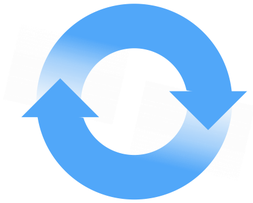
RStudio

How to start with Shiny, Part 2

How to customize reactions



Based on the slides by Garrett Grolemund

1

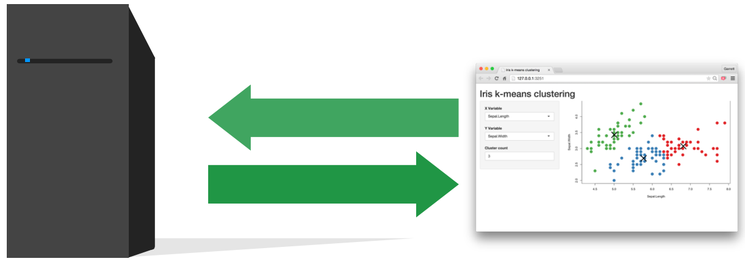
The story so far

2

RStudio

Slides at: bit.ly/shiny-quickstart-2

Every Shiny app is ***maintained*** by a computer running R



3

RStudio

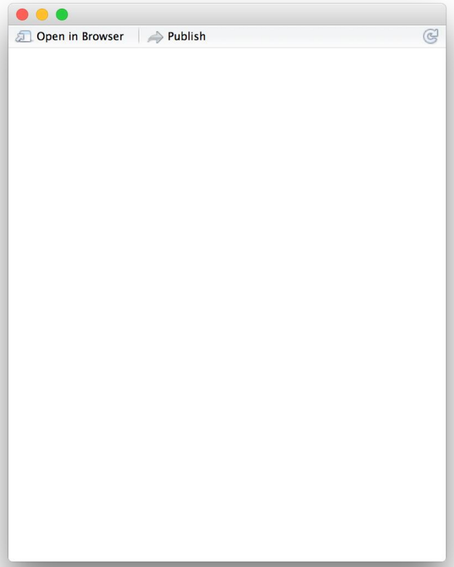
App template

The shortest viable shiny app


```
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

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

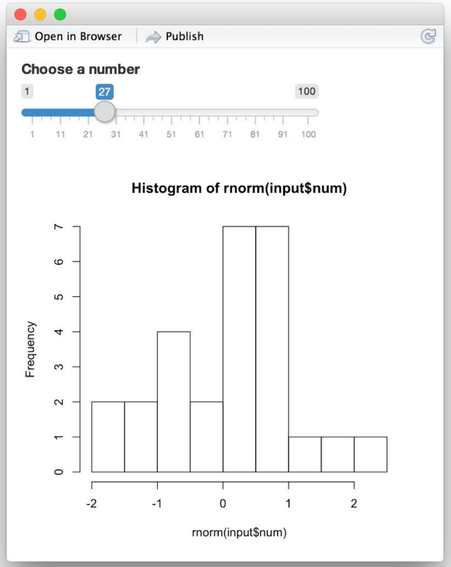

4

```r
library(shiny)

ui <- fluidPage(



)

server <- function(input, output) {



}

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



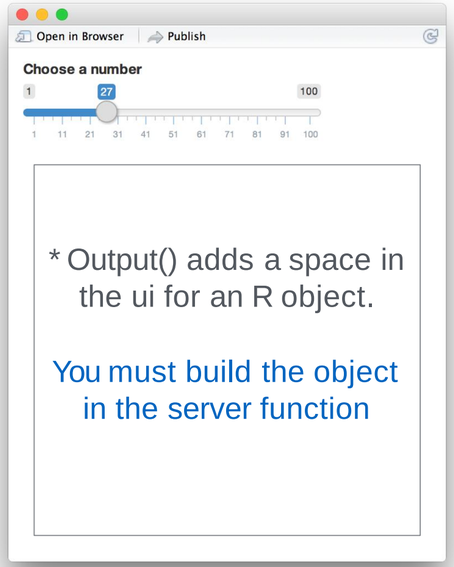
5

```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)
```



6

```
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)
```



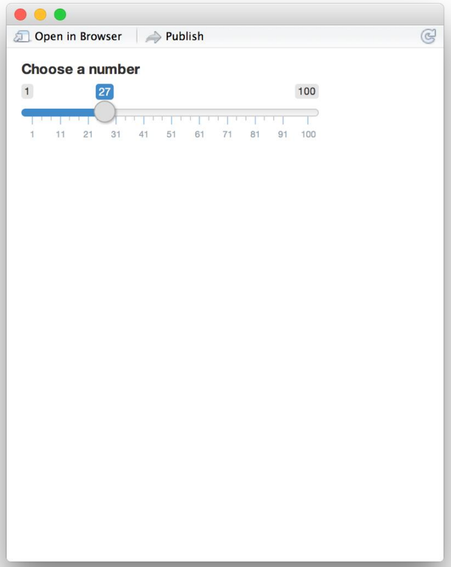
* Output() adds a space in the ui for an R object.

You must build the object in the server function

7

```
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)
```

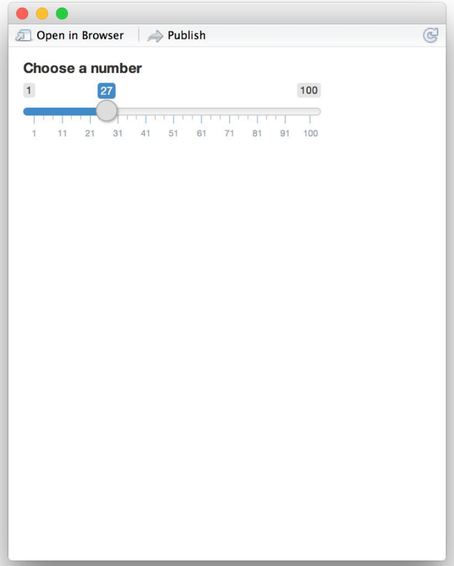
1



8

```
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)
```

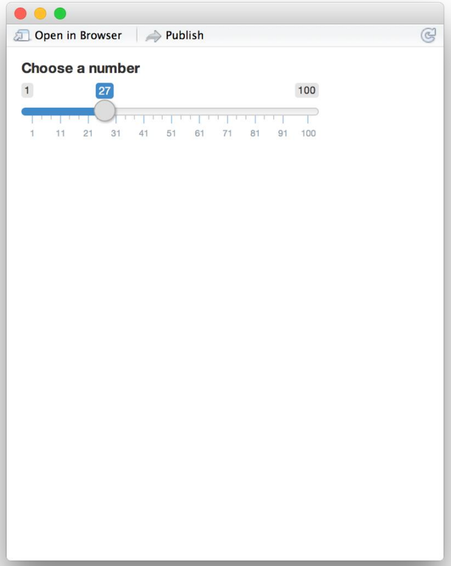
2



9

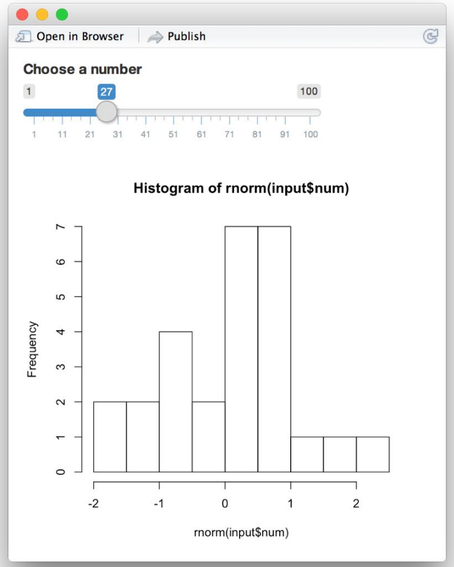
```
library(shiny)

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



10

```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)
```



11

What is Reactivity?

12

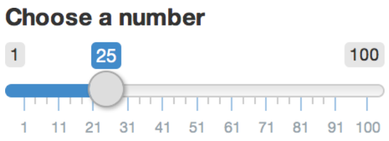
Begin with

reactive values

23

RStudio

Syntax



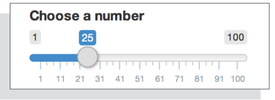

```
sliderInput(inputId = "num", label = "Choose a number", …)
```

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

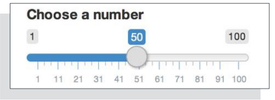
24

Input values

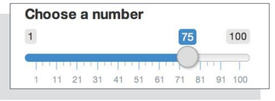
The input value changes whenever a user changes the input.




```
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 a reactive funciton.

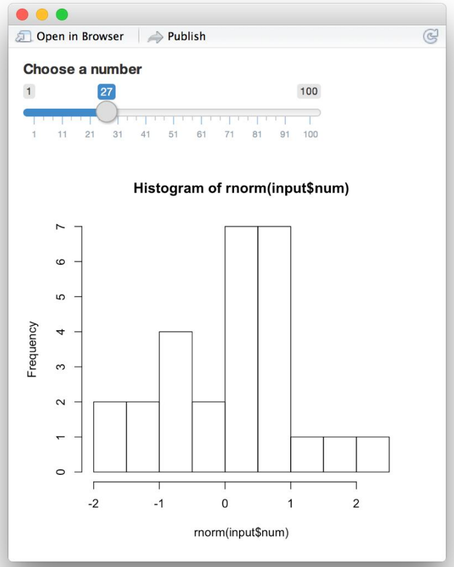
√

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

⚠

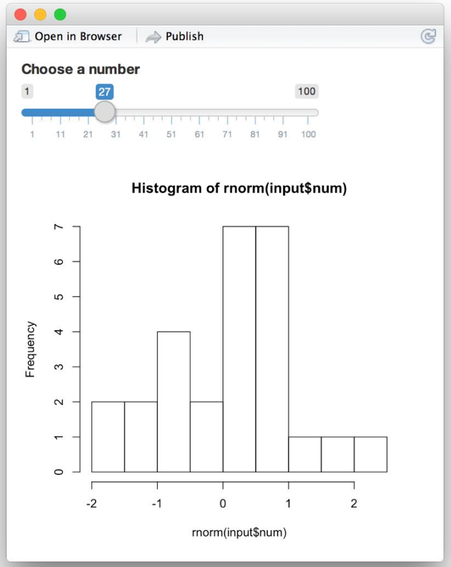
```
hist(rnorm(100, input$num))
```

```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)
```



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

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



28

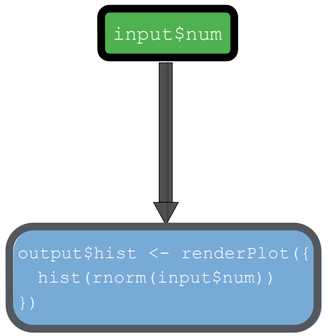
```r
library(shiny)

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

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

}

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

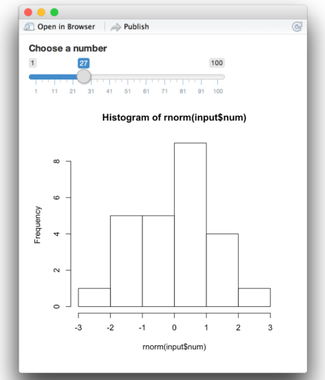
```
Error in .getReactiveEnvironment()
$currentContext() :
  Operation not allowed without an
active reactive context. (You tried
to do something that can only be done
from inside a reactive expression or
observer.)
```


29

Think of reactivity in R as a two step process

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



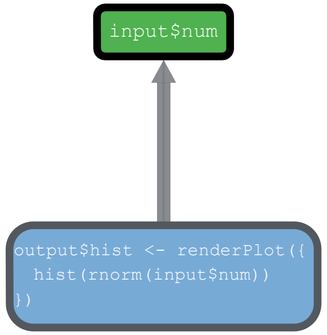


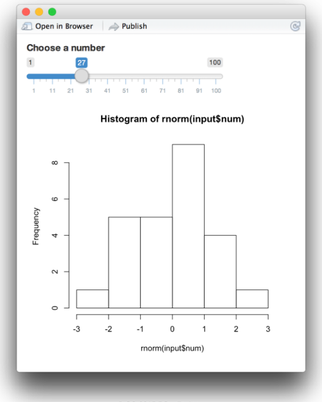
30

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)





31

RStudio

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.**

32

Reactive toolkit

(7 indispensable render functions)

33

RStudio

Reactive functions

1 Use a code chunk to build an object/widget
- **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?

34

Display output with render*()

35

RStudio

Slides at: bit.ly/shiny-quickstart-2

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

36

RStudio

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

37

RStudio

render*()

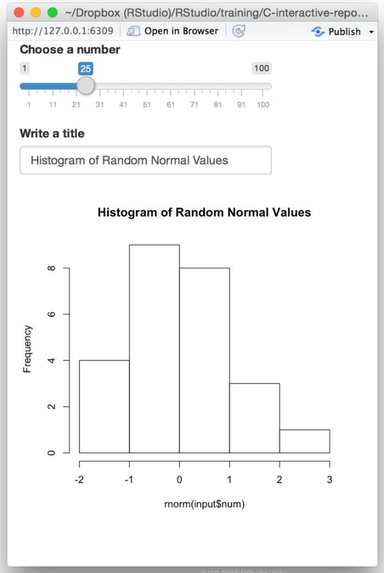
Builds reactive output to display in UI

```
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

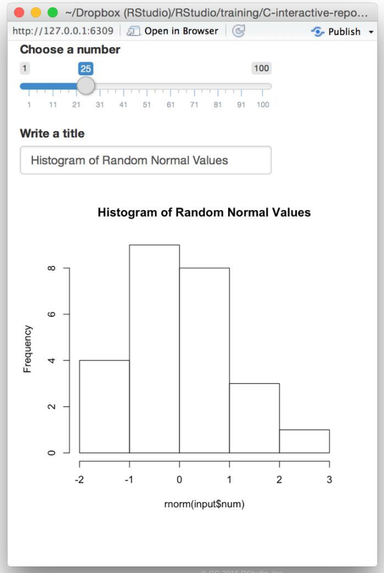
38

```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)
```

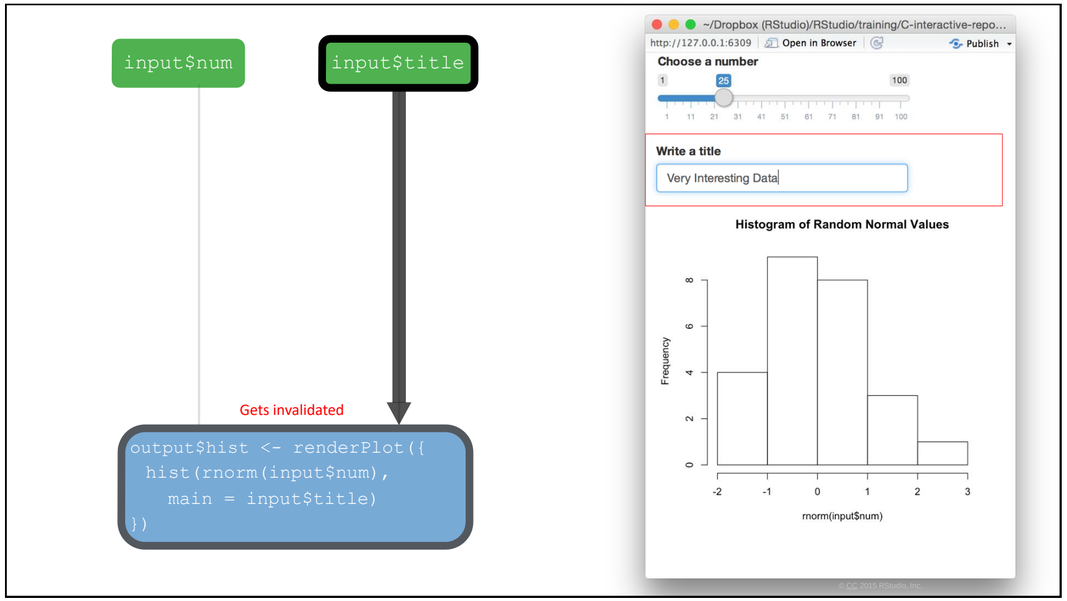


39

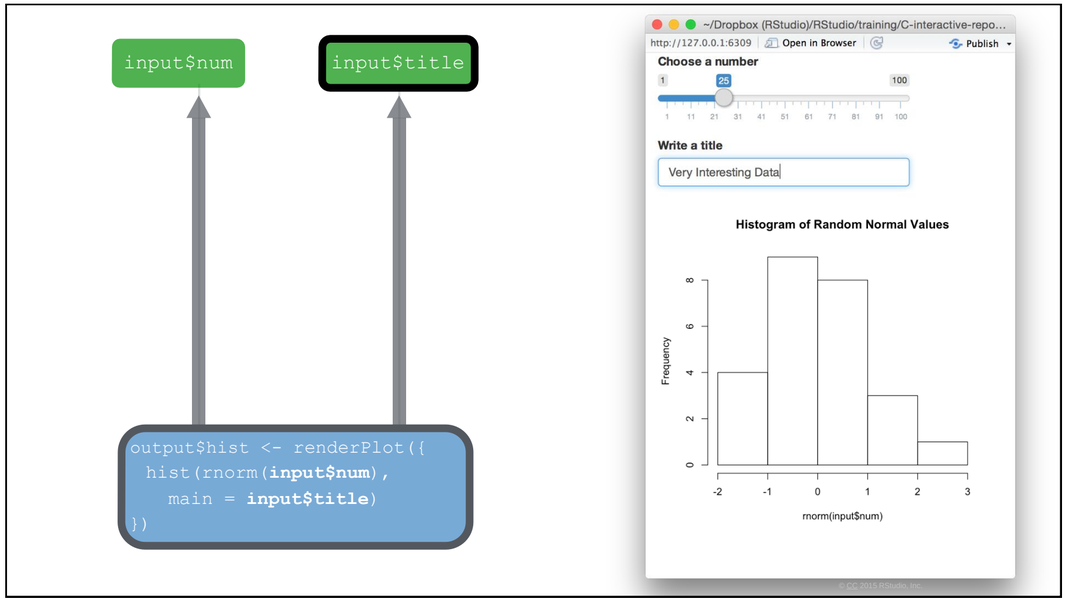

```r
# 01-two-inputs

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)
```

40


input$num
input$title
Gets invalidated
output$hist <- renderPlot({
hist(rnorm(input$num),
main = input$title)
})
~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
1
25
100
Write a title
Very Interesting Data
Histogram of Random Normal Values
Frequency
rnorm(input$num)

41


input$num
input$title
output$hist <- renderPlot({
hist(rnorm(input$num),
main = input$title)
})
~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
1
25
100
Write a title
Very Interesting Data
Histogram of Random Normal Values
Frequency
rnorm(input$num)

42

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)
})
```

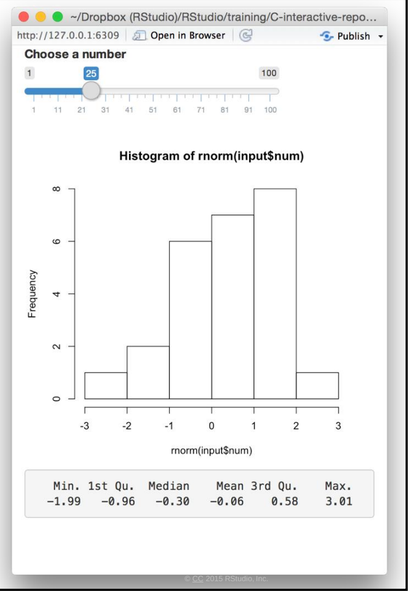
render*() makes an observer an object that has a **block of code** associated with it

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

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

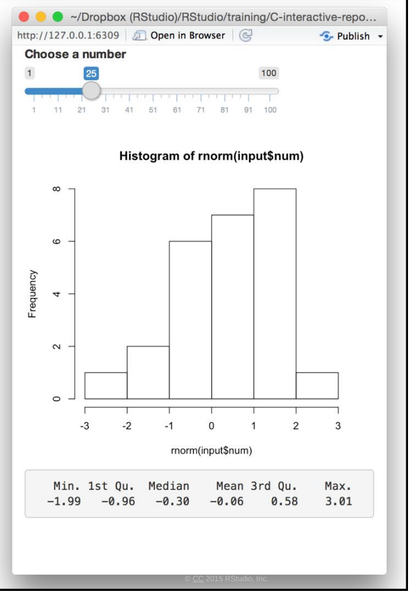
Modularize code with reactive()

```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)
```

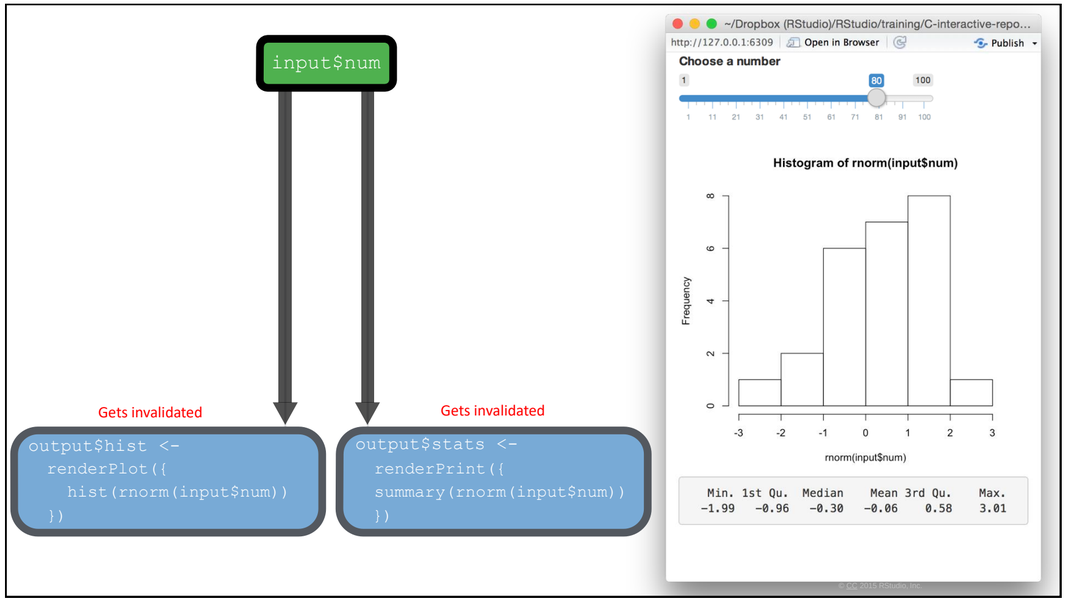


45

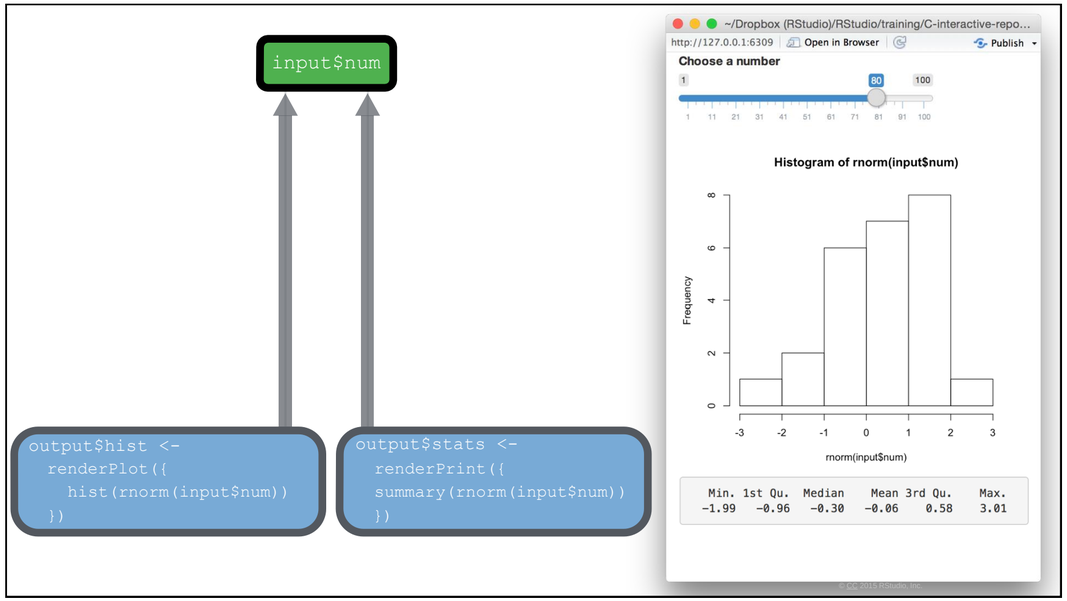

```r
# 02-two-outputs

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)
```

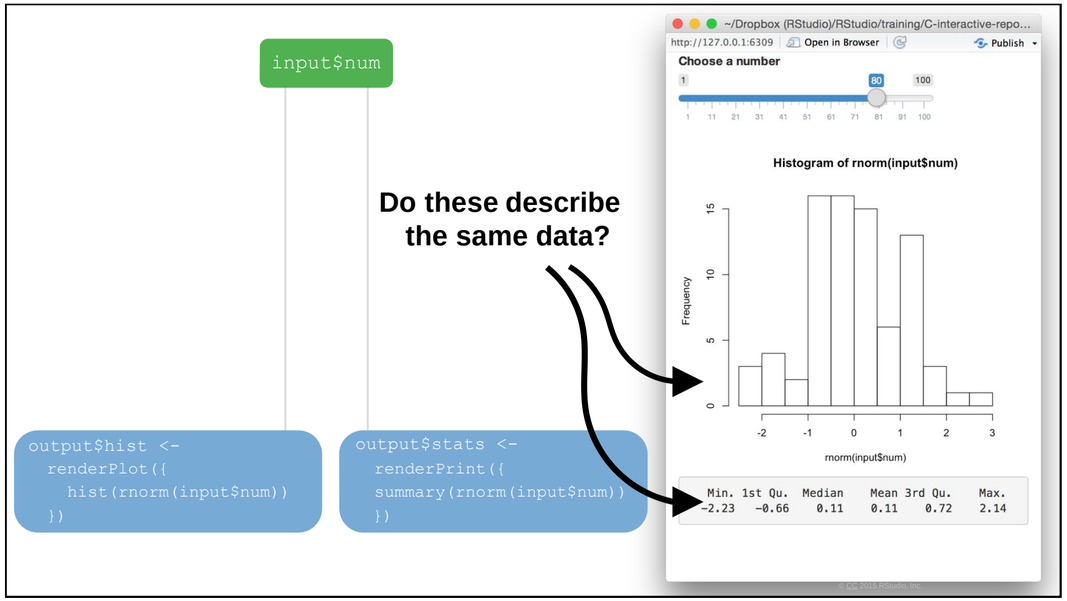
46


input$num
Gets invalidated
Gets invalidated
output$hist <-
renderPlot({
hist(rnorm(input$num))
})
output$stats <-
renderPrint({
summary(rnorm(input$num))
})
~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
1
80
100
Histogram of rnorm(input$num)
Frequency
rnorm(input$num)
Min. 1st Qu. Median Mean 3rd Qu. Max.
-1.99 -0.96 -0.30 -0.06 0.58 3.01

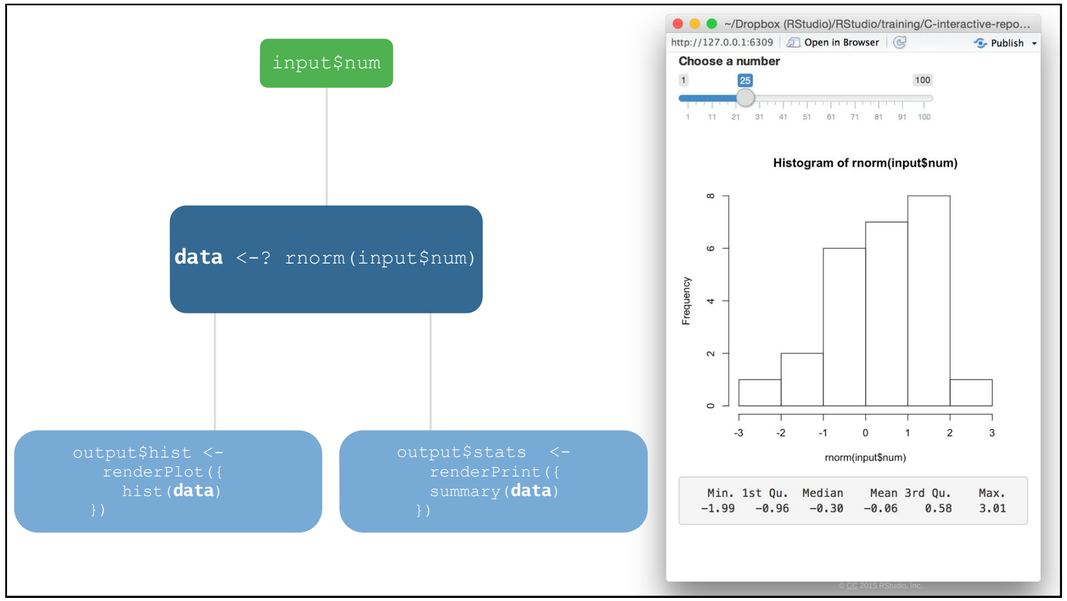
47


input$num
output$hist <-
renderPlot({
hist(rnorm(input$num))
})
output$stats <-
renderPrint({
summary(rnorm(input$num))
})
~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
1
80
100
Histogram of rnorm(input$num)
Frequency
rnorm(input$num)
Min. 1st Qu. Median Mean 3rd Qu. Max.
-1.99 -0.96 -0.30 -0.06 0.58 3.01

48


input$num
Do these describe the same data?
output$hist <-
renderPlot({
hist(rnorm(input$num))
})
output$stats <-
renderPrint({
summary(rnorm(input$num))
})
~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
Histogram of rnorm(input$num)
Frequency
rnorm(input$num)
Min. 1st Qu. Median Mean 3rd Qu. Max.
-2.23 -0.66 0.11 0.11 0.72 2.14

49


input$num
data <-? rnorm(input$num)
output$hist <-
renderPlot({
hist(data)
})
output$stats <-
renderPrint({
summary(data)
})
~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
Histogram of rnorm(input$num)
Frequency
rnorm(input$num)
Min. 1st Qu. Median Mean 3rd Qu. Max.
-1.99 -0.96 -0.30 -0.06 0.58 3.01

50

RStudio

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

51

RStudio

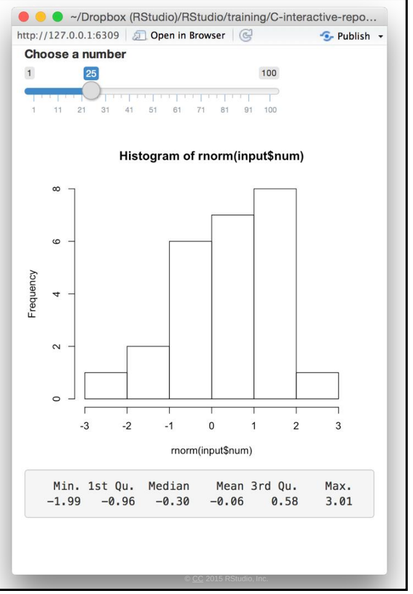
A reactive expression is special in two ways

```
data()
```

1 You call a reactive expression like a function

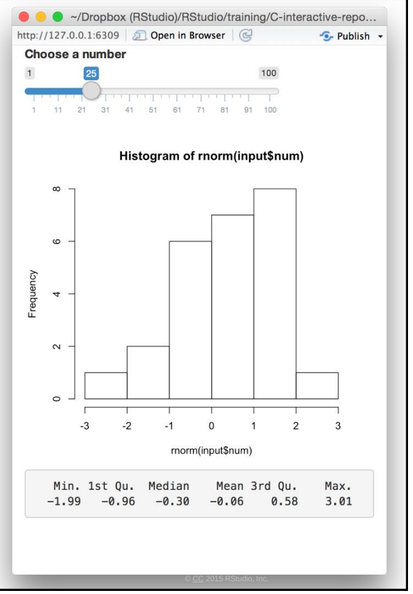
52

```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)
```



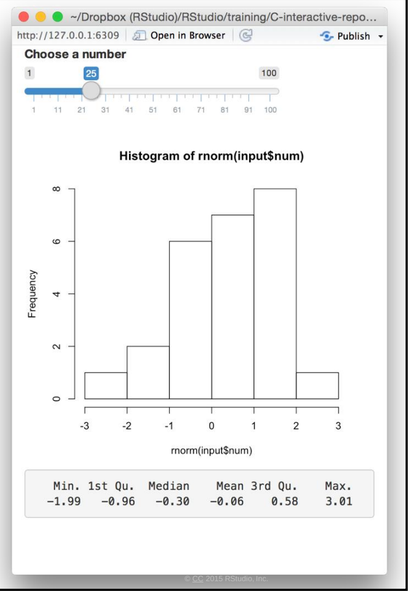
53

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

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

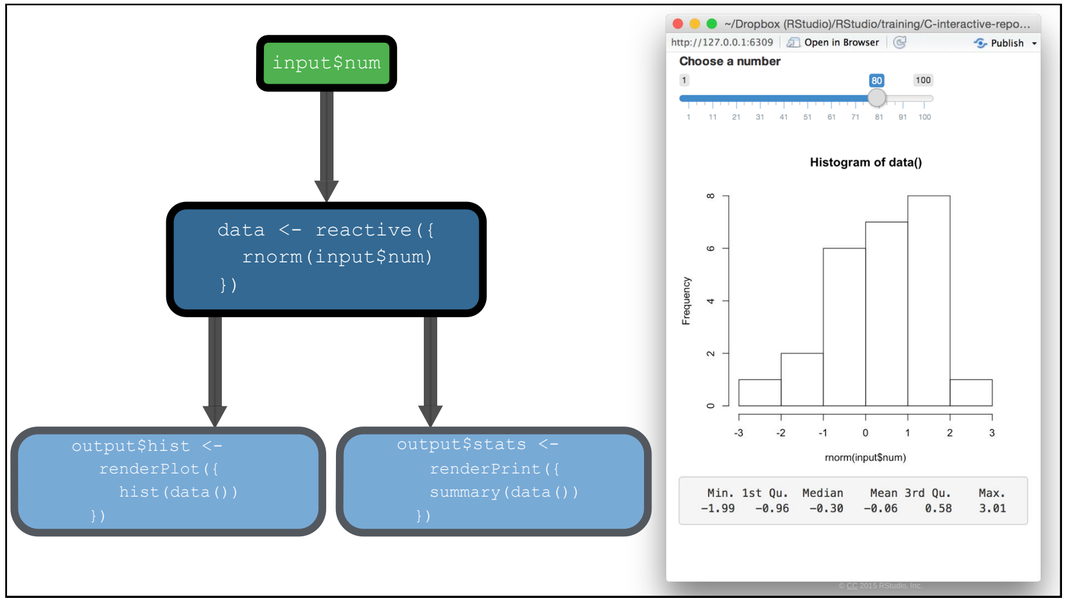


54

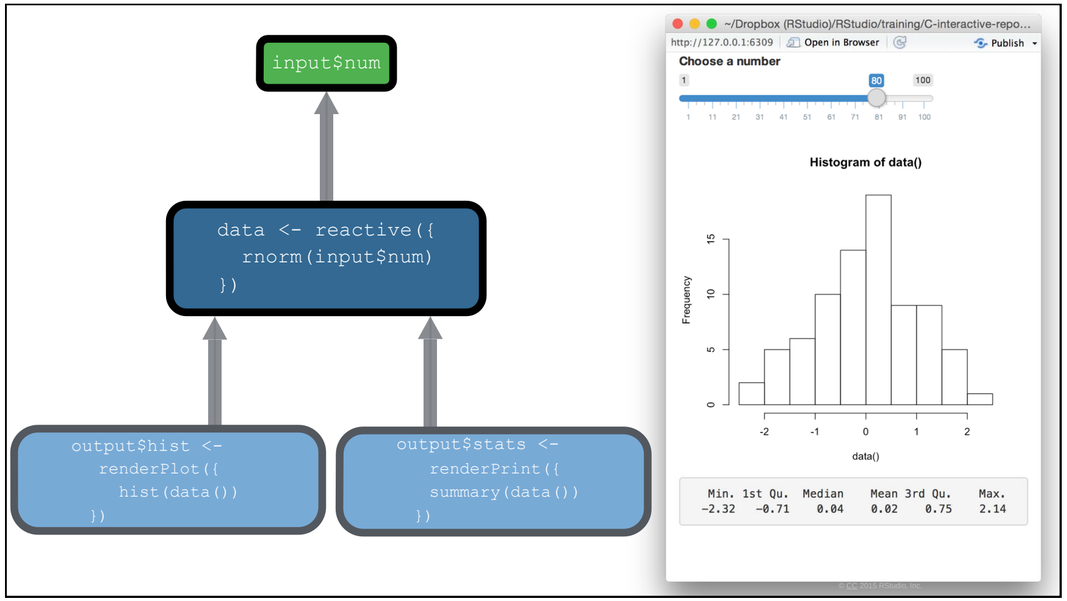
```r
# 03-reactive

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)
```



55



56



57

RStudio

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)

58

RStudio

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

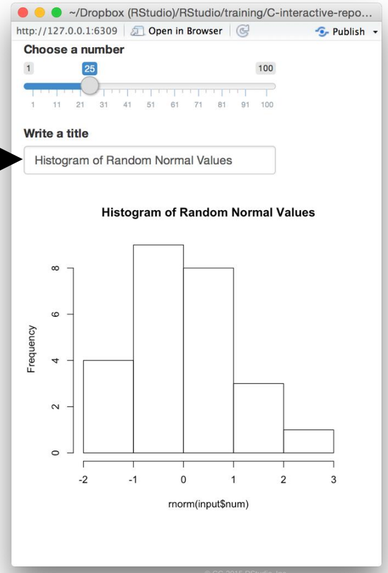
59

Prevent reactions with isolate()

60

```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?



61

RStudio

isolate()

Returns the result as a non-reactive value

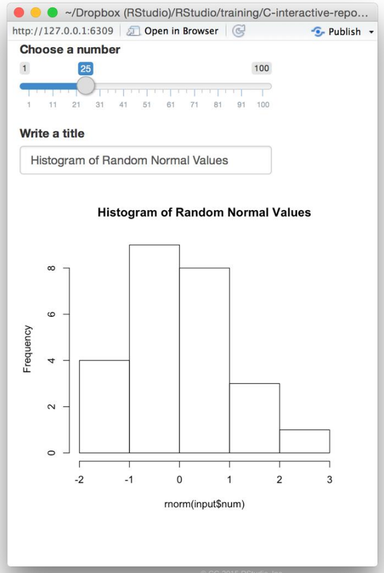
```r
isolate({ rnorm(input$num) })
```

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

code used to build object

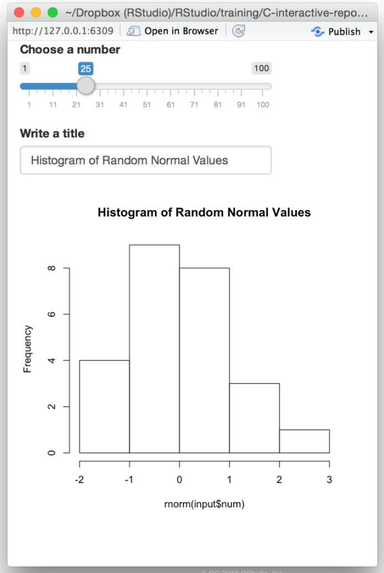
62

```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)
```

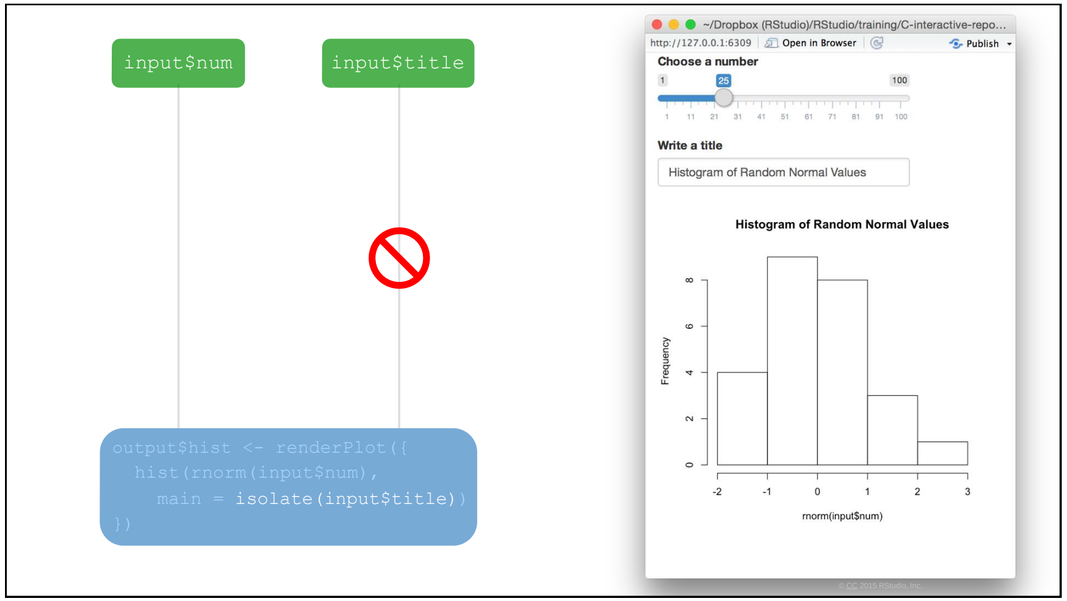


63

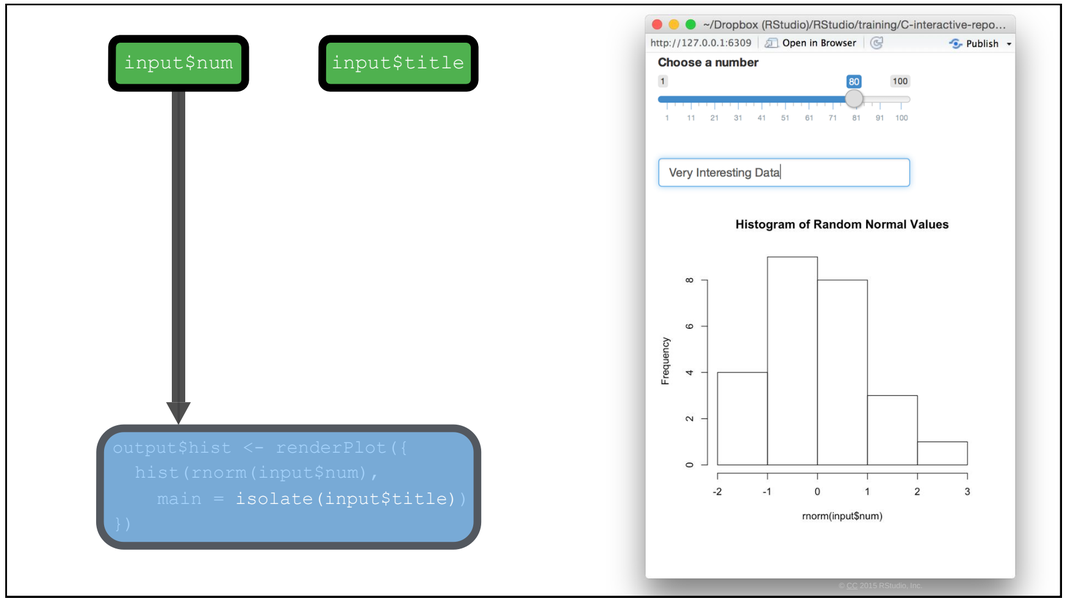
```r
# 04-isolate

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)
```



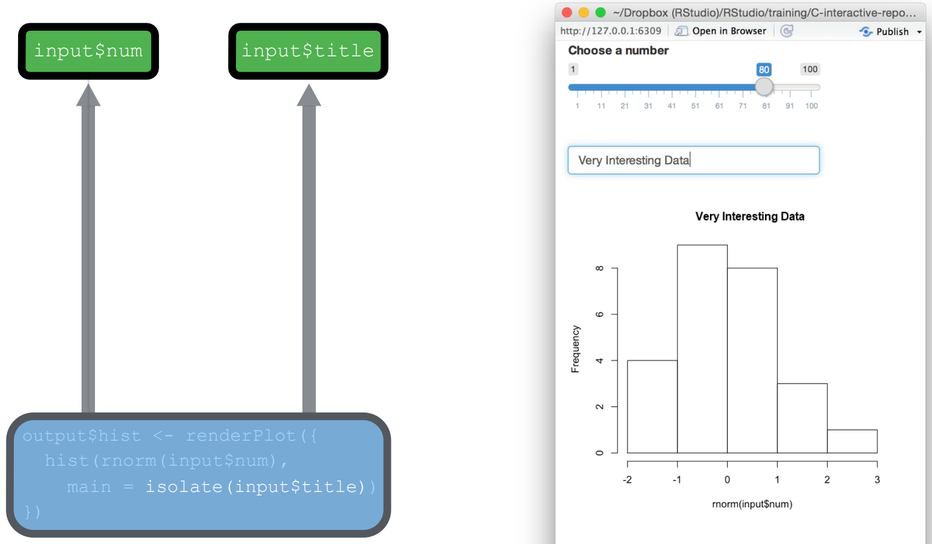
64


input$num
input$title
output$hist <- renderPlot({
hist(rnorm(input$num),
main = isolate(input$title))
})
~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
1
25
100
Write a title
Histogram of Random Normal Values
Histogram of Random Normal Values
Frequency
rnorm(input$num)

65


input$num
input$title
output$hist <- renderPlot({
hist(rnorm(input$num),
main = isolate(input$title))
})
~/Dropbox (RStudio)/RStudio/training/C-interactive-repo...
http://127.0.0.1:6309
Open in Browser
Publish
Choose a number
1
80
100
Very Interesting Data
Histogram of Random Normal Values
Frequency
rnorm(input$num)

66



67

RStudio

Recap: isolate()

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

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

68

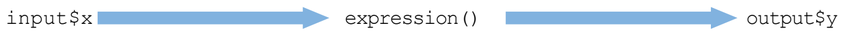
Trigger code

with observeEvent()

69

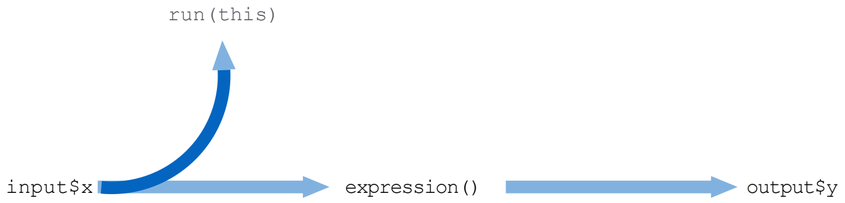
`input$x` → `output$y`

70



71

Slides at: bit.ly/shiny-quickstart-2



72

RStudio

Action buttons

An Action Button

Click Me!

input function

input name (for internal use)

label to display

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


73

```
# 05-actionButton

library(shiny)

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

server <- function(input, output) {



}

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

74

RStudio

observeEvent()

Triggers code to run on server

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

reactive value(s) to respond to

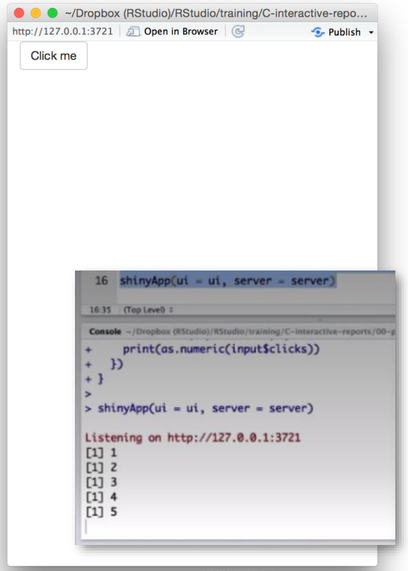
(observer invalidates ONLY when this value changes)

code block to run whenever observer is invalidated

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

75

```
# 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)
```



76

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

77

Recap: observeEvent()

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

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

observe()

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

78