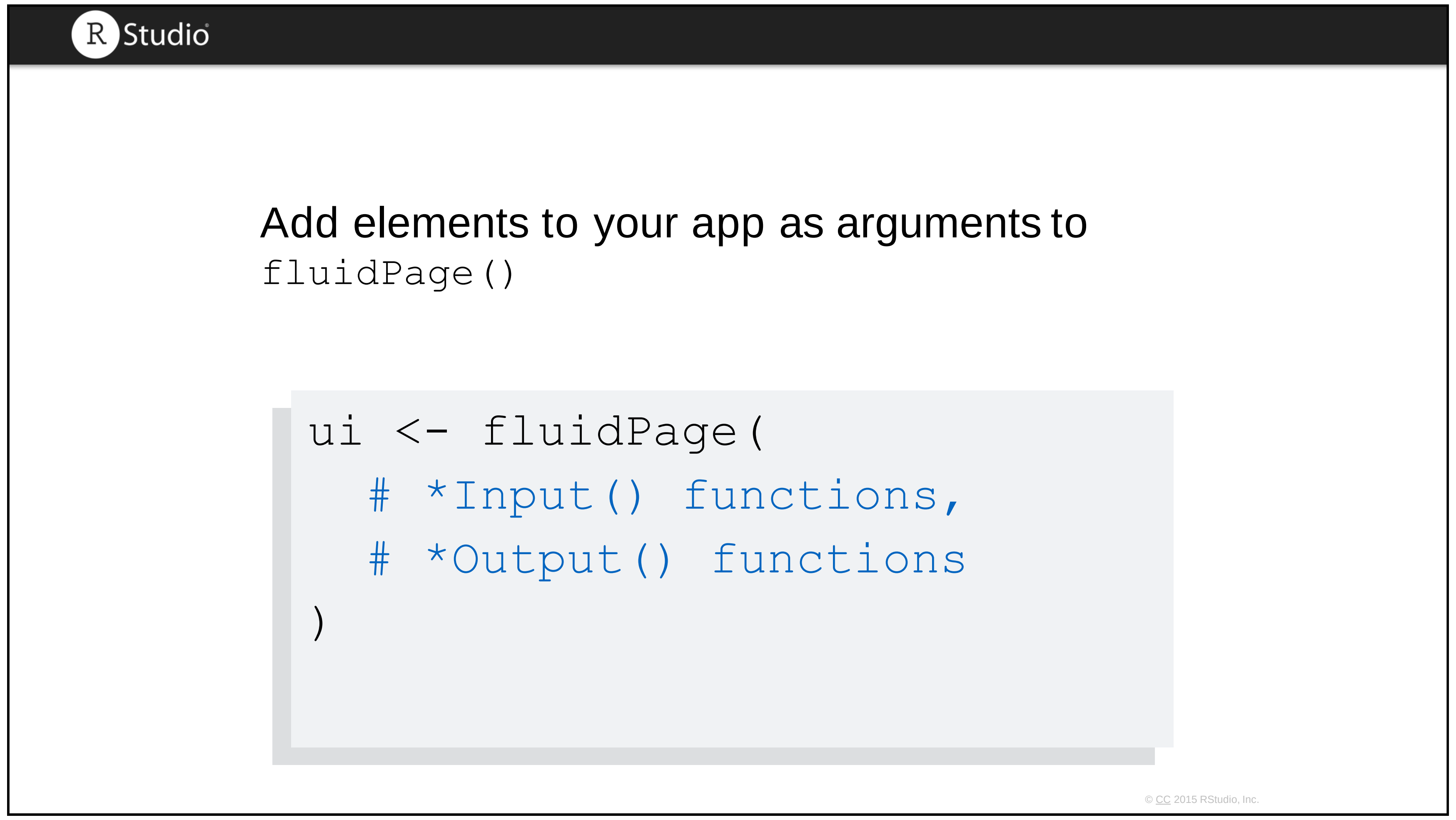
Build your app around **inputs** and **outputs**

Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

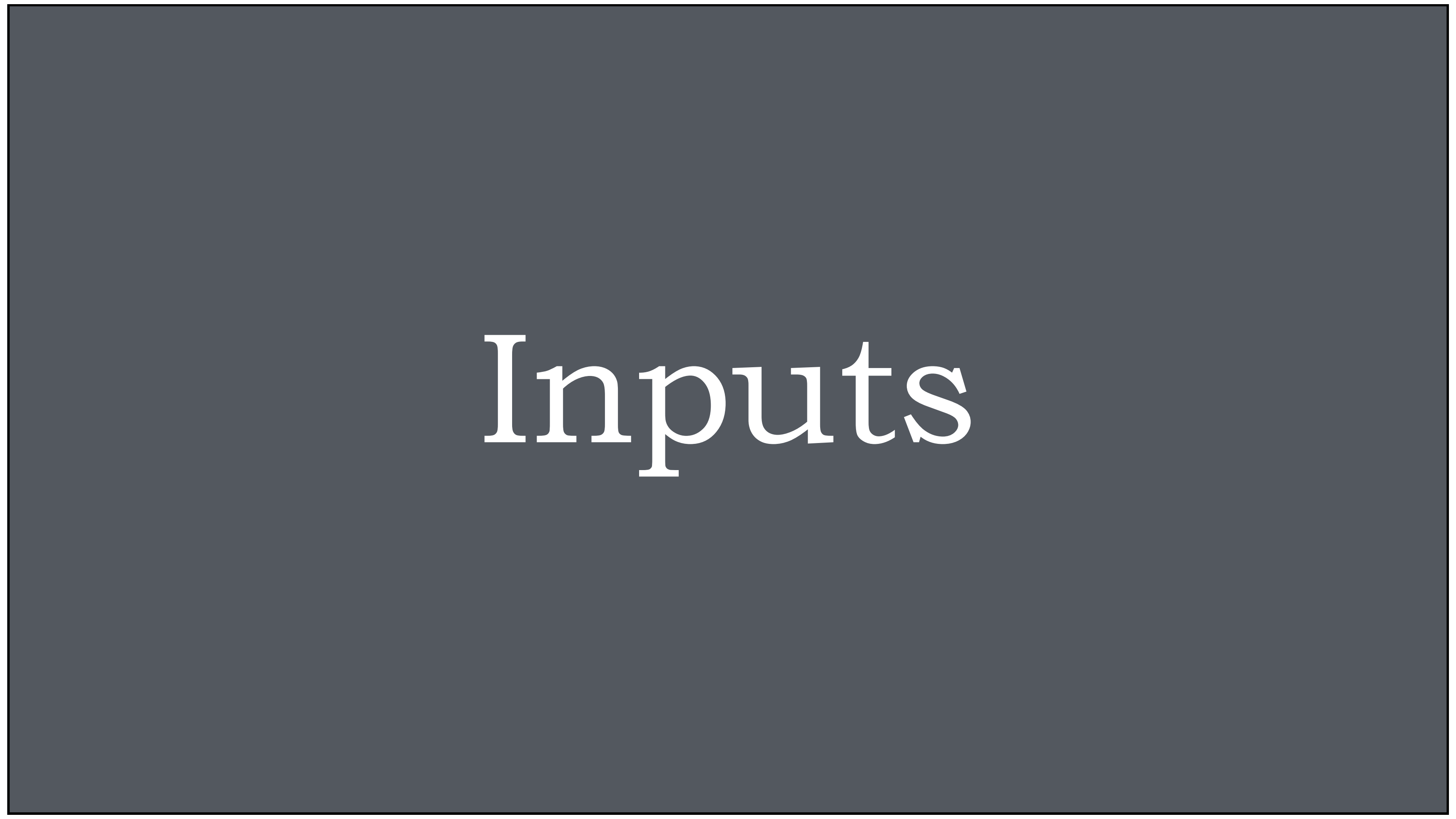
Cluster count: 3



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

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

16



Inputs

17

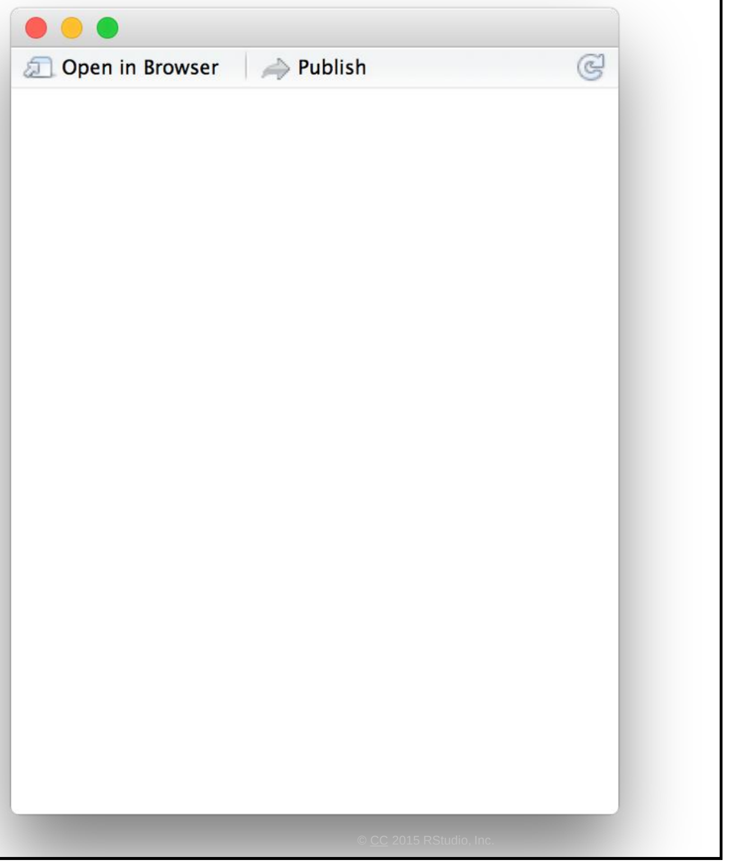
Create an input with an ***Input()** function.

```
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-prettify-separator="," data-keyboard="true"
   data-keyboard-step="1.01010101010101"/>
</div>
```

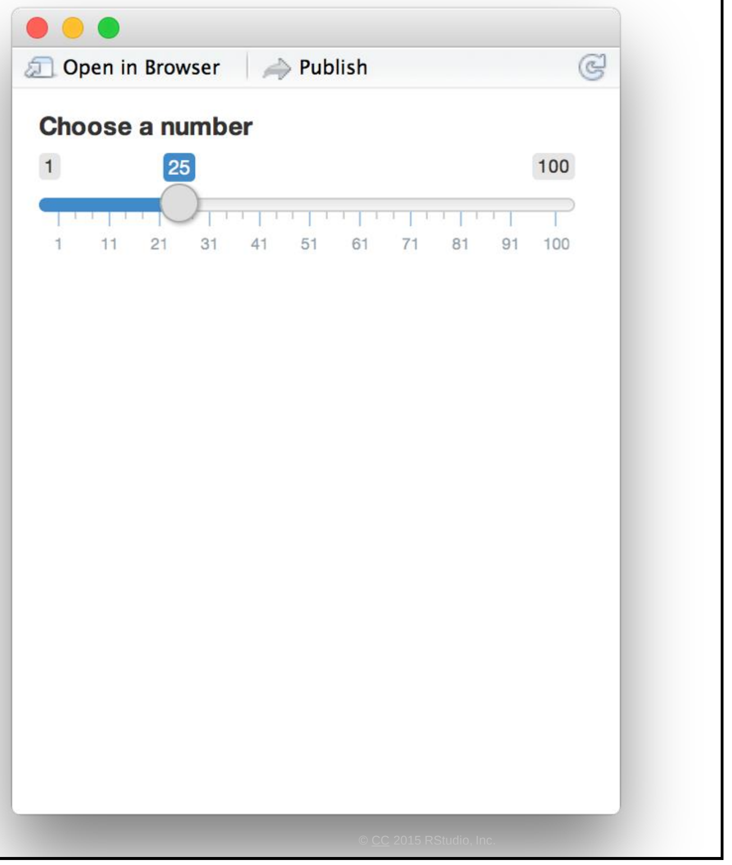
Create an input with an input function.

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



)

server <- function(input, output) {}

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

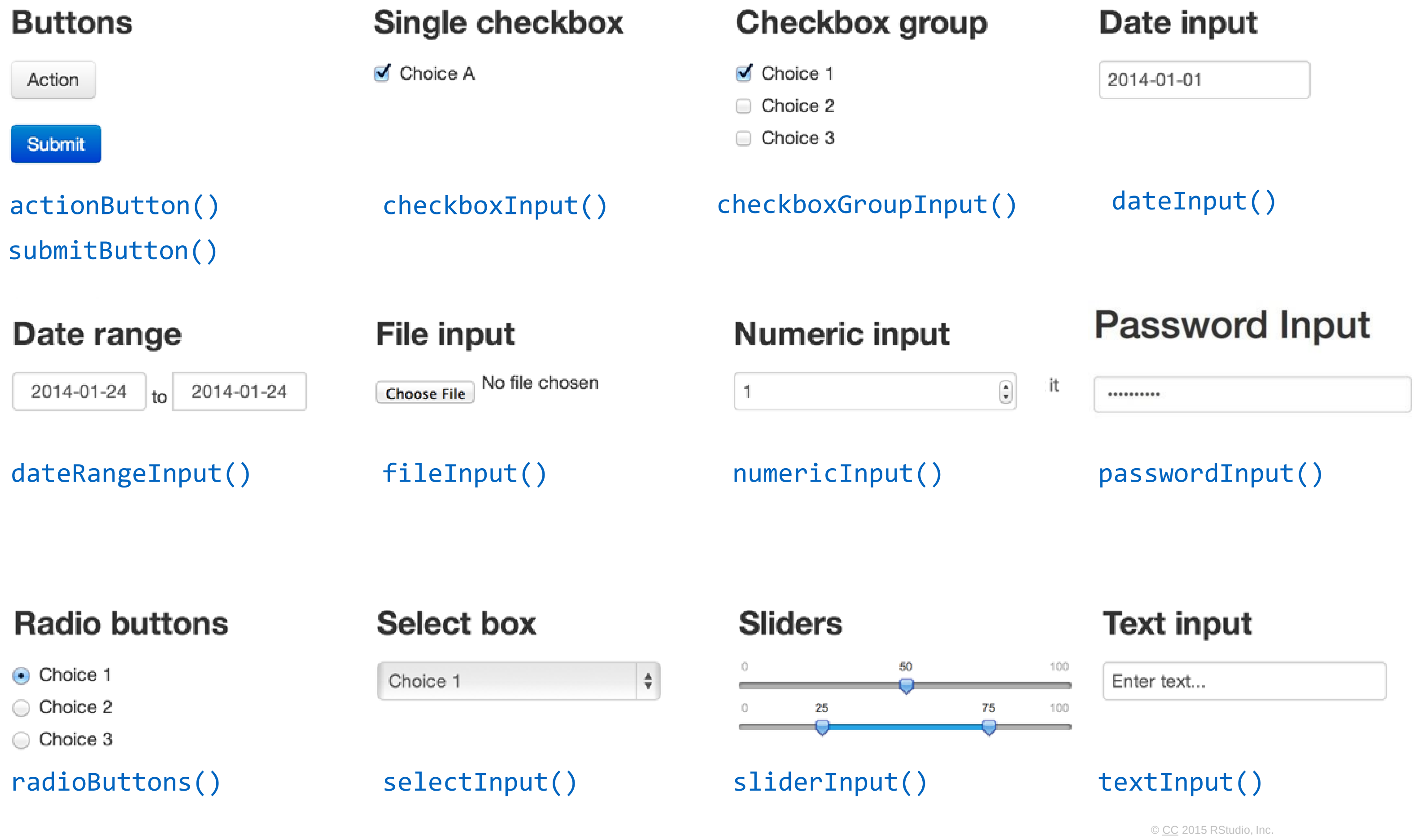


Create an input with an input function.

```
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)
```



20



Buttons

`actionButton()`
`submitButton()`

Single checkbox

`checkboxInput()`

Checkbox group

`checkboxGroupInput()`

Date input

`dateInput()`

Date range

`dateRangeInput()`

File input

`fileInput()`

Numeric input

`numericInput()`

Password Input

`passwordInput()`

Radio buttons

`radioButtons()`

Select box

`selectInput()`

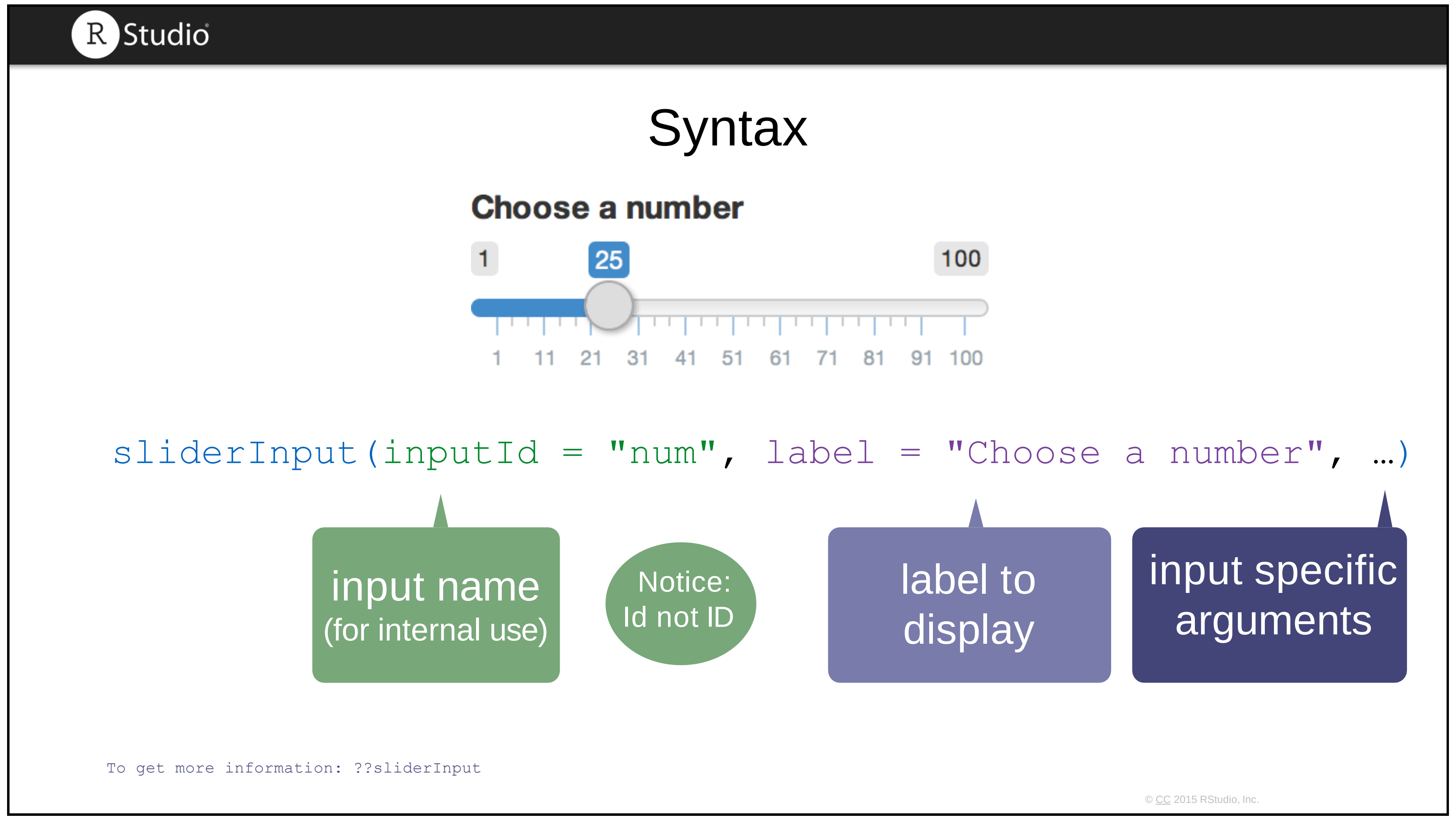
Sliders

`sliderInput()`

Text input

`textInput()`

21



Syntax

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

input name (for internal use)

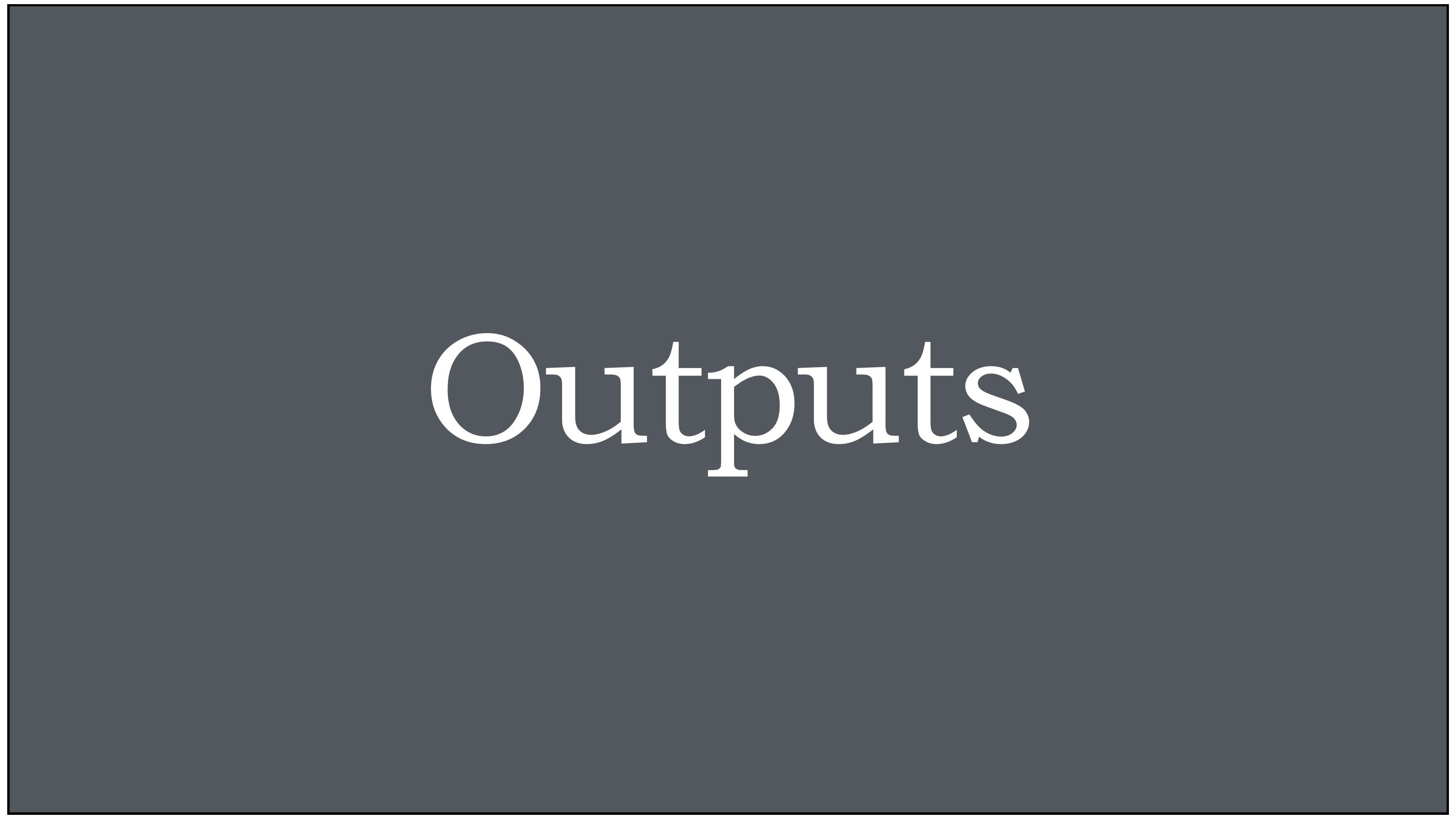
Notice: Id not ID

label to display

input specific arguments

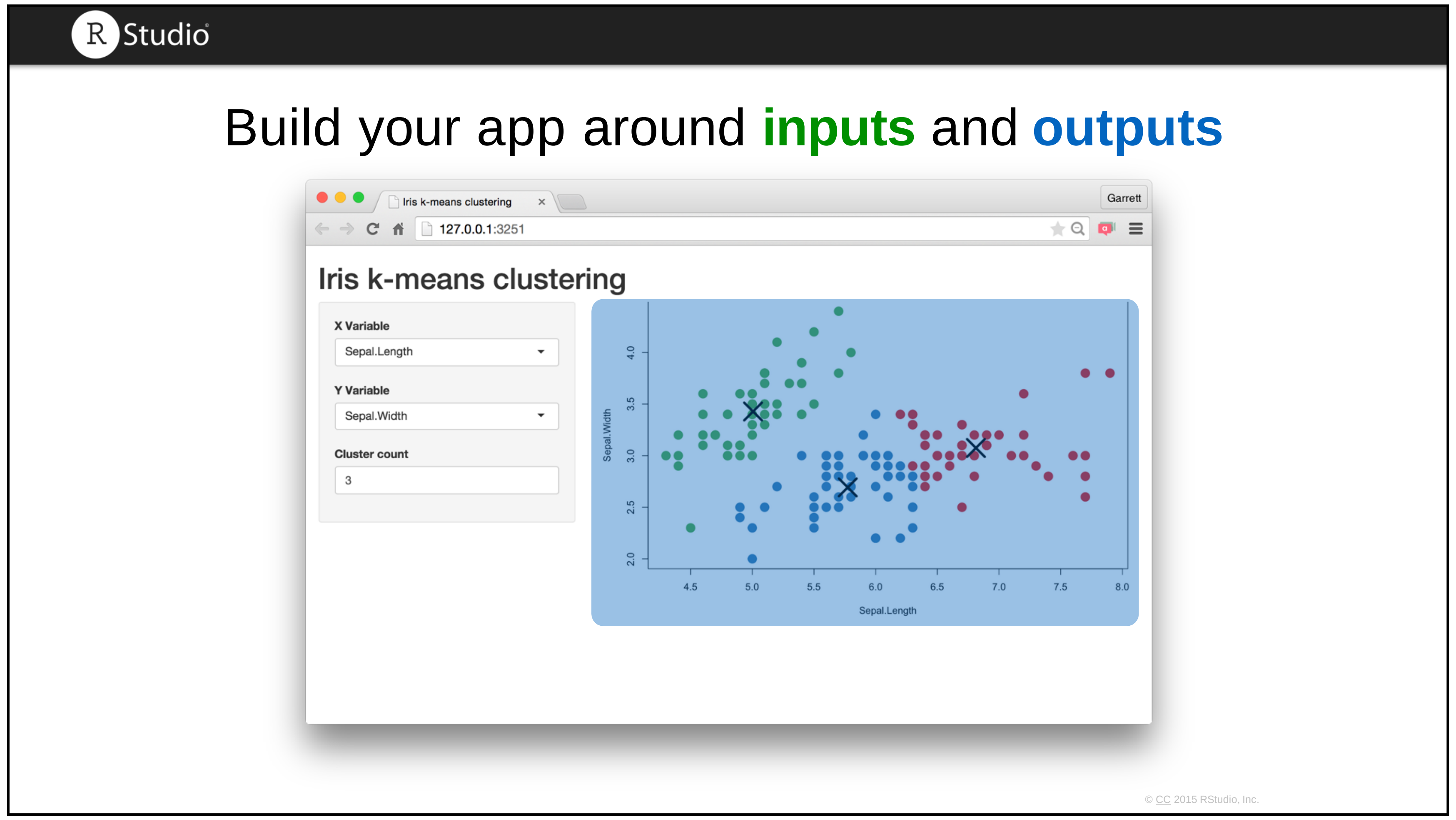
To get more information: ??sliderInput

22



Outputs

23



Build your app around inputs and outputs

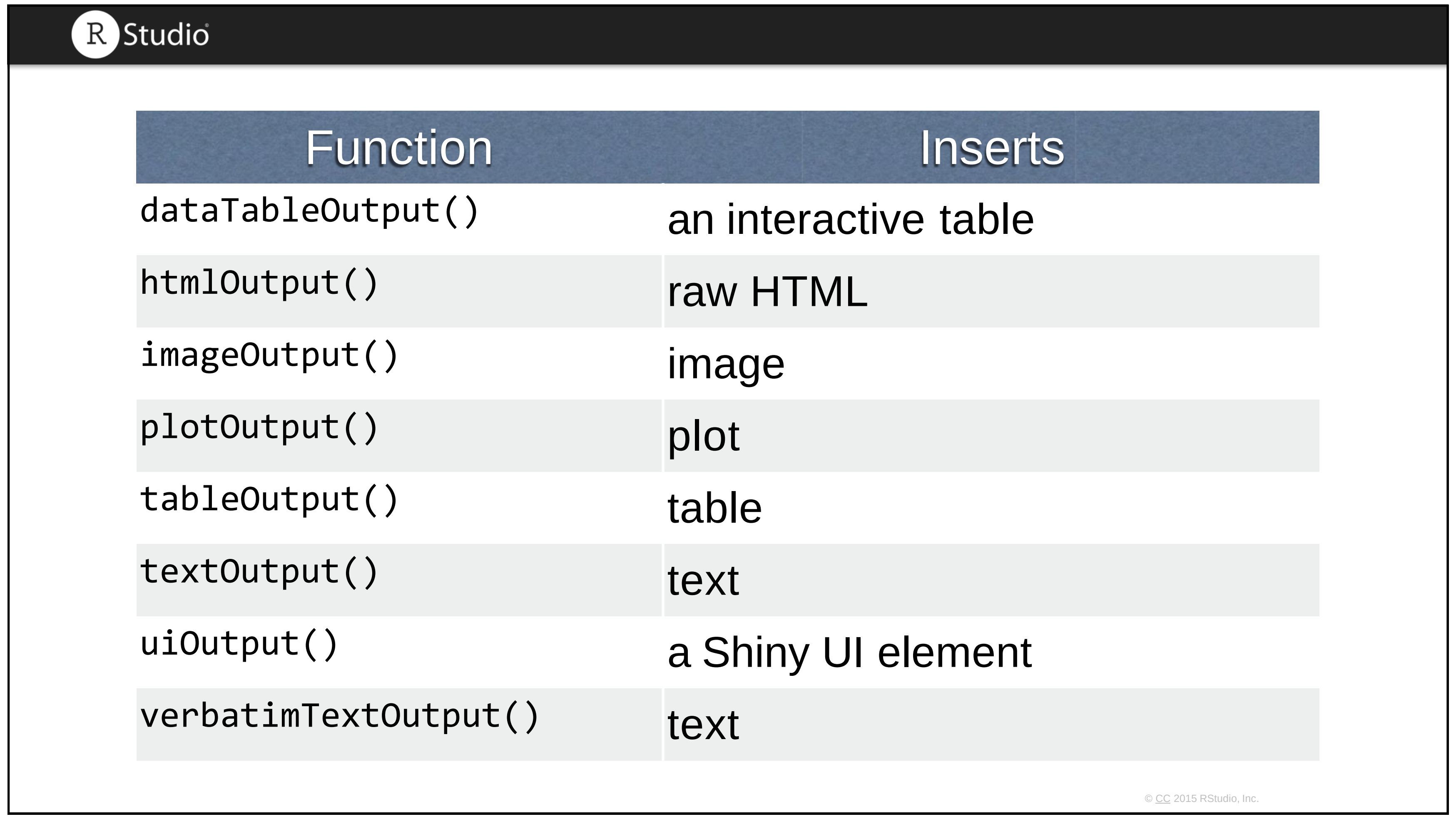
Iris k-means clustering

X Variable: Sepal.Length

Y Variable: Sepal.Width

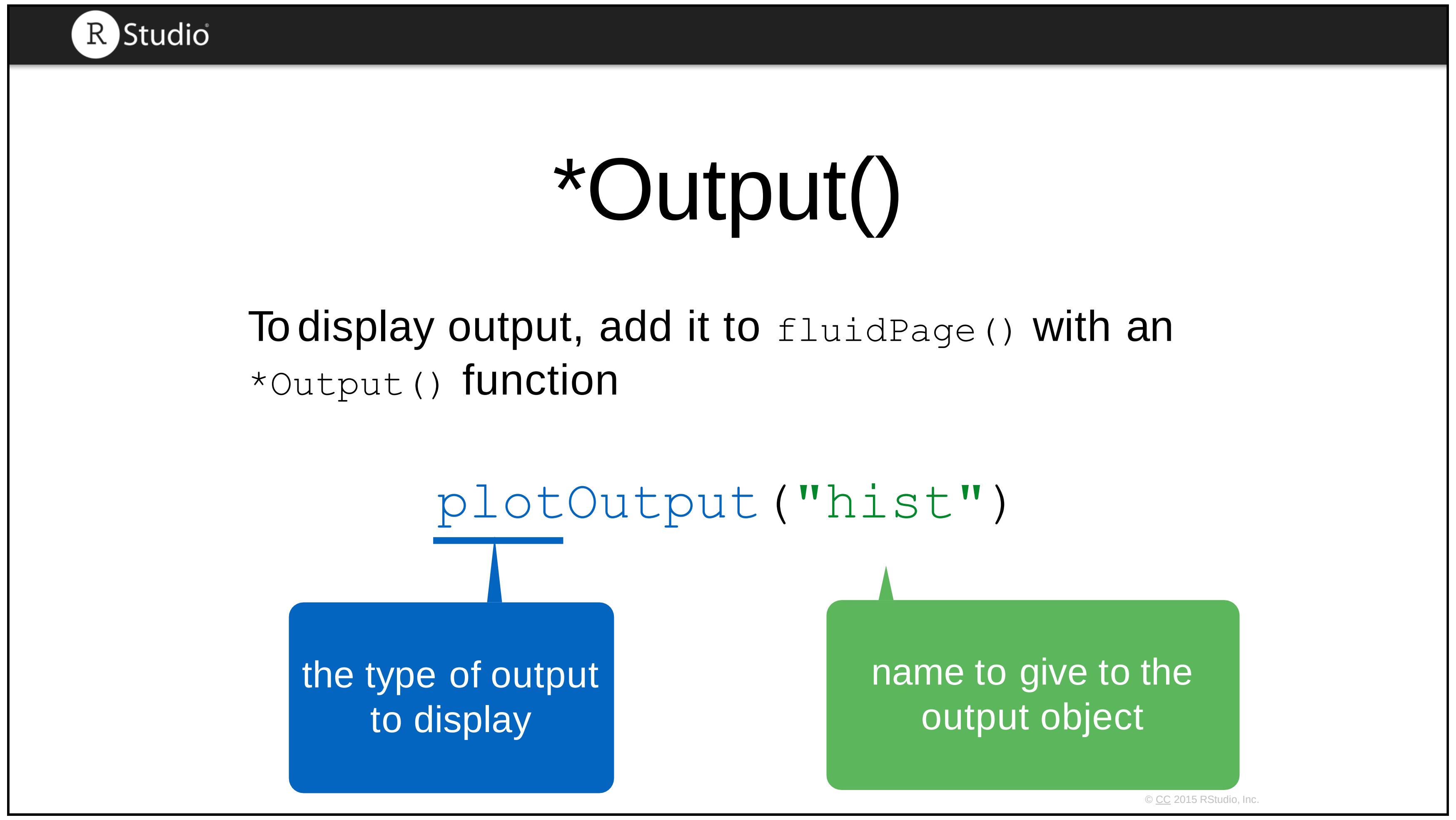
Cluster count: 3

24



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

25



R Studio

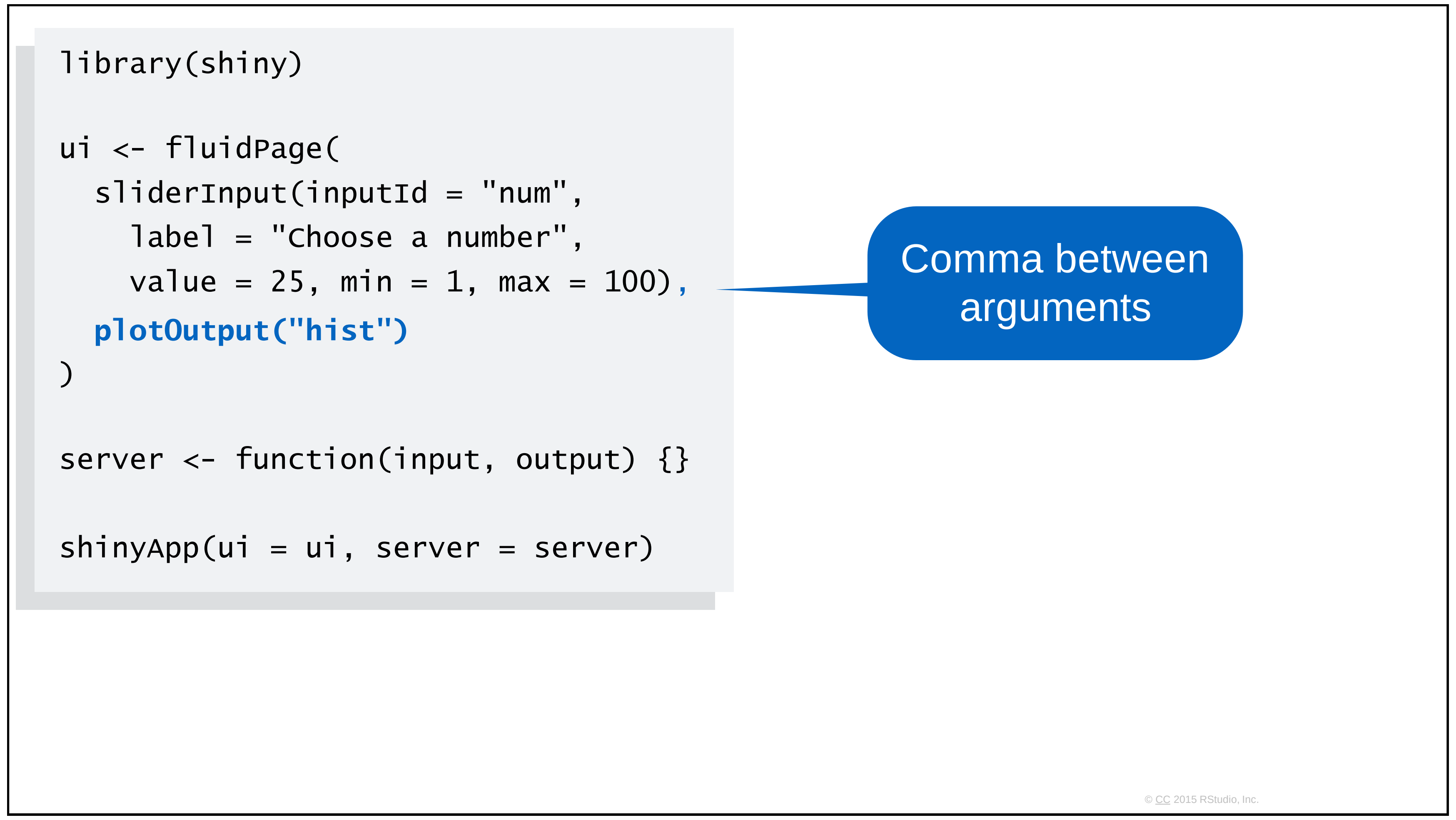
*Output()

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

26

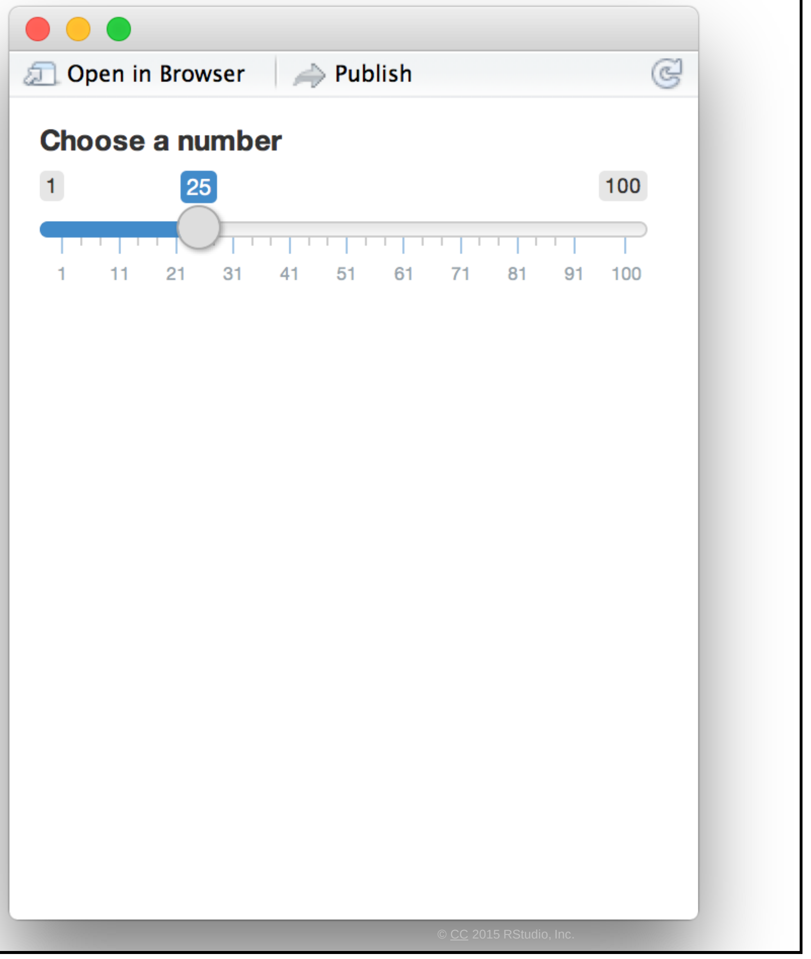


```
library(shiny)

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

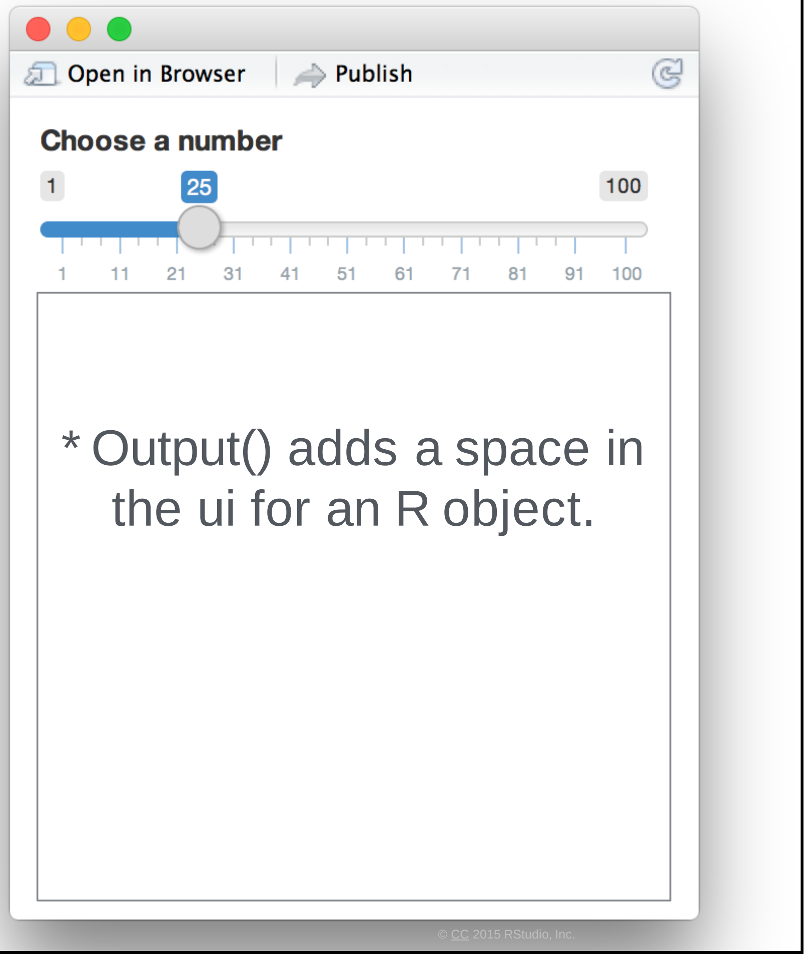
27

```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)
```



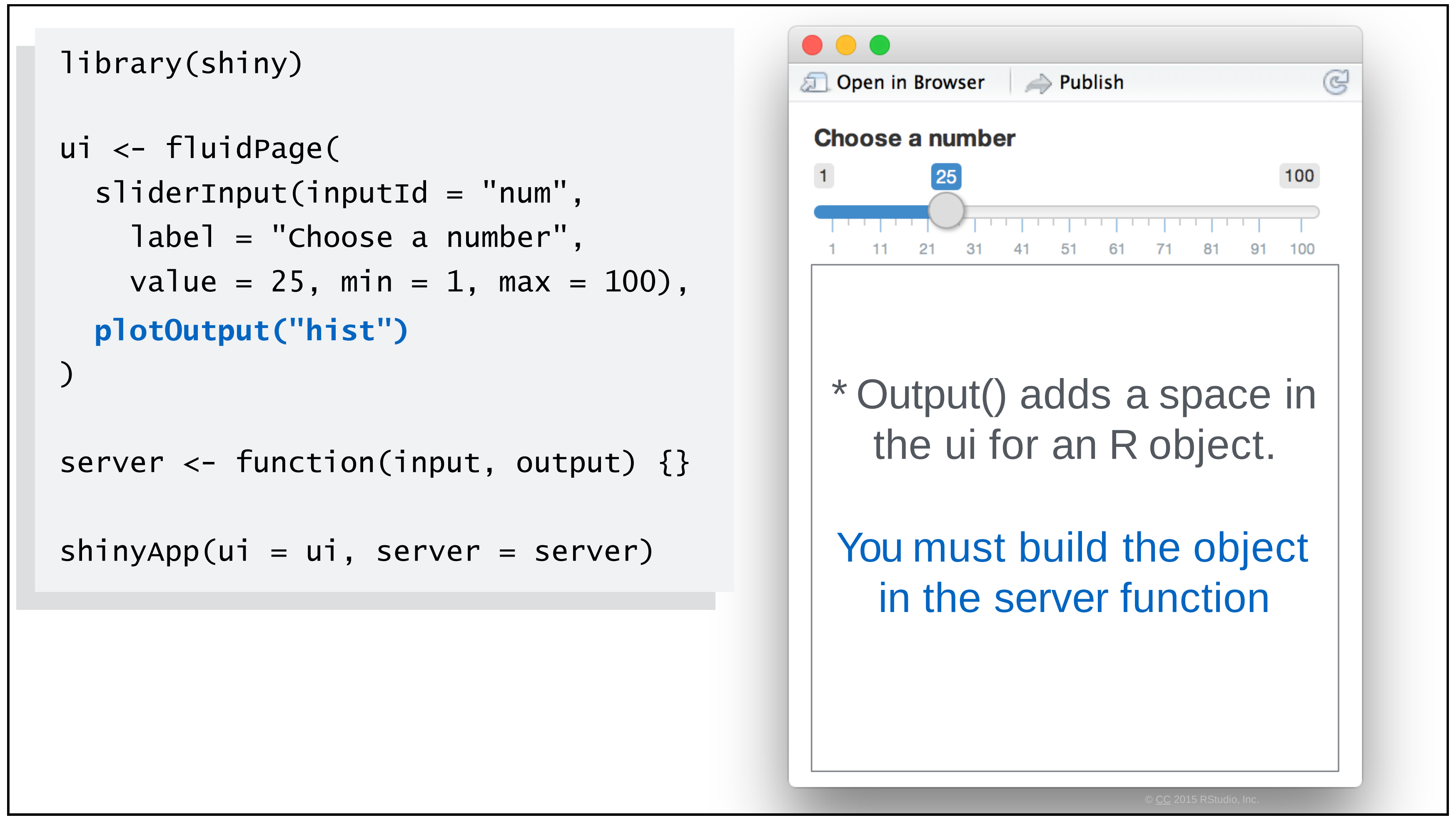
28

```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.

29



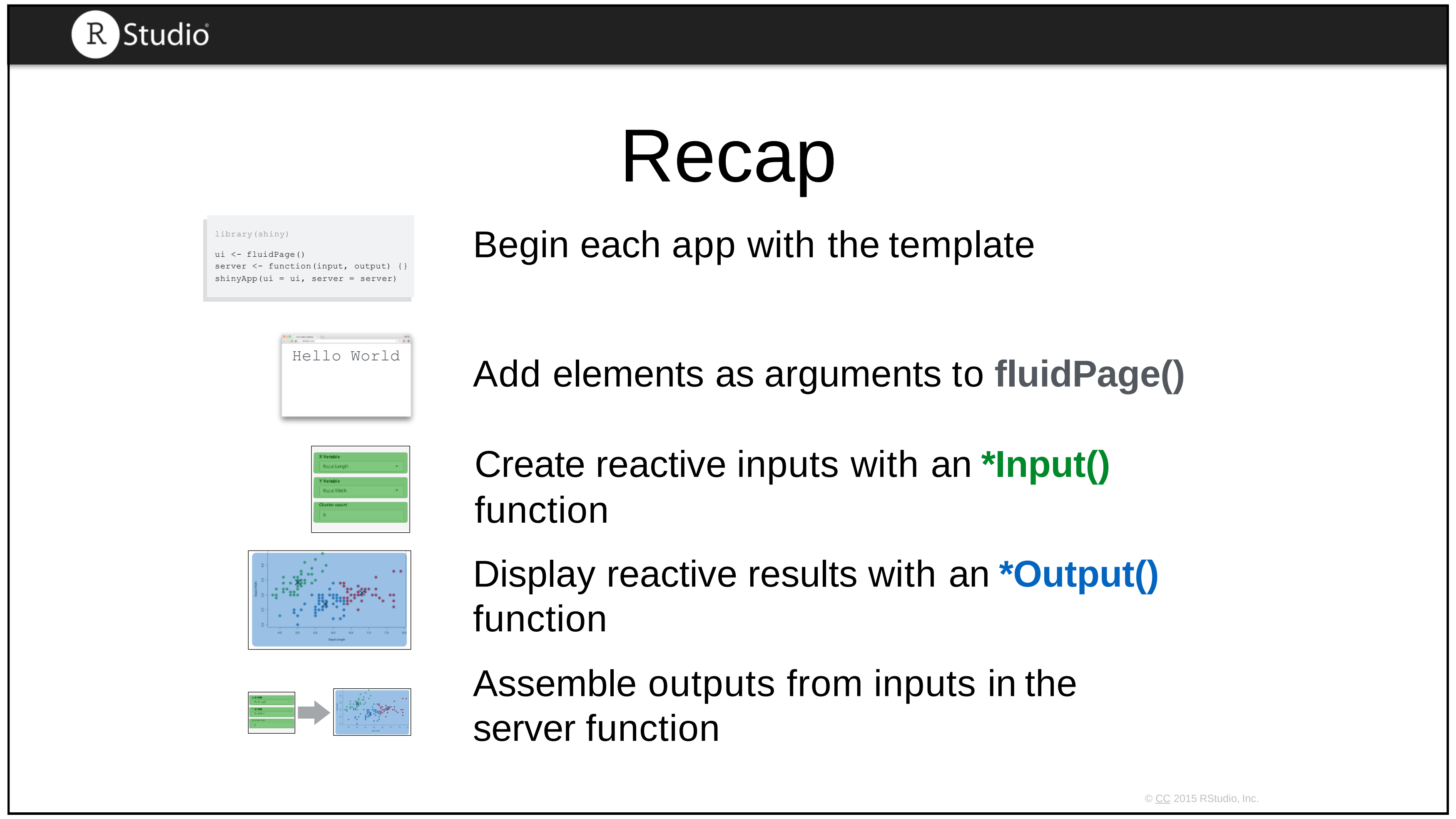
```
library(shiny)

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

© CC 2015 RStudio, Inc.

30



Recap

Begin each app with the template

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

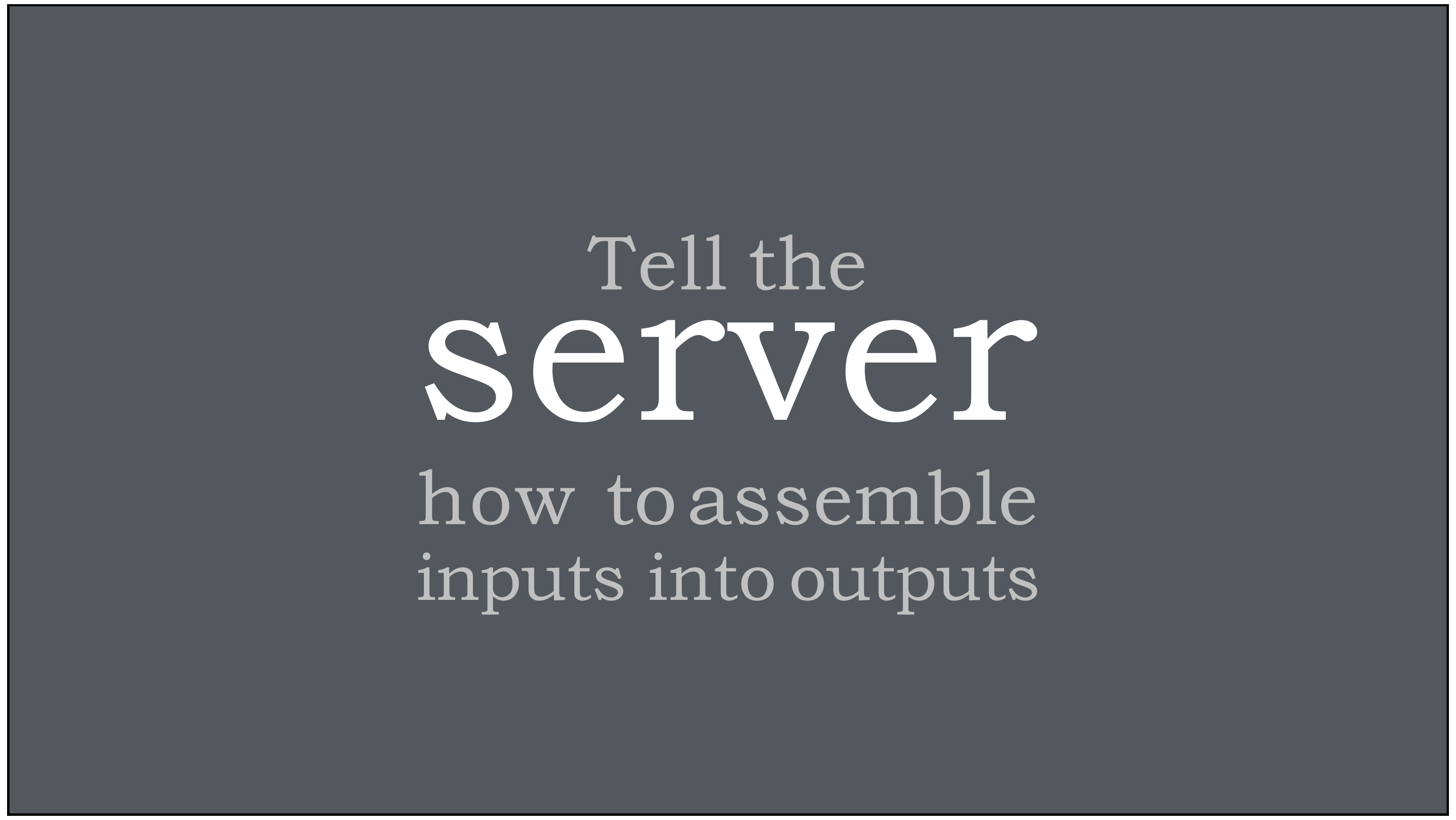
Create reactive inputs with an ***Input()** function

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

Assemble outputs from inputs in the server function

© CC 2015 RStudio, Inc.

31

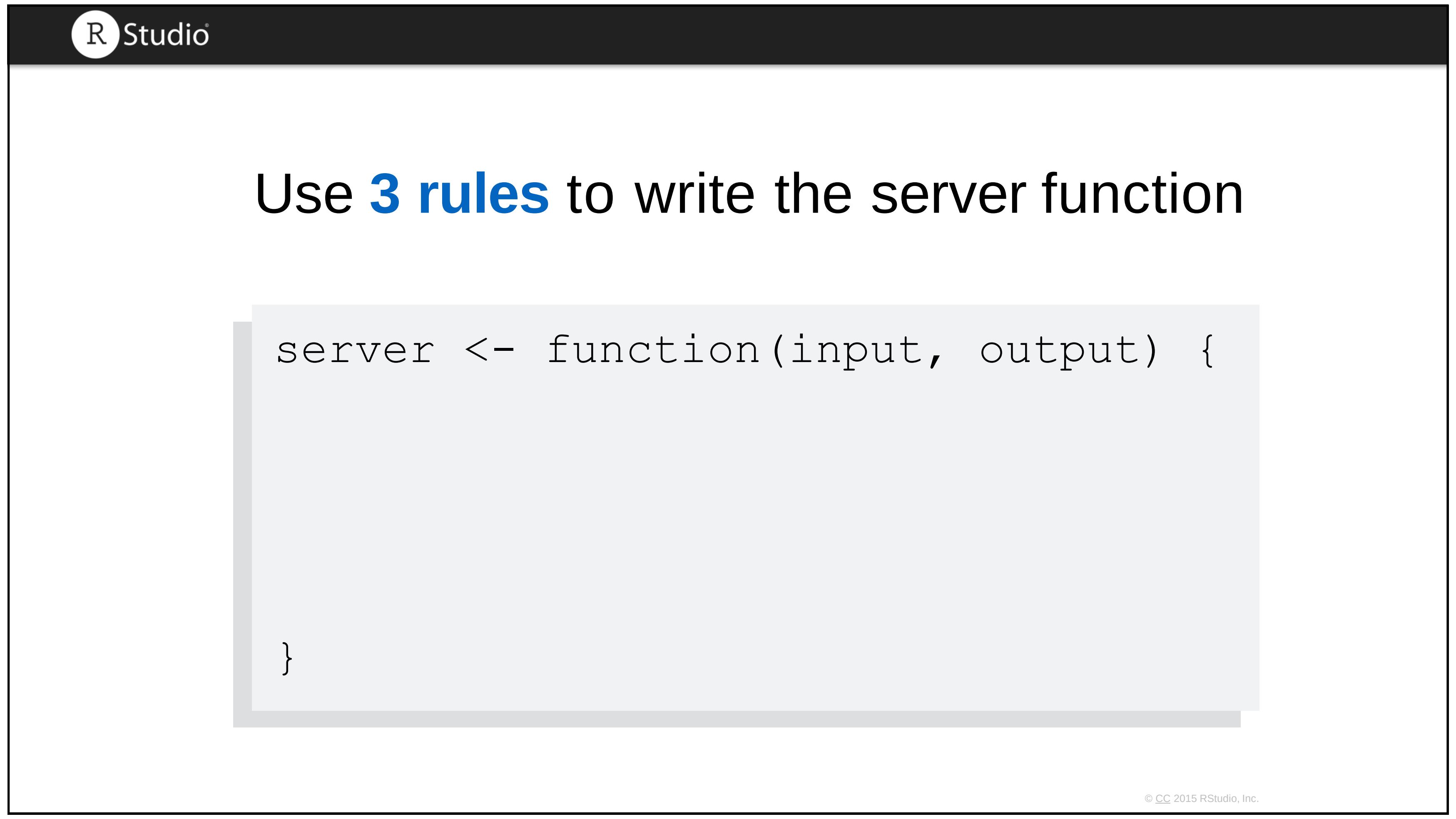


Tell the

server

how to assemble
inputs into outputs

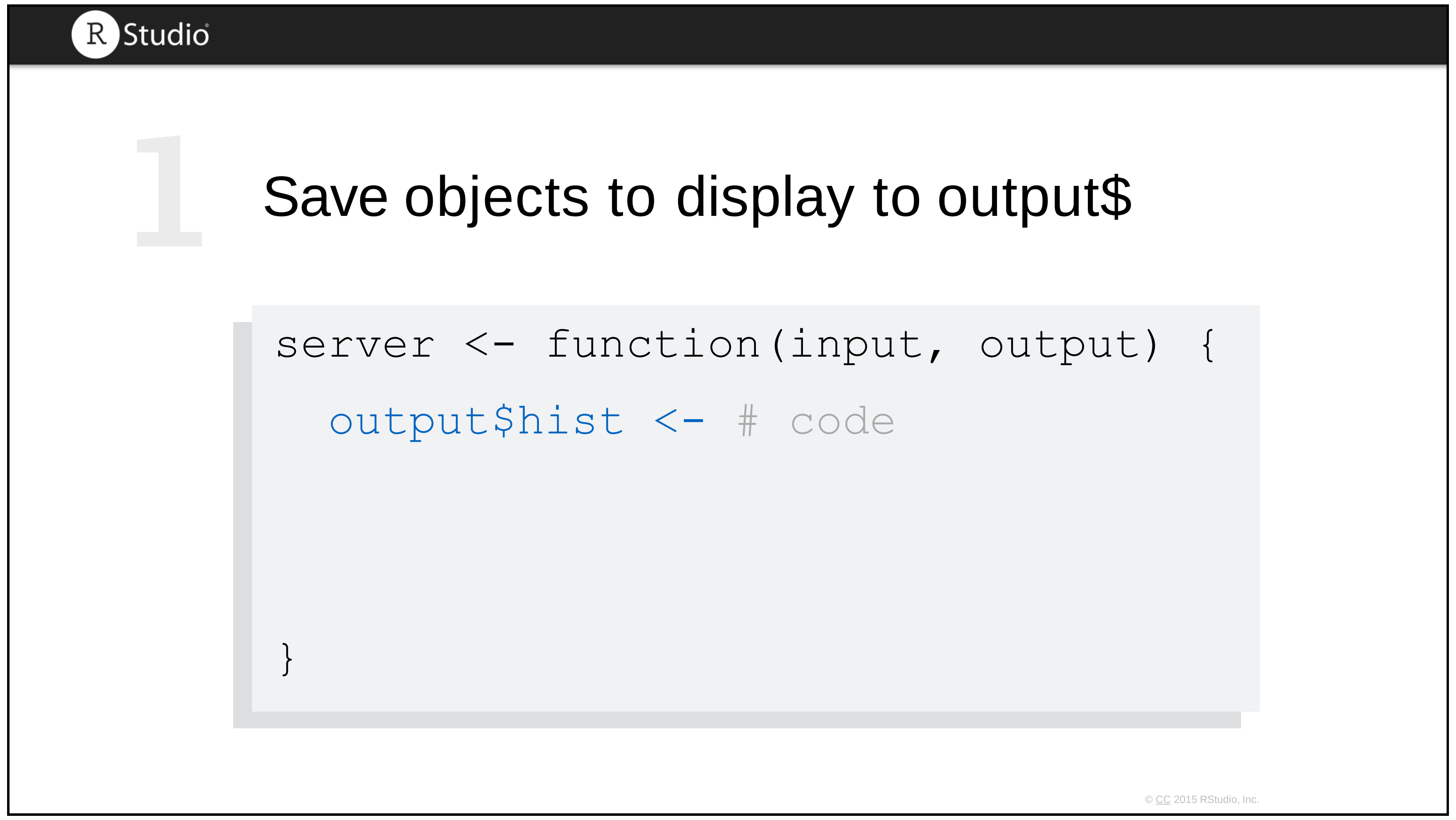
32



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

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




}
```

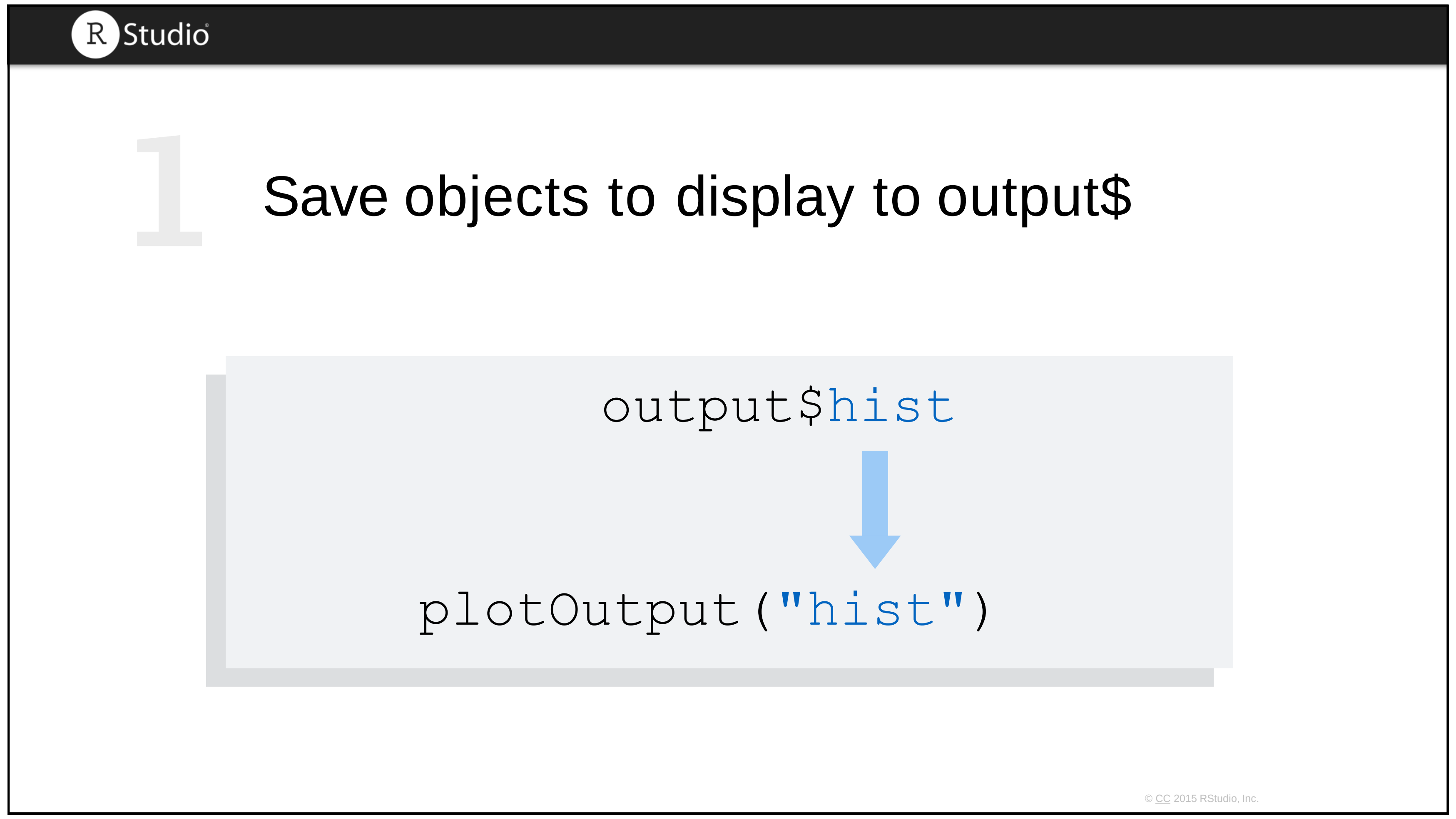
33



1 Save objects to display to output$

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



}
```

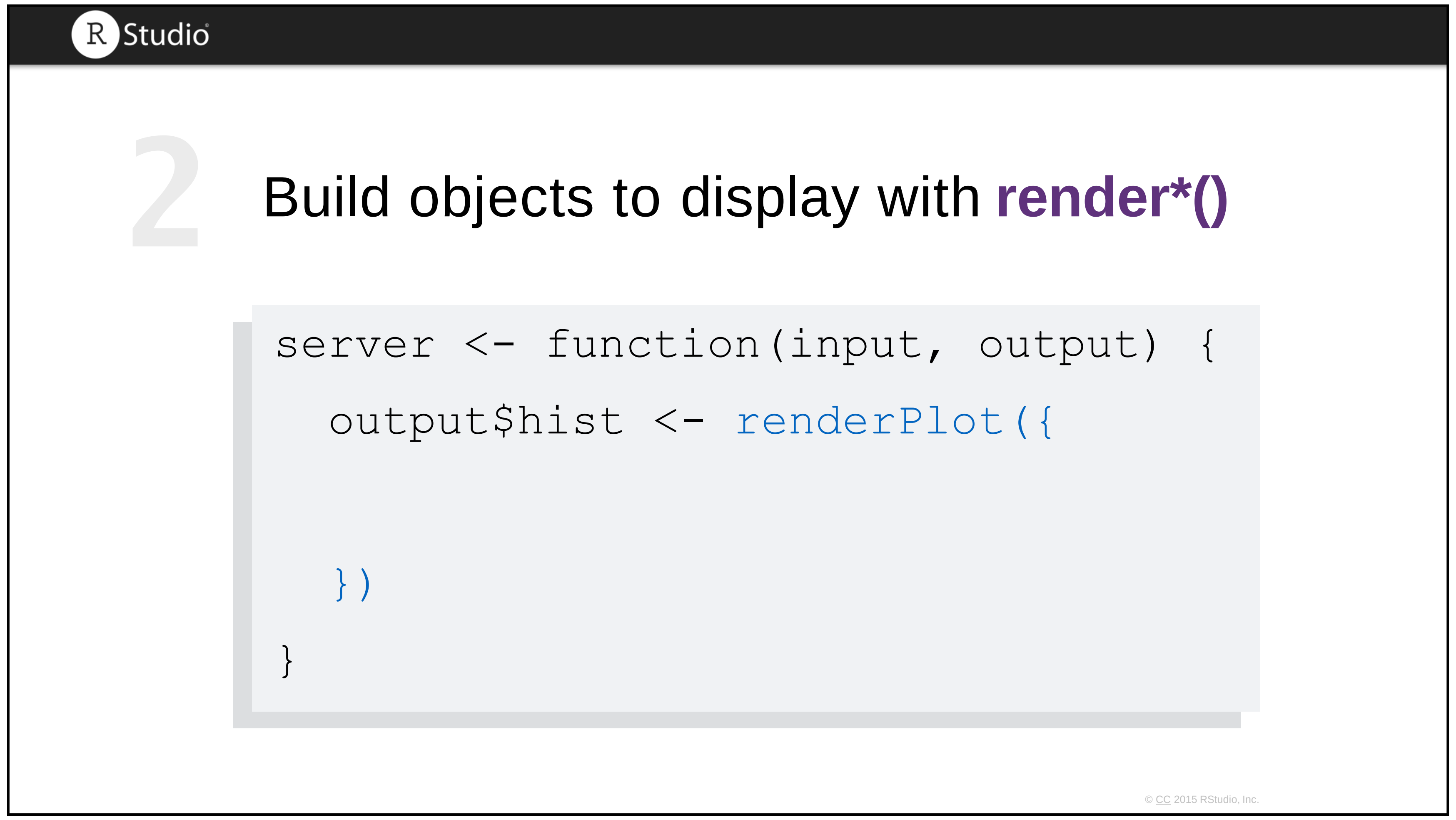
34



1 Save objects to display to output$

```
output$hist
    ↓
plotOutput("hist")
```

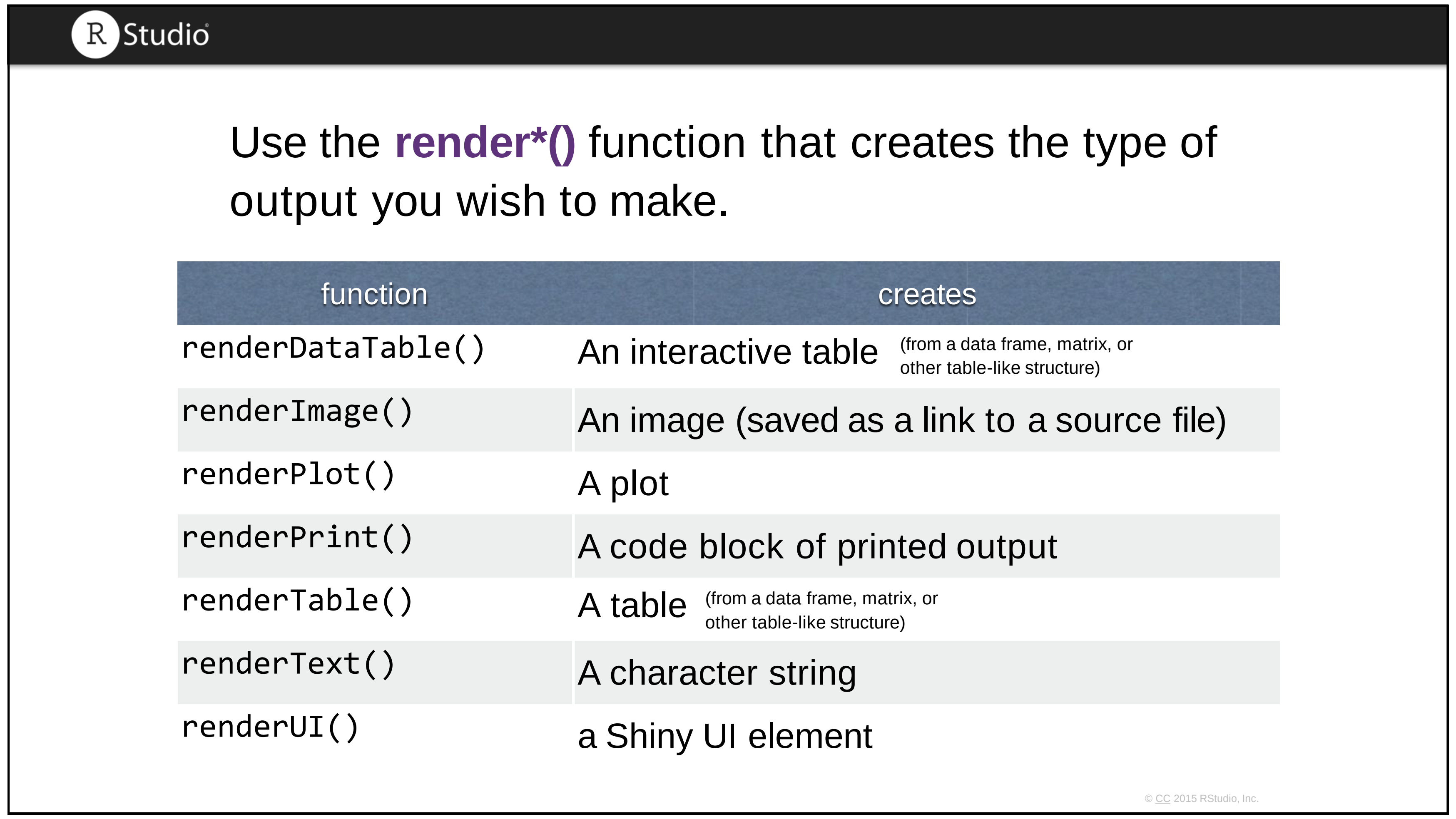
35



RStudio

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

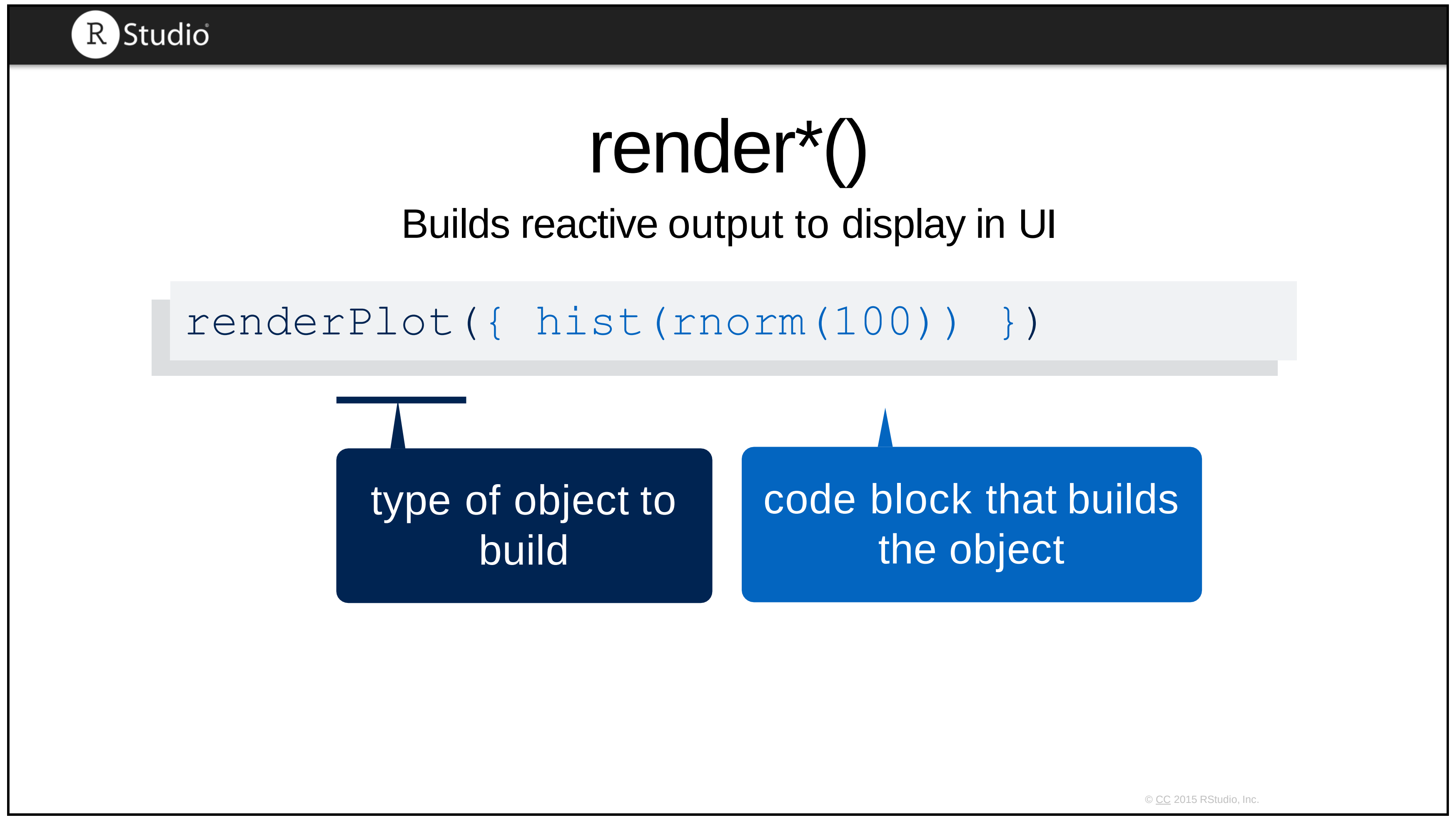
```
server <- function(input, output) {
  output$hist <- renderPlot({


  })
}
```



RStudio

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



RStudio

render*()

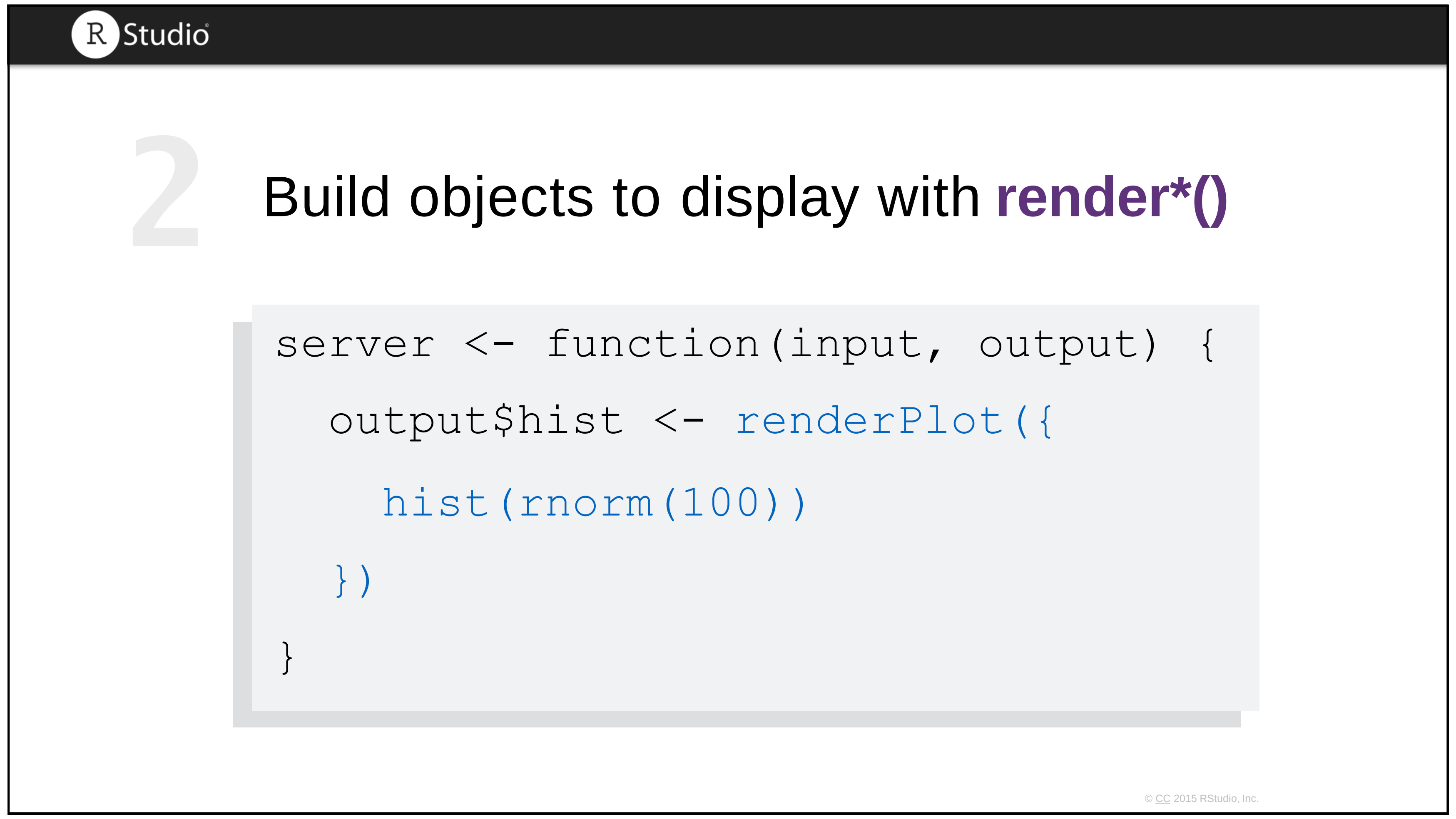
Builds reactive output to display in UI

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

type of object to build

code block that builds the object

38

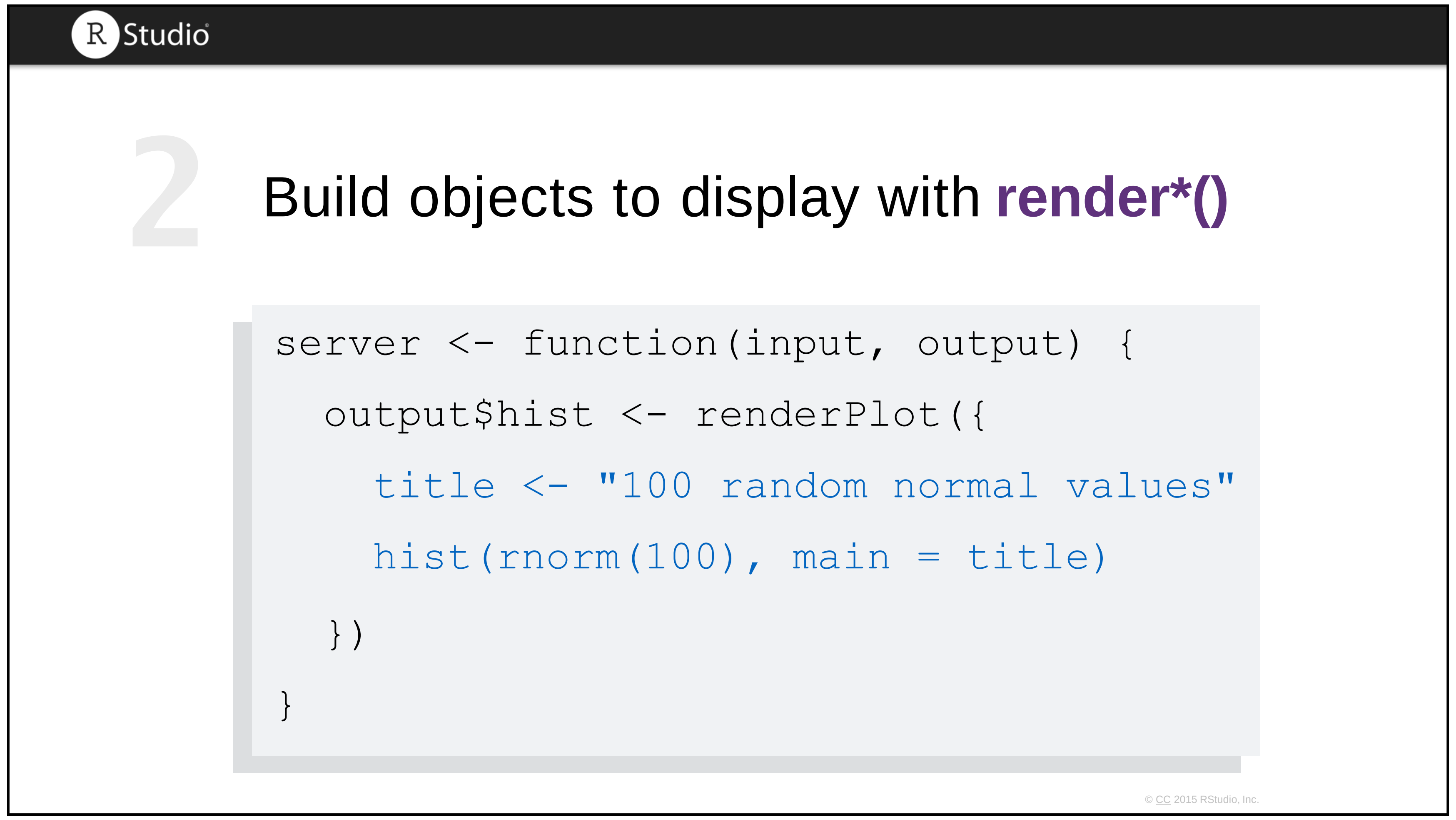


RStudio

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

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

39

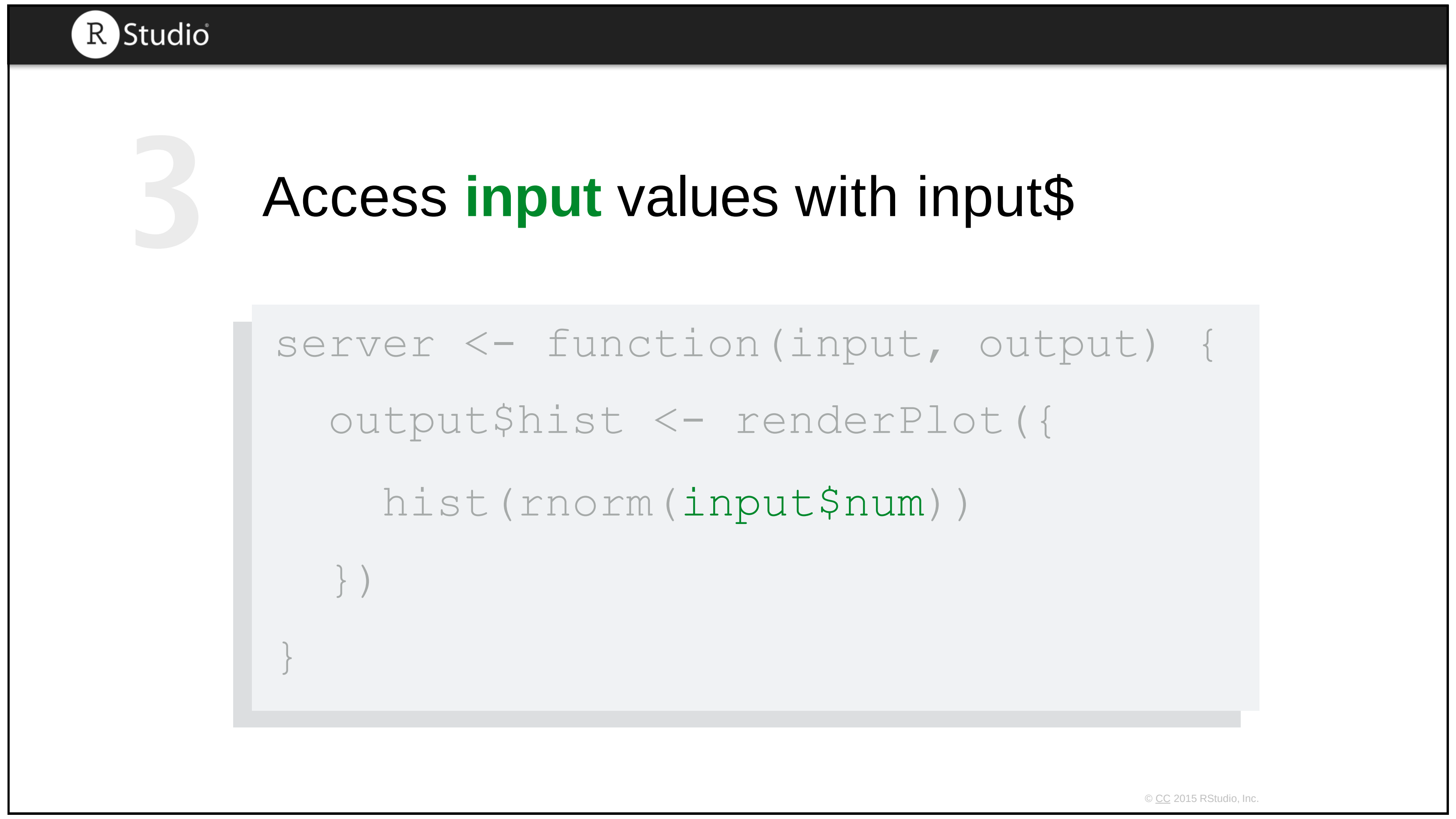


RStudio

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

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

40

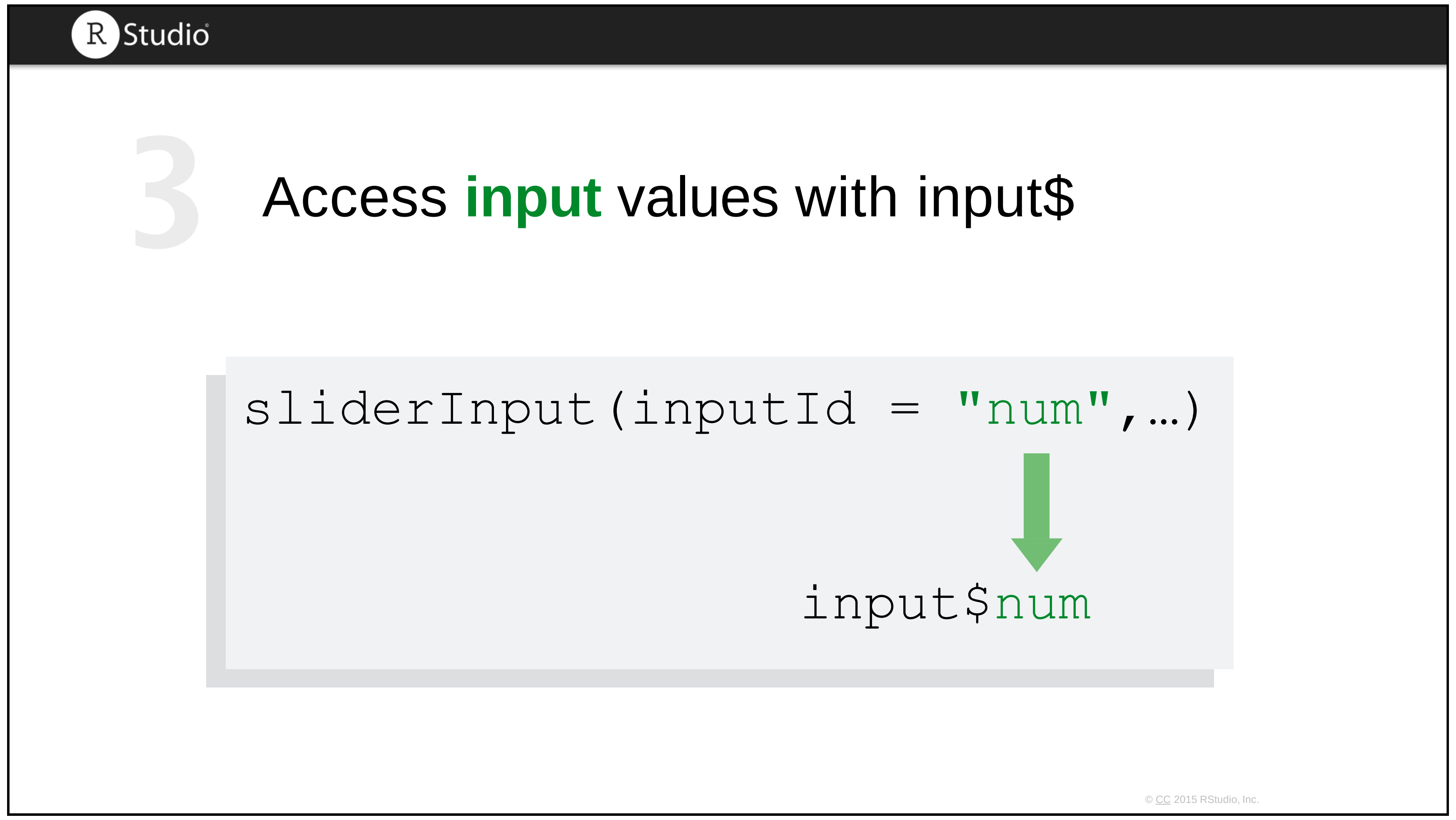


RStudio

3 Access **input** values with input$

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

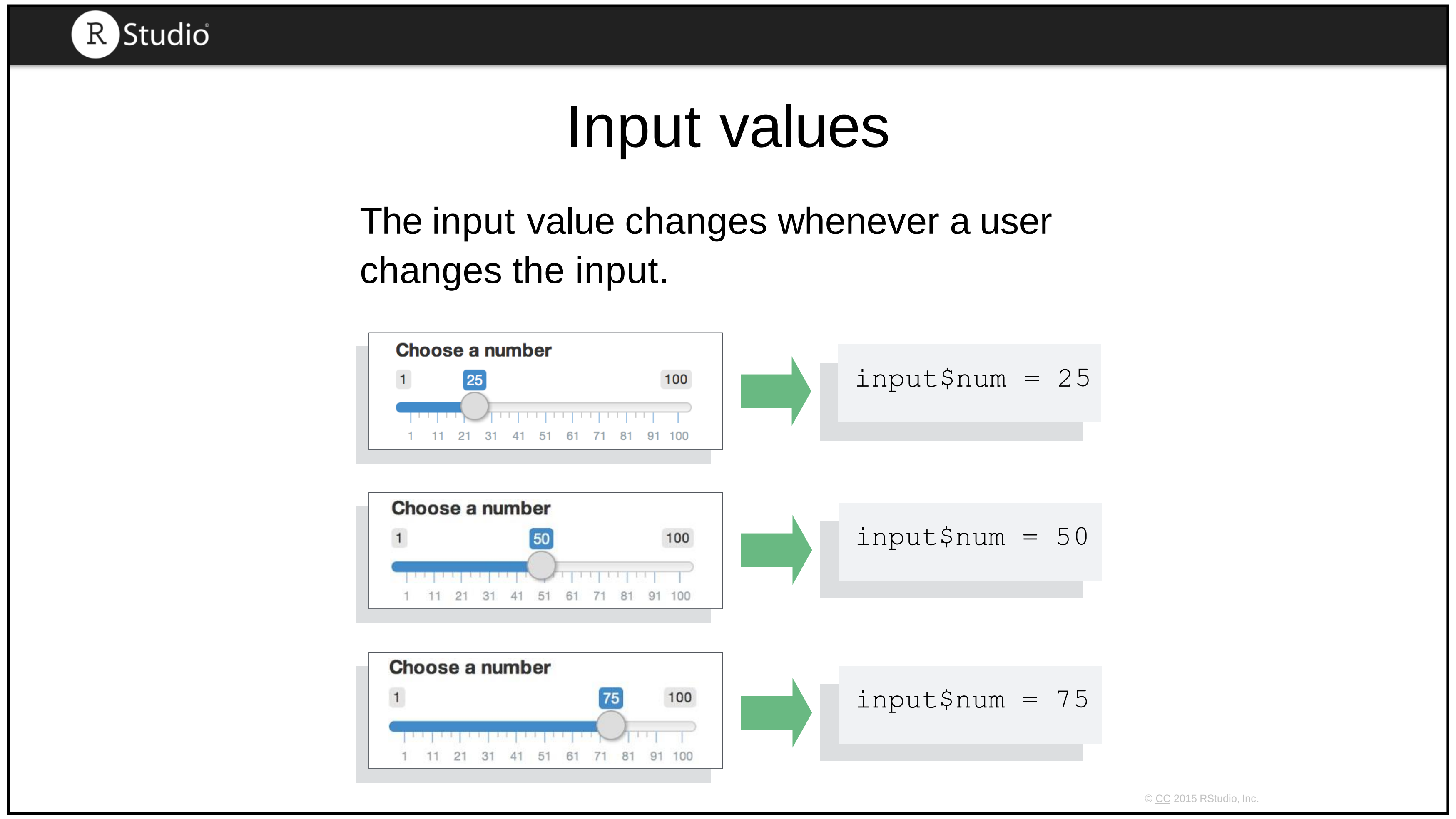
41



Access **input** values with input$

```
sliderInput(inputId = "num",…)

                    input$num
```

42



Input values

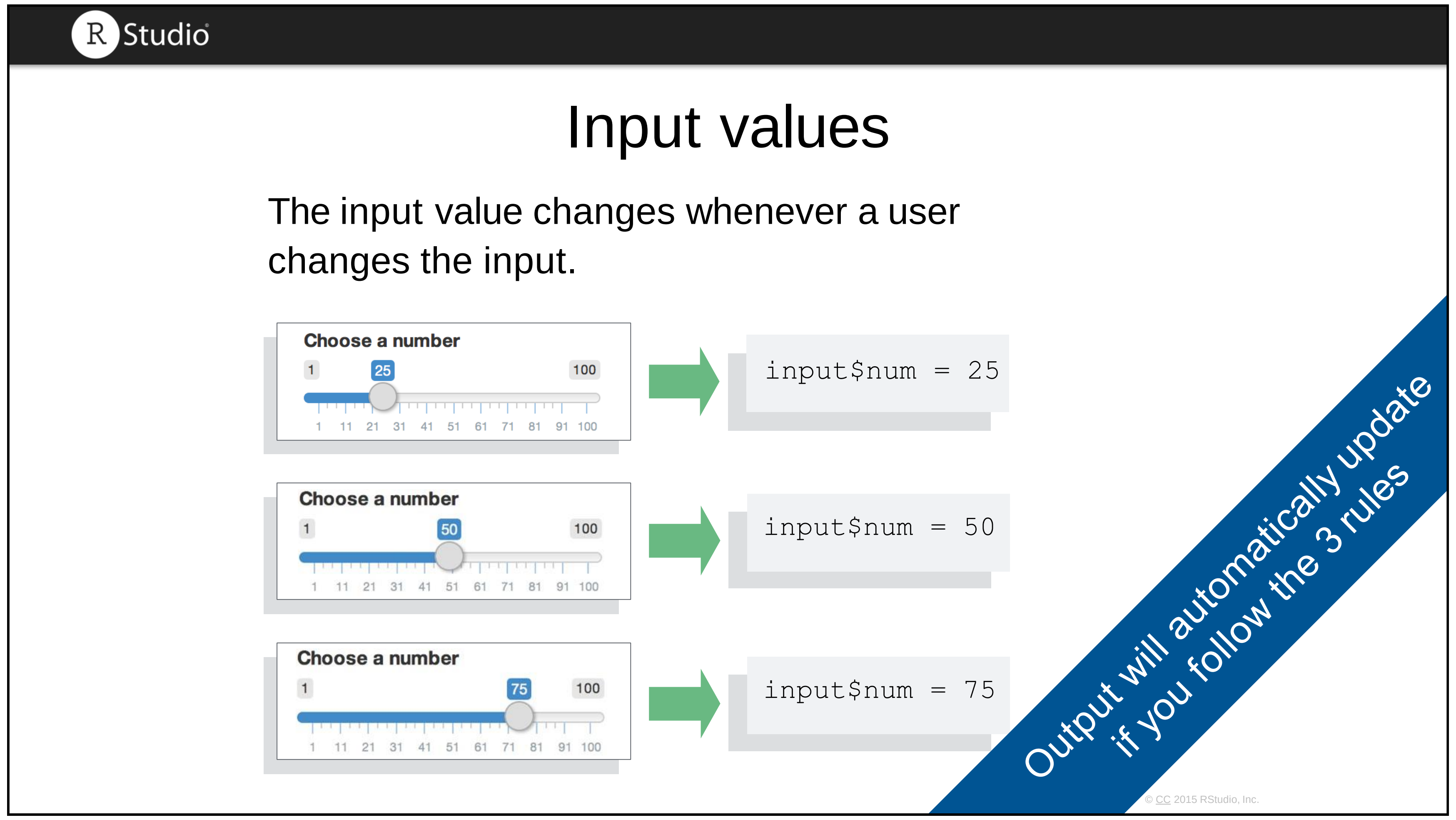
The input value changes whenever a user changes the input.

```
input$num = 25
```

```
input$num = 50
```

```
input$num = 75
```

43



Input values

The input value changes whenever a user changes the input.

Choose a number
1
25
100
1 11 21 31 41 51 61 71 81 91 100
Choose a number
1
50
100
1 11 21 31 41 51 61 71 81 91 100
Choose a number
1
75
100
1 11 21 31 41 51 61 71 81 91 100

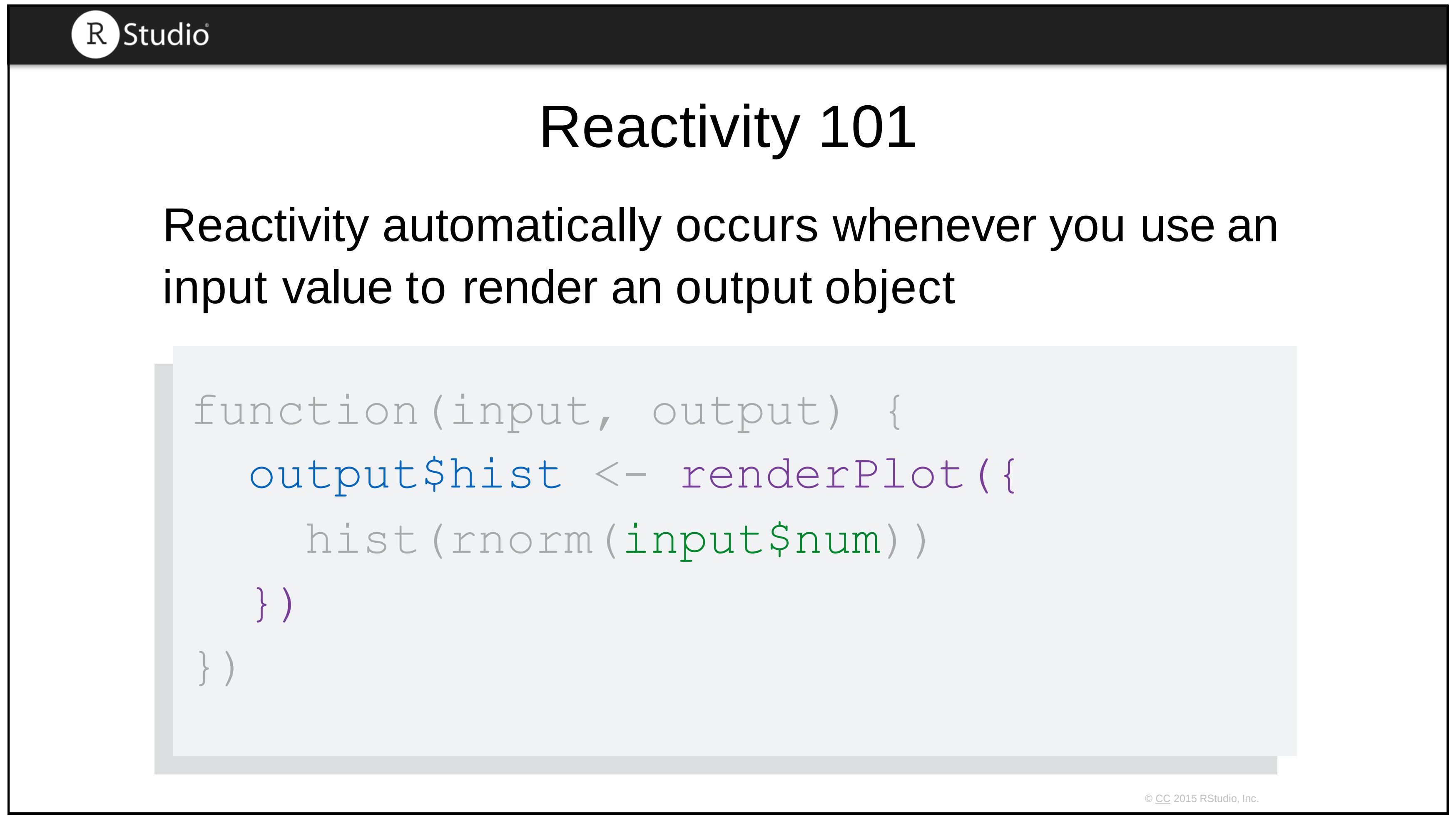
```
input$num = 25
```

```
input$num = 50
```

```
input$num = 75
```

Output will automatically update if you follow the 3 rules

44

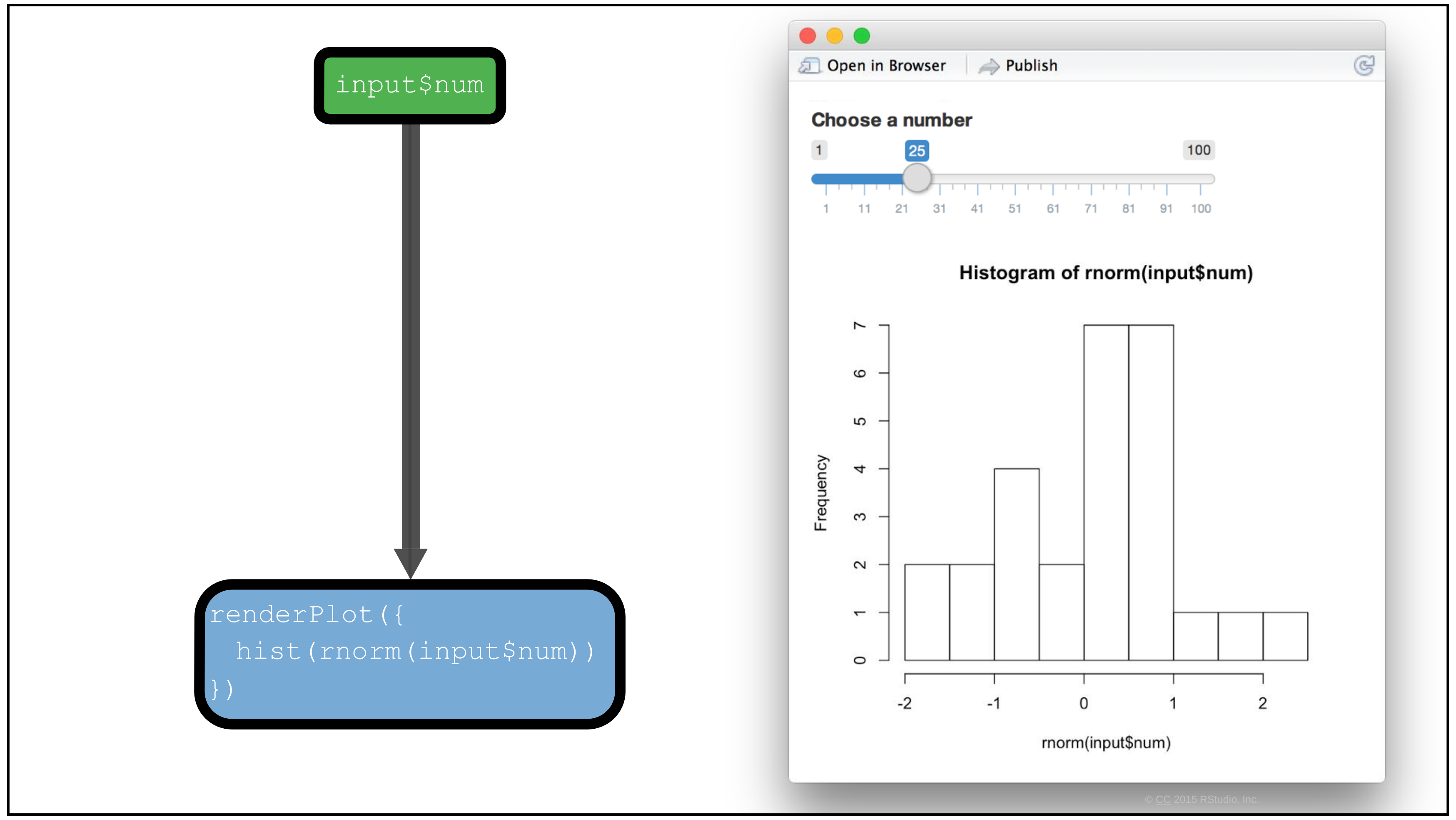


Reactivity 101

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

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

45

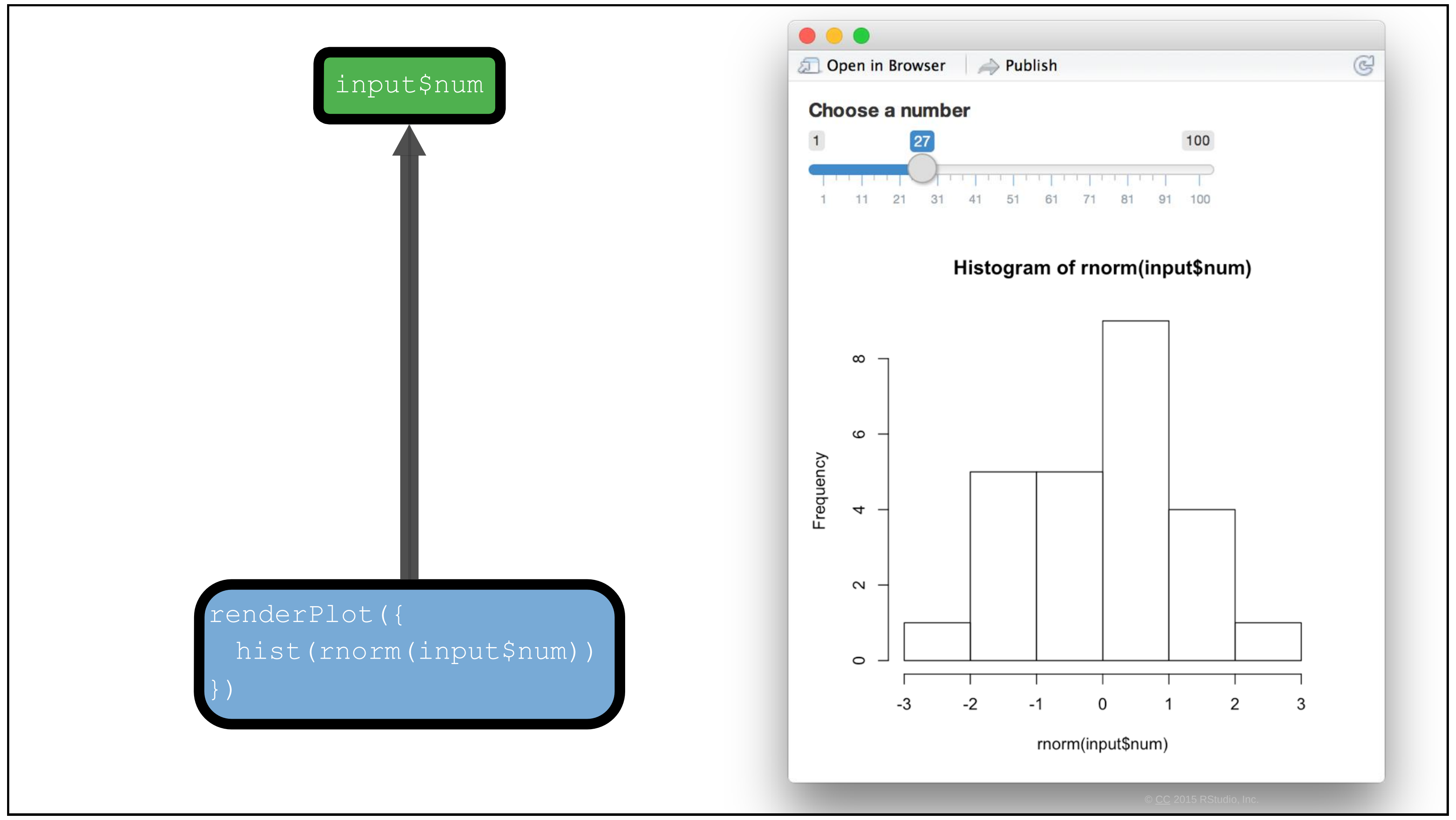


```
input$num
```

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

Choose a number: 25

46

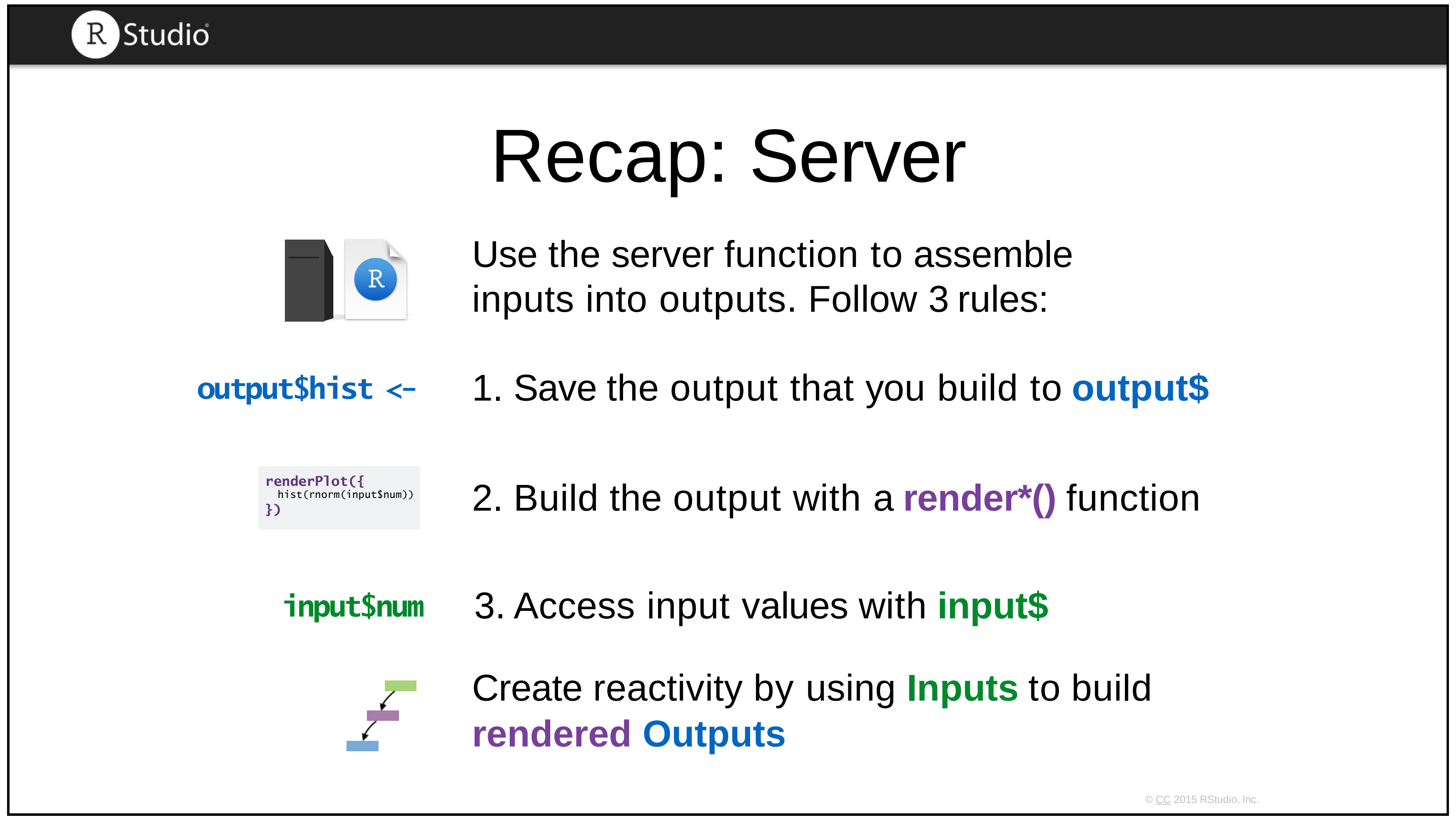


```
input$num
```

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

Choose a number: 27

47



RStudio

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

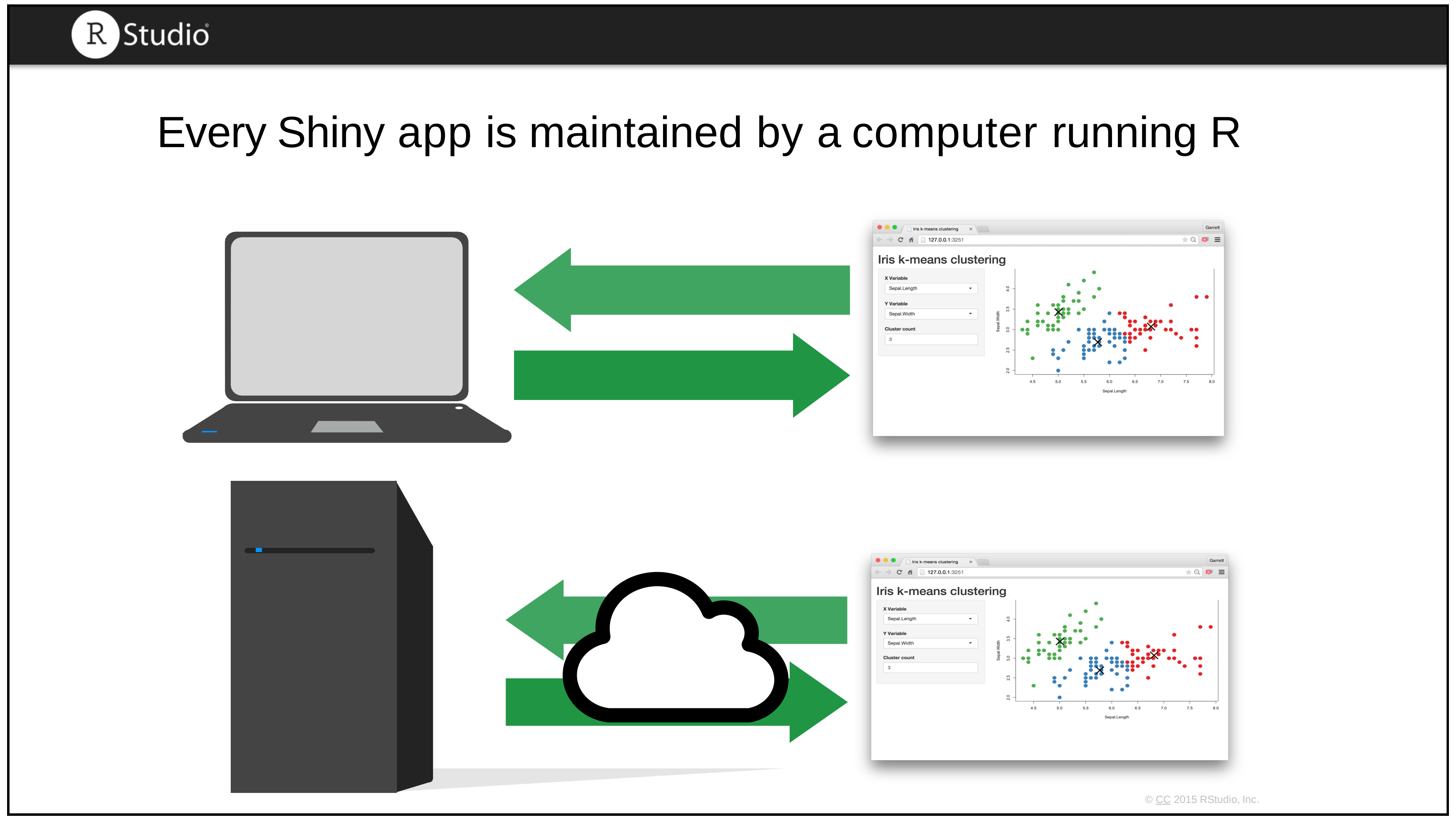
`input$num` 3. Access input values with **input$**

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

48

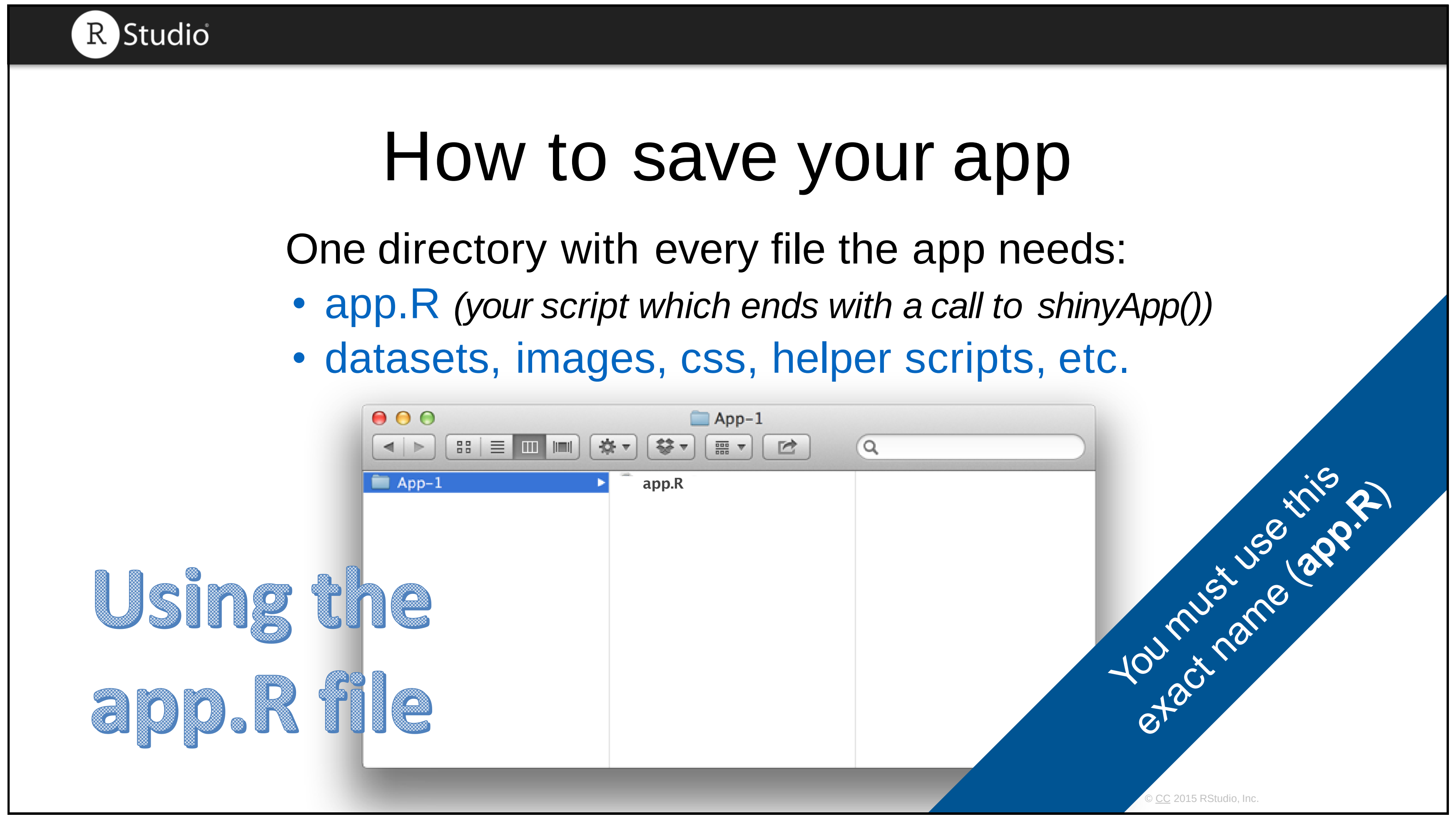
Share your app !

49



Every Shiny app is maintained by a computer running R

50



How to save your app

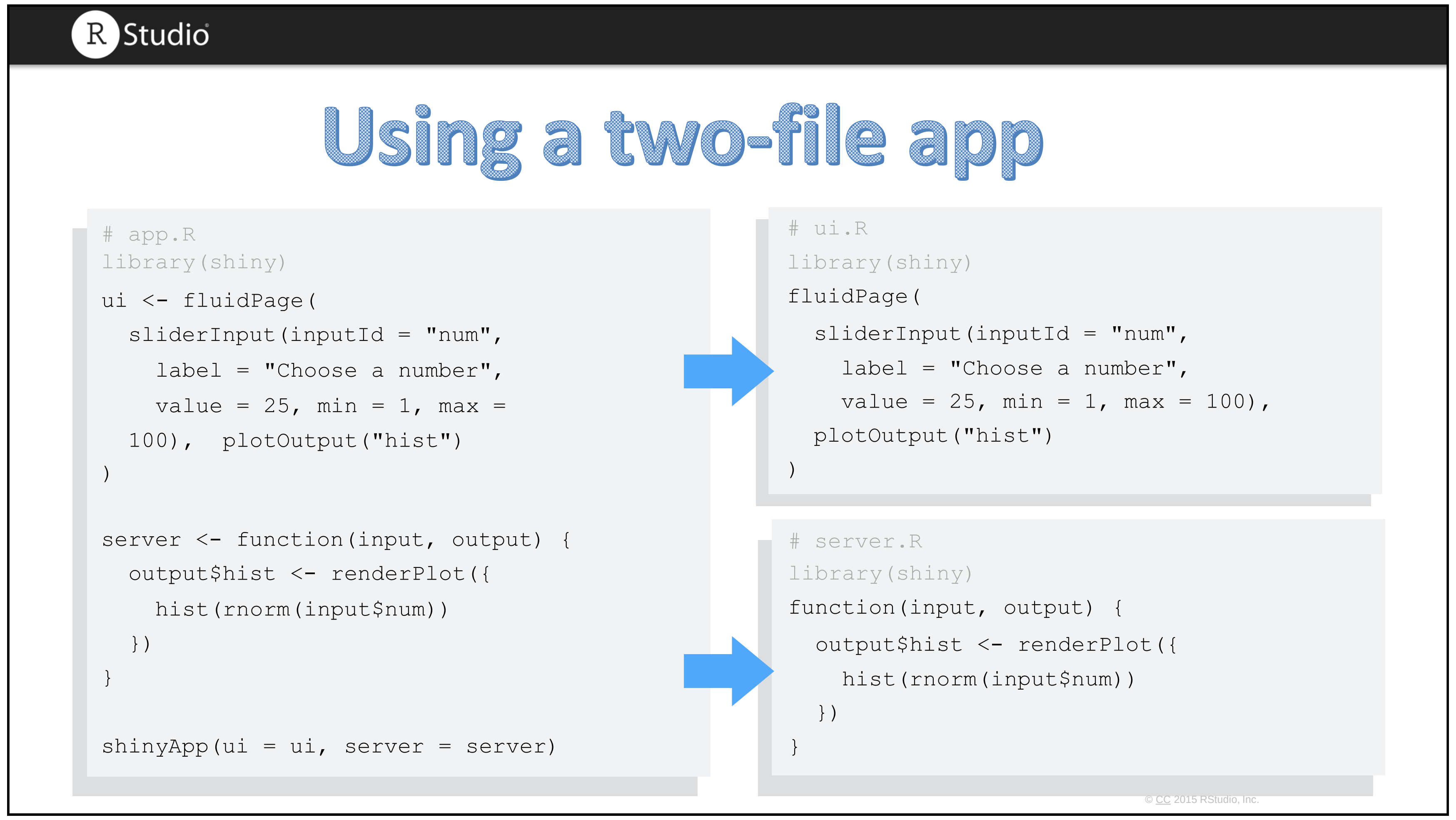
One directory with every file the app needs:

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

Using the app.R file

You must use this exact name (**app.R**)

51



RStudio

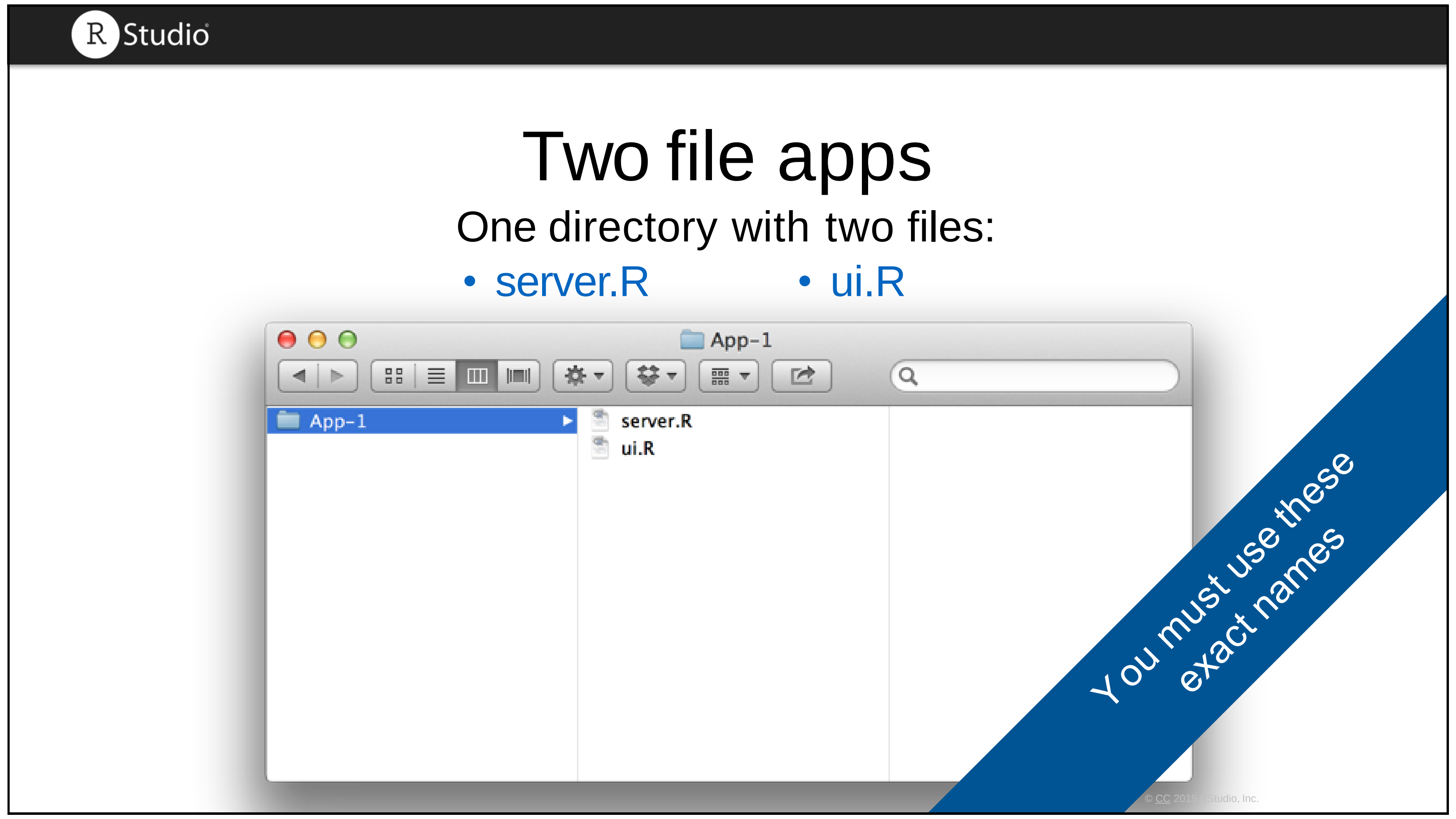
Using a two-file app

```
# app.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)
```

```
# 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))
  })
}
```

52



RStudio

Two file apps

One directory with two files:

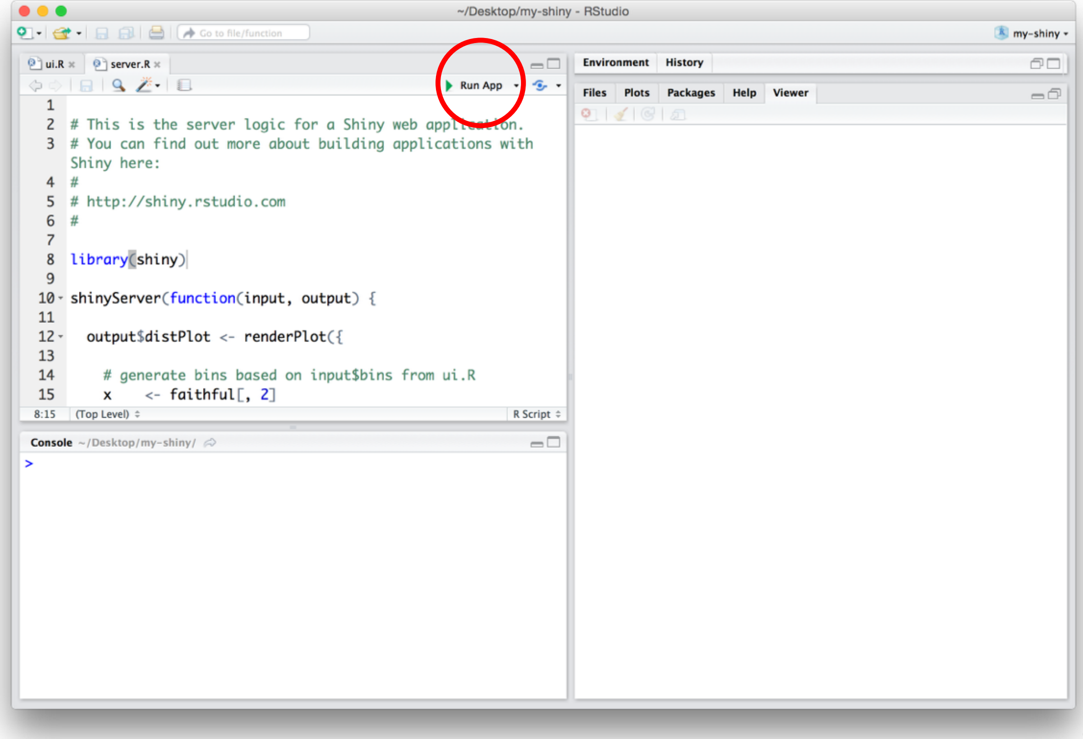
- server.R
- ui.R

You must use these exact names

53

R Studio

Launch an app



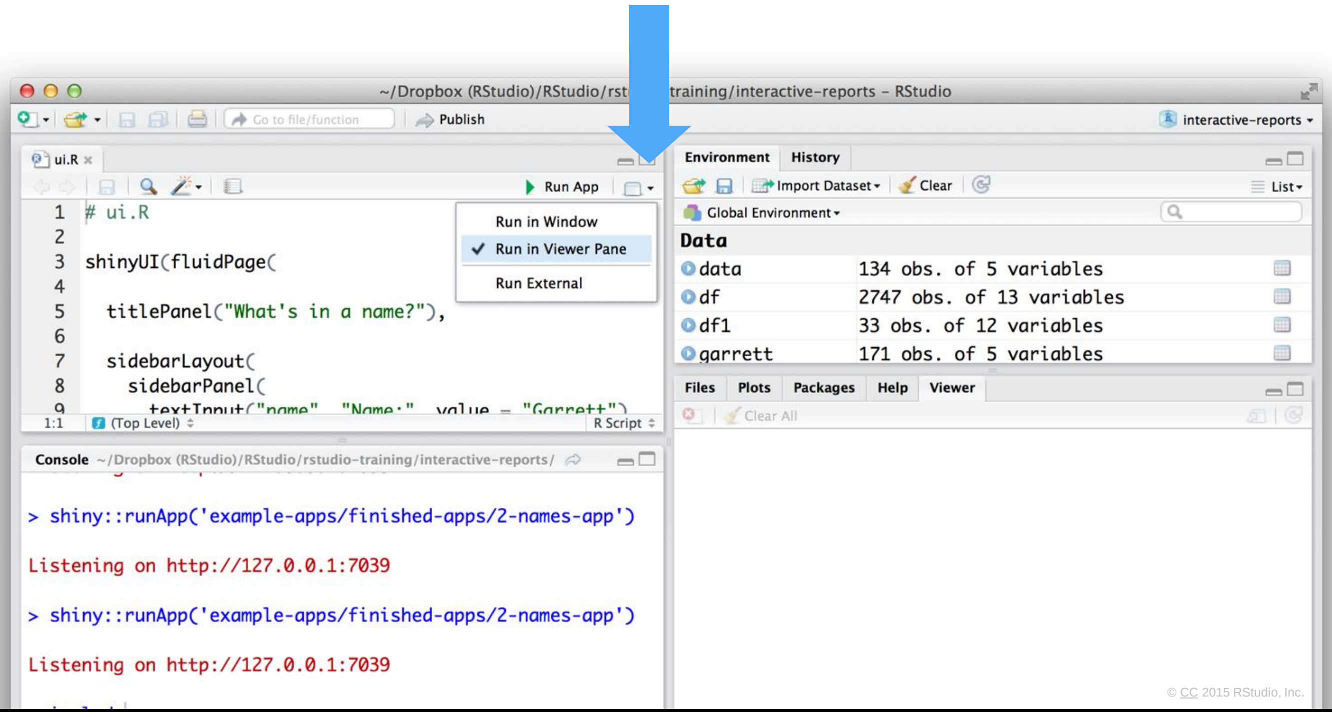
```r

# This is the server logic for a Shiny web application.
# You can find out more about building applications with Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
```

```
>
```

54

R Studio

Display options



```r
# ui.R

shinyUI(fluidPage(

  titlePanel("What's in a name?"),

  sidebarLayout(
    sidebarPanel(
      textInput("name", "Name:", value = "Garrett"),
```

```
Data
data        134 obs. of 5 variables
df          2747 obs. of 13 variables
df1         33 obs. of 12 variables
garrett     171 obs. of 5 variables
```

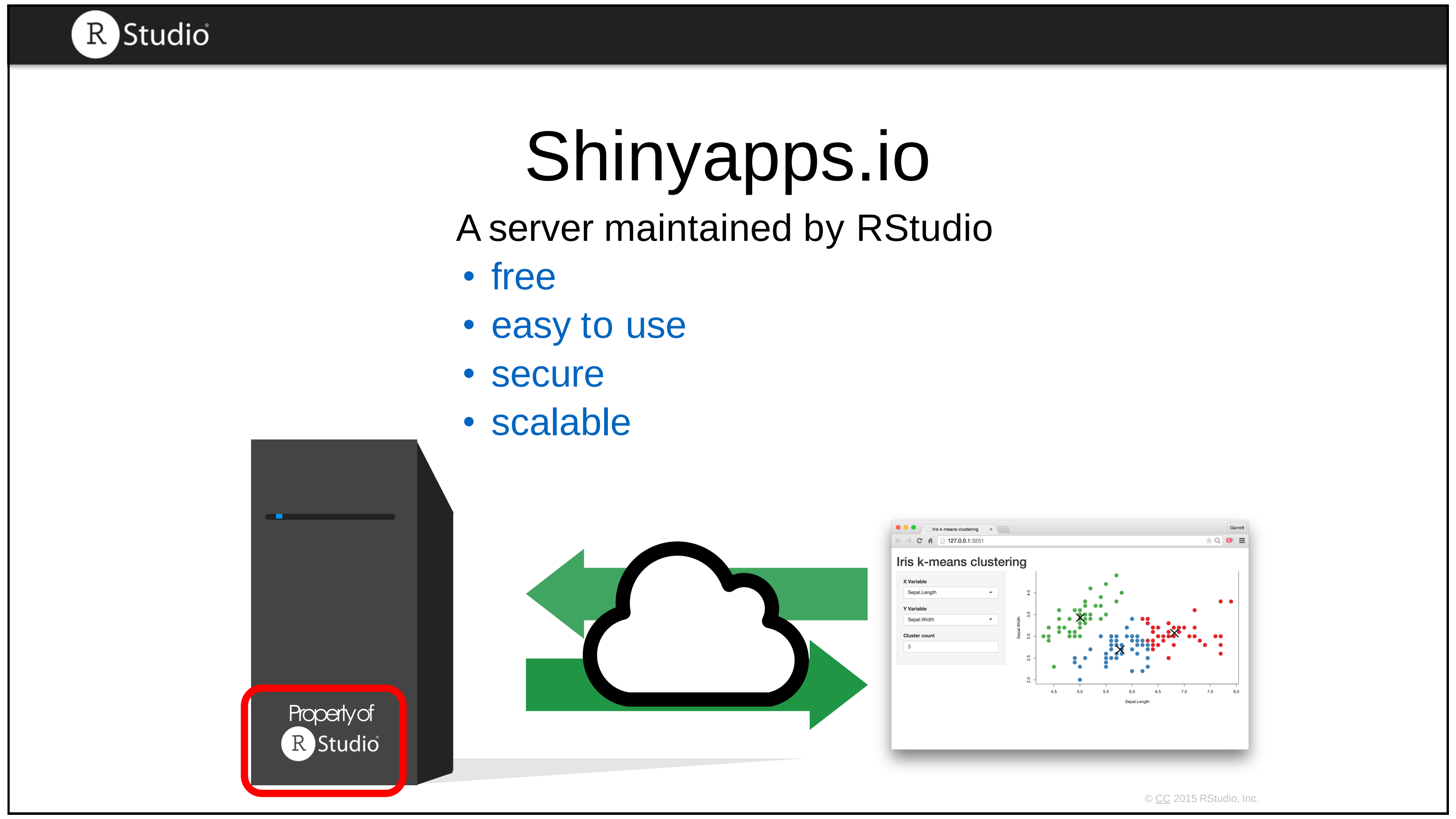
```
> shiny::runApp('example-apps/finished-apps/2-names-app')

Listening on http://127.0.0.1:7039

> shiny::runApp('example-apps/finished-apps/2-names-app')

Listening on http://127.0.0.1:7039
```

55

Use shinyapps.io

!

56



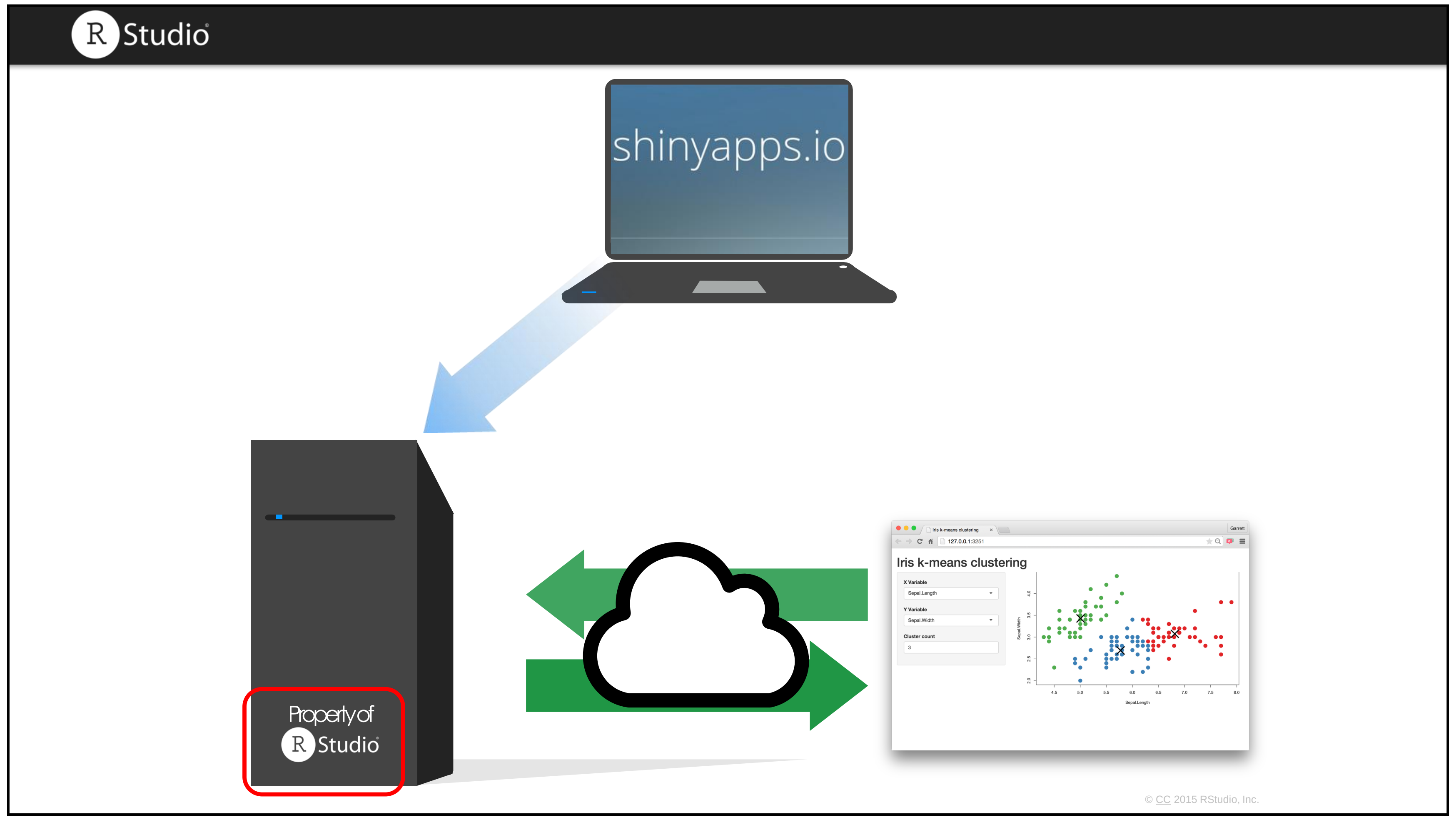
RStudio

Shinyapps.io

A server maintained by RStudio

- free
- easy to use
- secure
- scalable

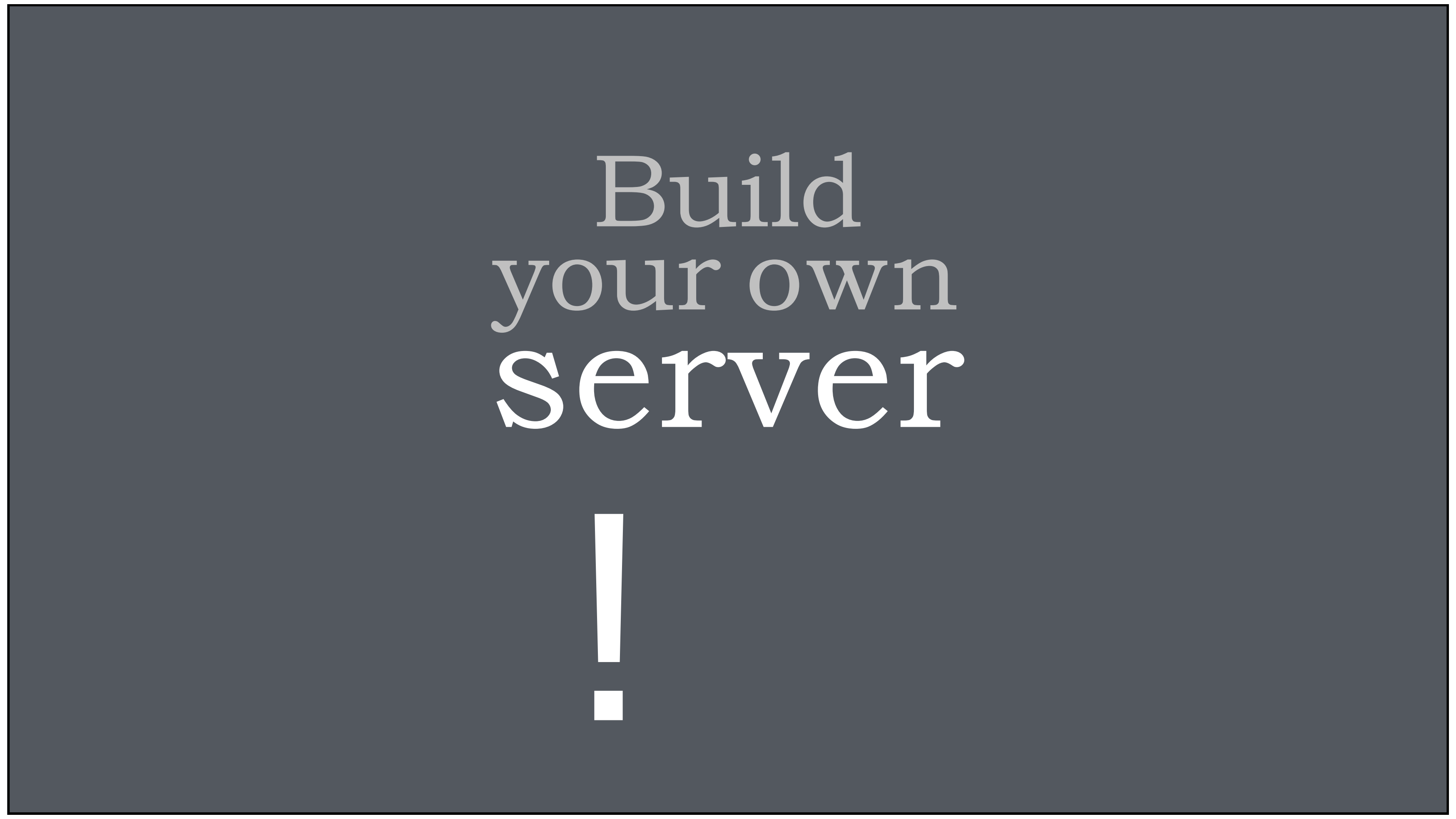
57



Iris k-means clustering

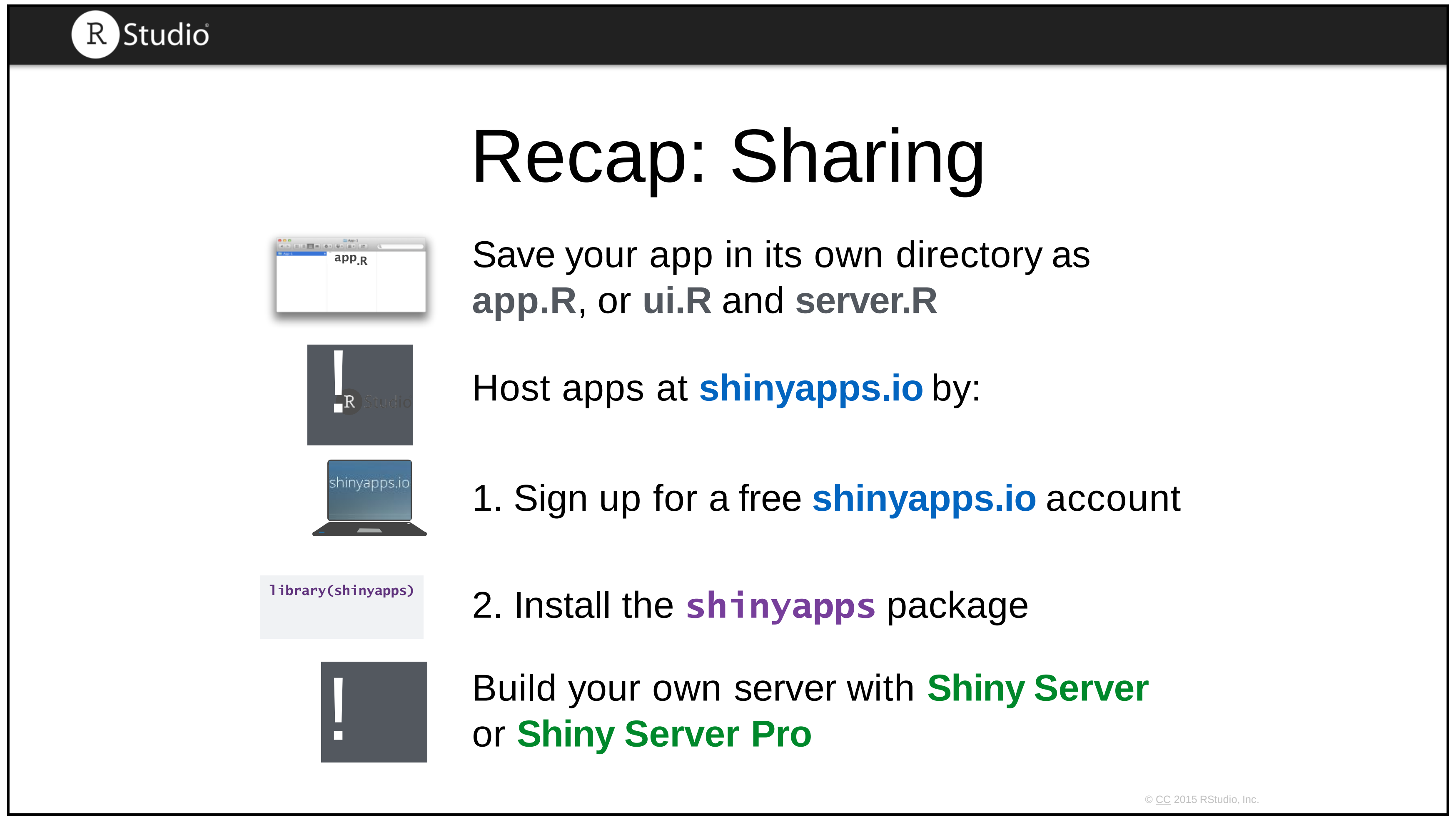
© CC 2015 RStudio, Inc.

58



Build your own server!

60


R Studio
app.R
shinyapps.io

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

```
library(shinyapps)
```

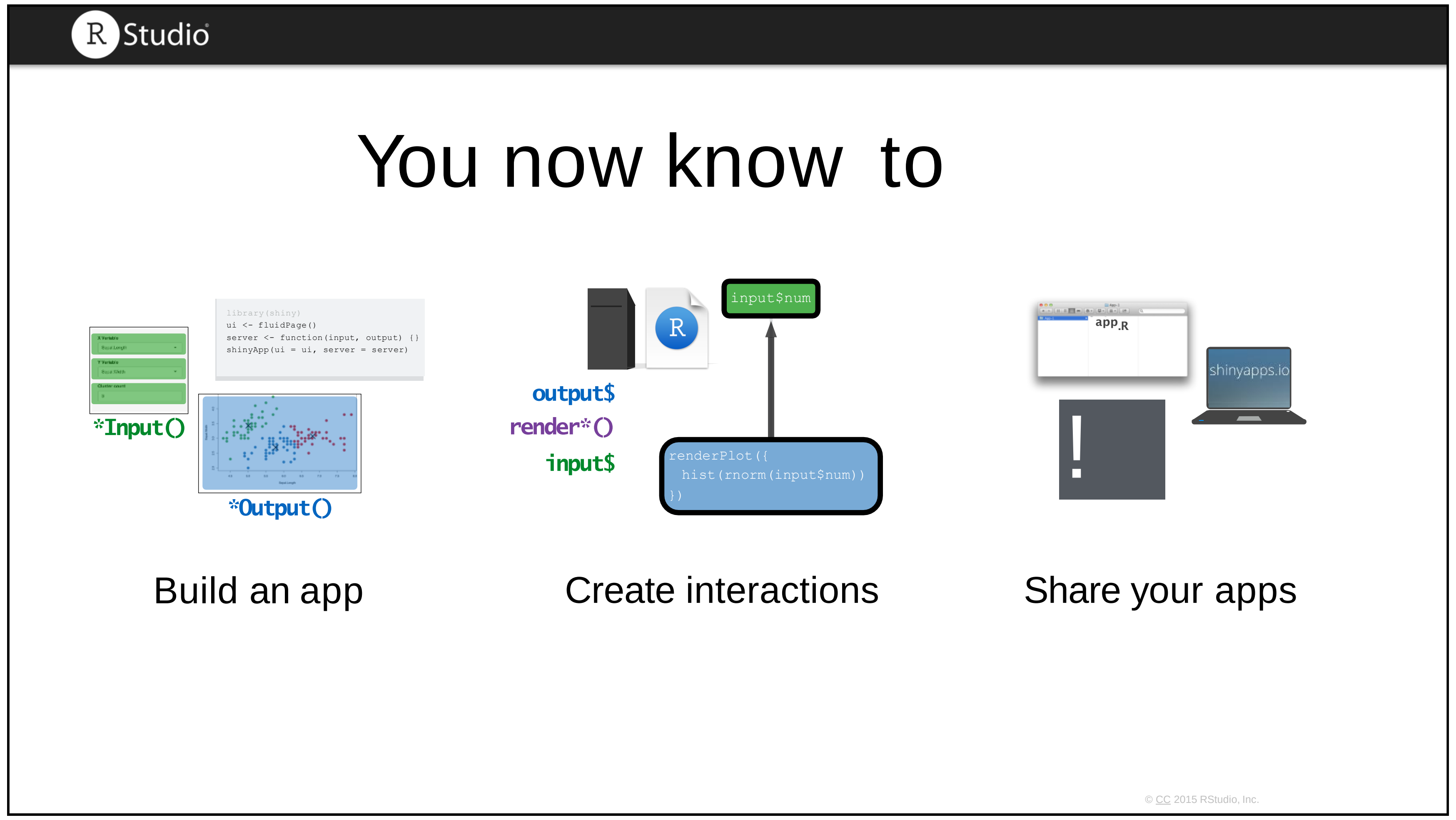
2. Install the **`shinyapps`** package

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

63

Recap all

64



You now know to

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

Build an app

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

Create interactions

Share your apps

65