Exercise IV – R Shiny
Part 2 – Shiny App Control

CEE412 / CET522

TRANSPORTATION DATA MANAGEMENT AND VISUALIZATION

WINTER 2020

Shiny App Control

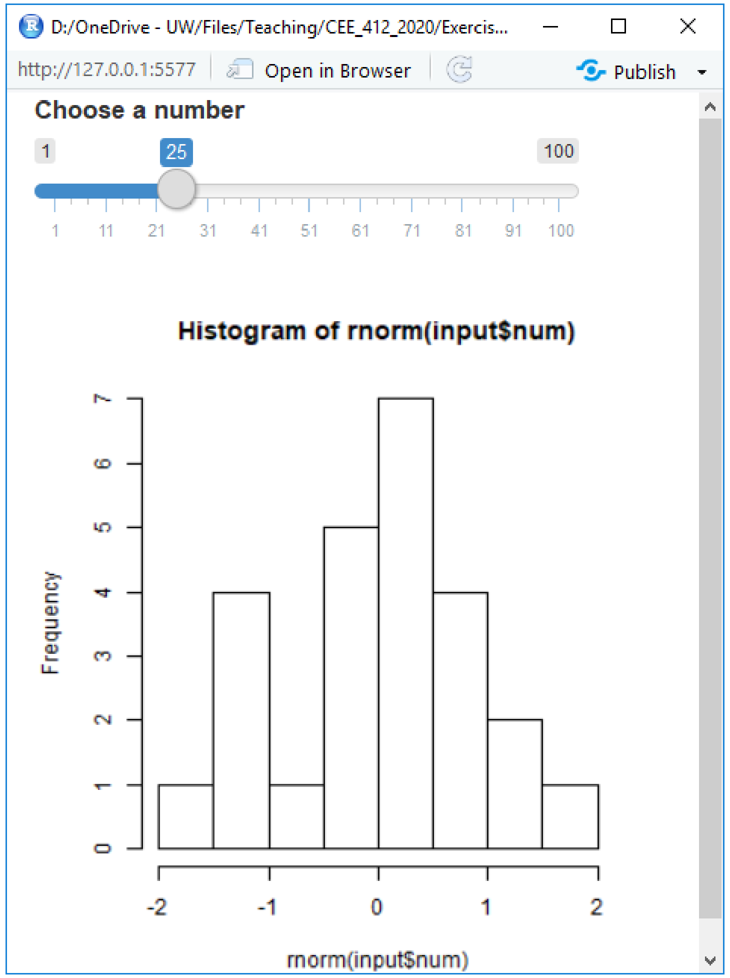
- In this section, we will introduce how to control and customize your
  - Inputs of the UI widgets
  - Outputs of the UI widgets

- Some of the demos come from the Shiny tutorial: https://github.com/rstudio-education/shiny.rstudio.com-tutorial

Multiple Inputs

- Taking the hist demo in Part 1 as an example.
  - If we want to customize the histogram title, how can change the title without changing the source code?
  - We specify more inputs in the UI

```
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")
)
```



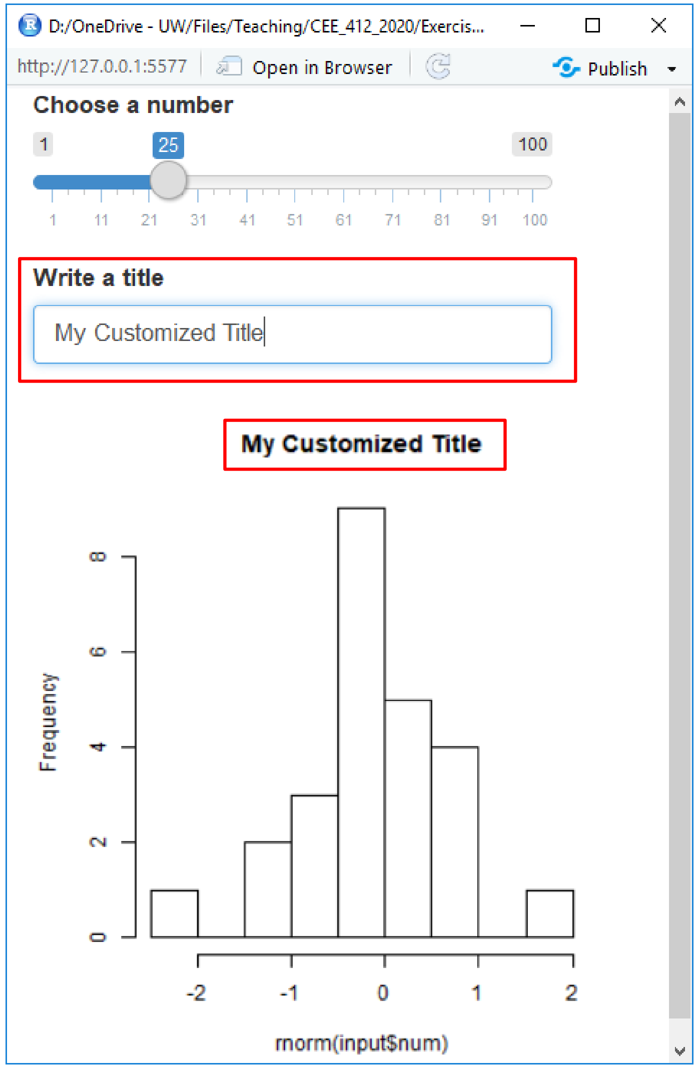
Multiple Inputs

Demo code file:
Exercises → Exercise 4 → Scripts → part_2_demo_1_two_inputs.R

- Create a new R file and run the following code:

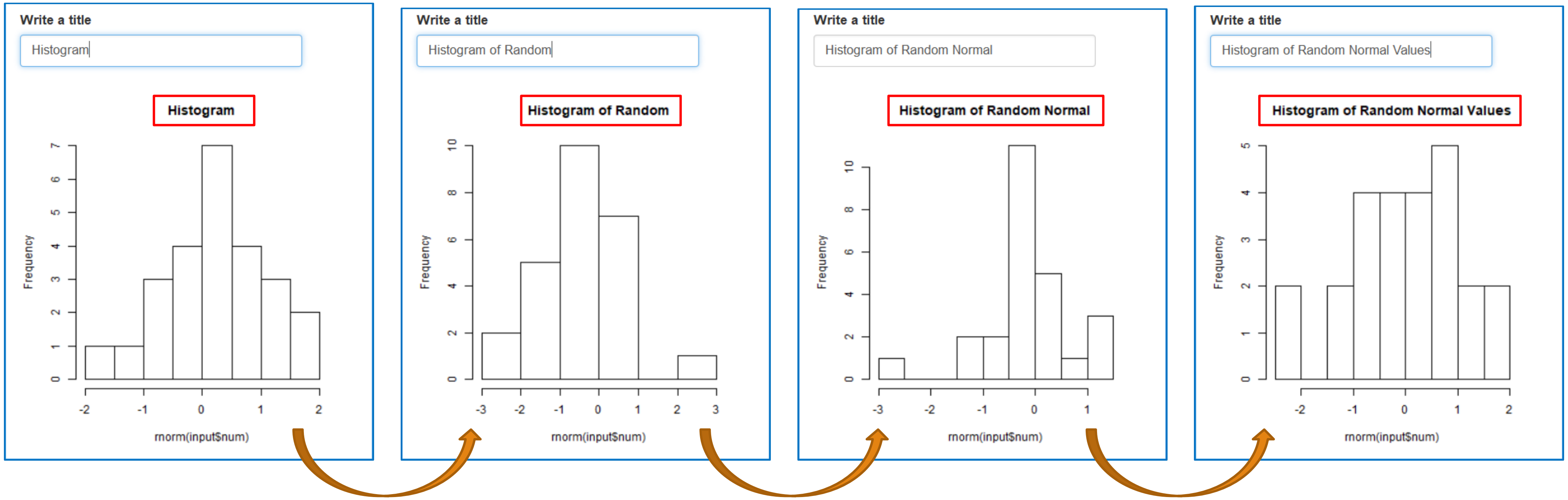
```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  textInput(inputId = "title",
    label = "Write a title",
    value = "Histogram of Random Normal Values"),
  plotOutput("hist")
)

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

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

Run



Reactivity

- In this example, you will find the title of the histogram updates when you typing the title in the textbox. It will influence the appearance of the histogram (it is updating the randomly generated samples). Why is that?



Reactivity

- Reasons:
  - The inputs are **reactive values**
    - Reactive value is the value that changes/ reacts to the input.
  - Reactive values work together with **reactive functions**, including rendering functions (page 23 in Exercise 4 Part 1).

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

This input will provide a value saved as input$num. It is a reactive value

work with →

Reactive functions

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

Not Reactive function

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

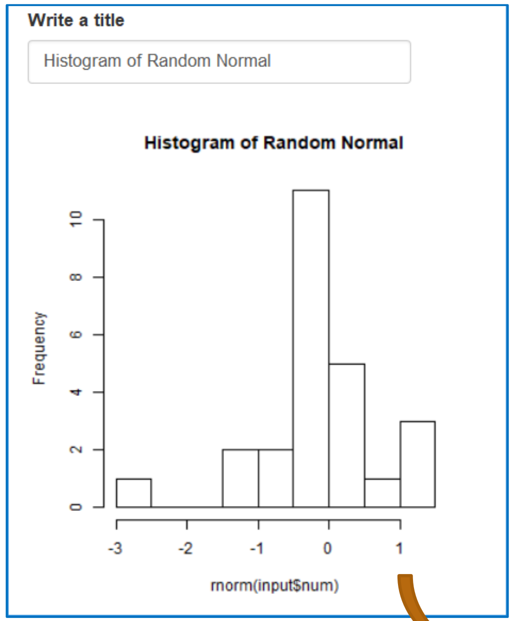
Isolate()

- Coming back to our question, can we prevent the title field from updating the plot?
- Yes. Use the **Isolate()** function
  - It returns the result as a non-reactive value

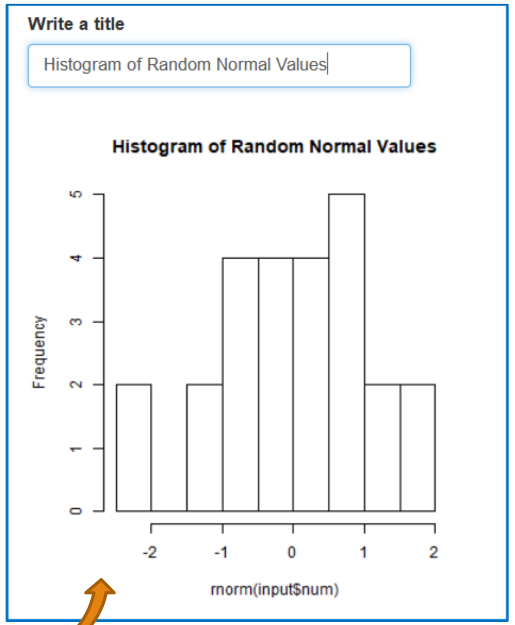
```
isolate( { hist( rnorm( input$num ) ) } )
```

Object will NOT respond to any reactive value in the code

Code used to build object


Write a title

Histogram of Random Normal


Write a title

Histogram of Random Normal Values

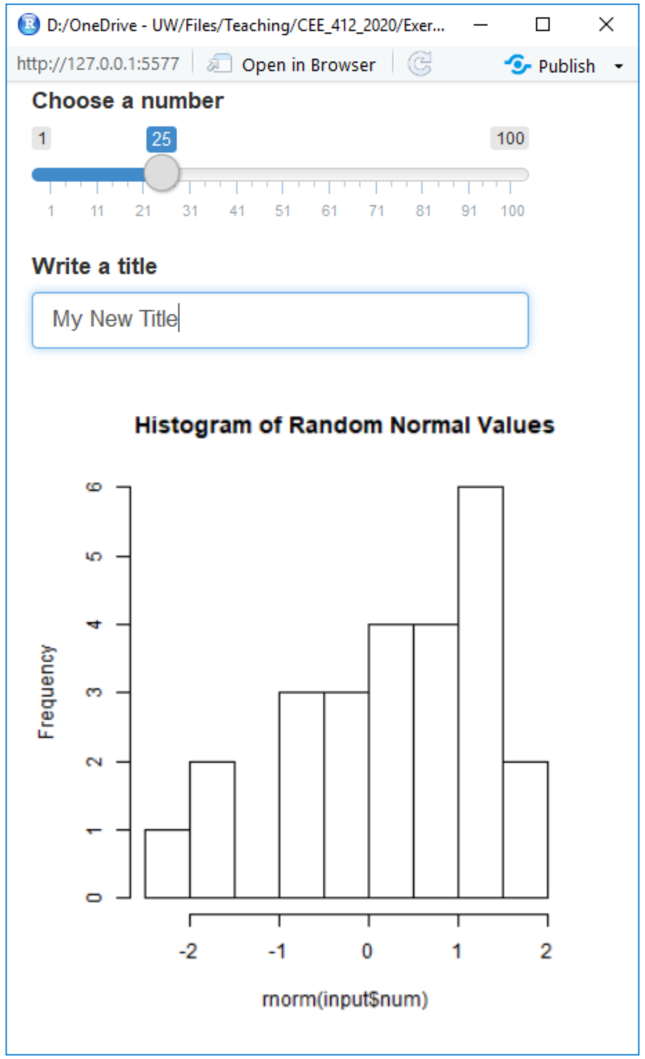
Isolate()

Demo code file:
Exercises → Exercise 4 → Scripts → part_2_demo_2_isolate.R

- Create a new R file and run the following code:

```
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)
```

Run



Reactive ()

- Use the **reactive()** function
  - It returns the result as a reactive value
    - For example:

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

    - This data (an reactive object) can be used by the output in the server function.
    - The reactive object can be used in multiple outputs.
    - The reactive() function is very useful for automatically updating UI given a local variable in your R code.

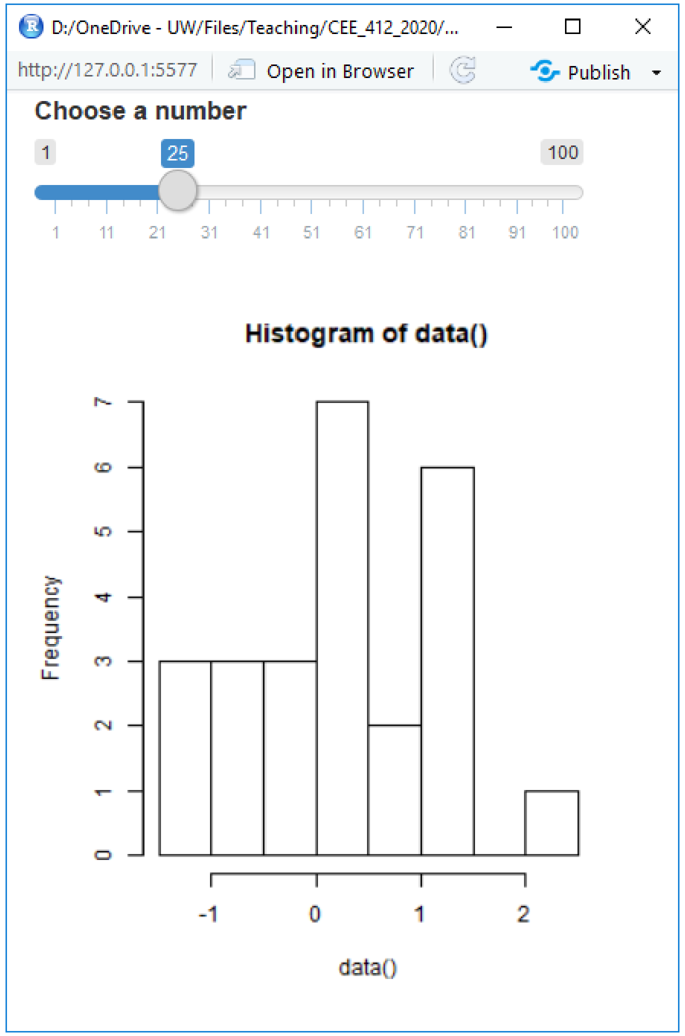
Demo code file:
Exercises → Exercise 4 → Scripts → part_2_demo_3_reactive.R

Reactive ()

- Create a new R file and run the following code:

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

Run



Multiple Outputs

- Taking the same hist demo as an example
  - If we want to summarize the randomly generated data to show the max, min, mean, etc., how can we display these values?
  - We add more outputs in the UI and define the outputs in the server function:

```
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))
        })
}
```

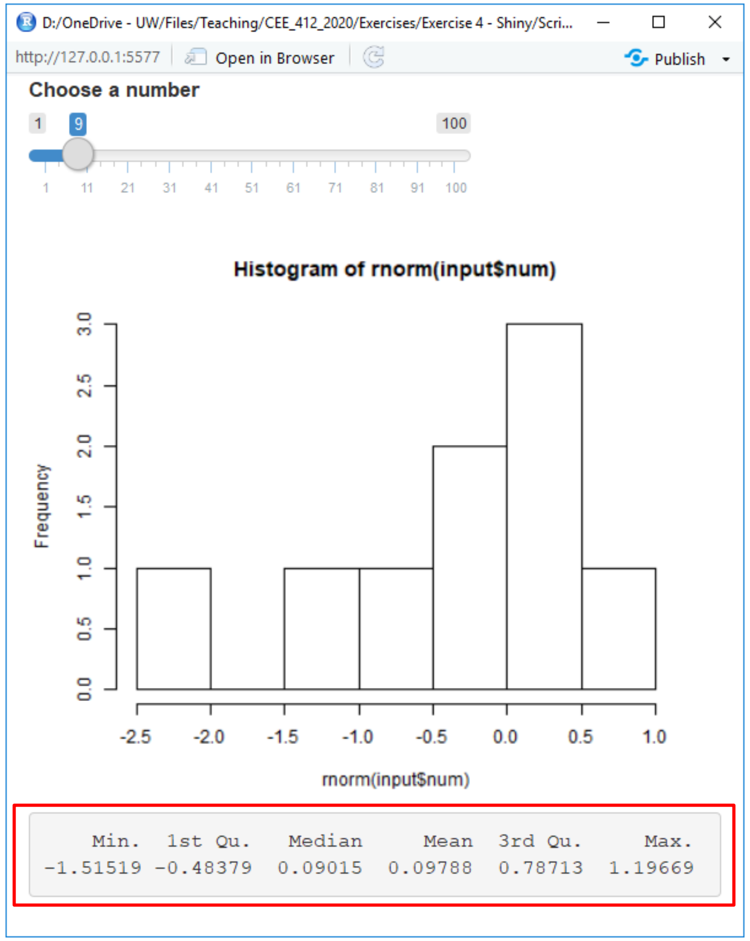
Multiple Outputs

Demo code file:
Exercises → Exercise 4 → Scripts → part_2_demo_4_two_outputs.R

- Create a new R file and run the following code:

```
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)
```

Run



D:/OneDrive - UW/Files/Teaching/CEE_412_2020/Exercises/Exercise 4 - Shiny/Scri...
http://127.0.0.1:5577
Open in Browser
Publish
Choose a number
1
9
100
Histogram of rnorm(input$num)
Frequency
rnorm(input$num)

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
-1.51519 -0.48379  0.09015  0.09788  0.78713  1.19669 
```

Event Handling

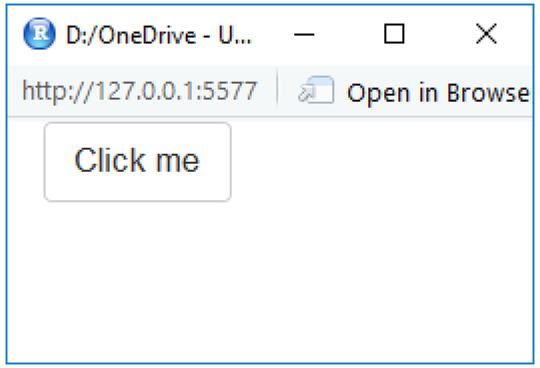
- Action Buttons

```
actionButton(inputId = "clicks", label = "Click me")
```

Input function

Input name (for internal use)

Label to display



- In server side, it needs an observeEvent()

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

```
observeEvent(input$clicks, {print(as.numeric(input$clicks))})
```

Reactive value (the event) to respond to

Code block to run whenever it observe the event

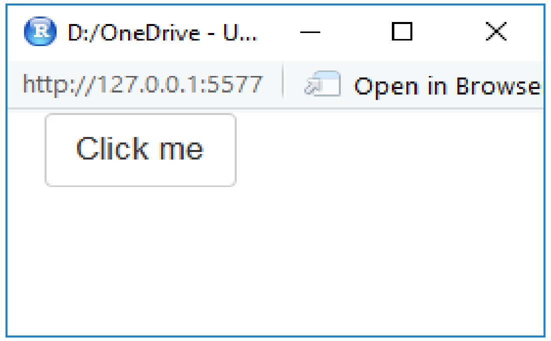
Event Handling

Demo code file:
Exercises → Exercise 4 → Scripts
→ part_2_demo_5_actionButton.R

- Create a new R file and run the following code:

```
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)
```

Run



Results shown in the R console:

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Find more info about action buttons: http://shiny.rstudio.com/articles/action-buttons.html

eventReactive()

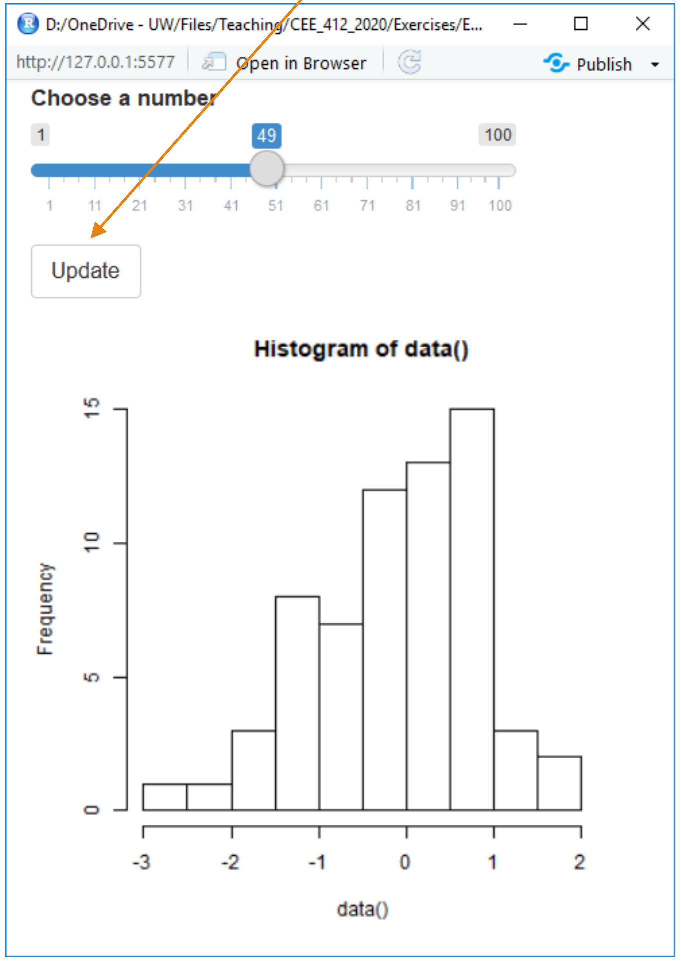
- Can we prevent the graph from updating until we hit the button?
- Yes. Using eventReactive() to delay reactions

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

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

Reactive value to respond to

Code used to build the object



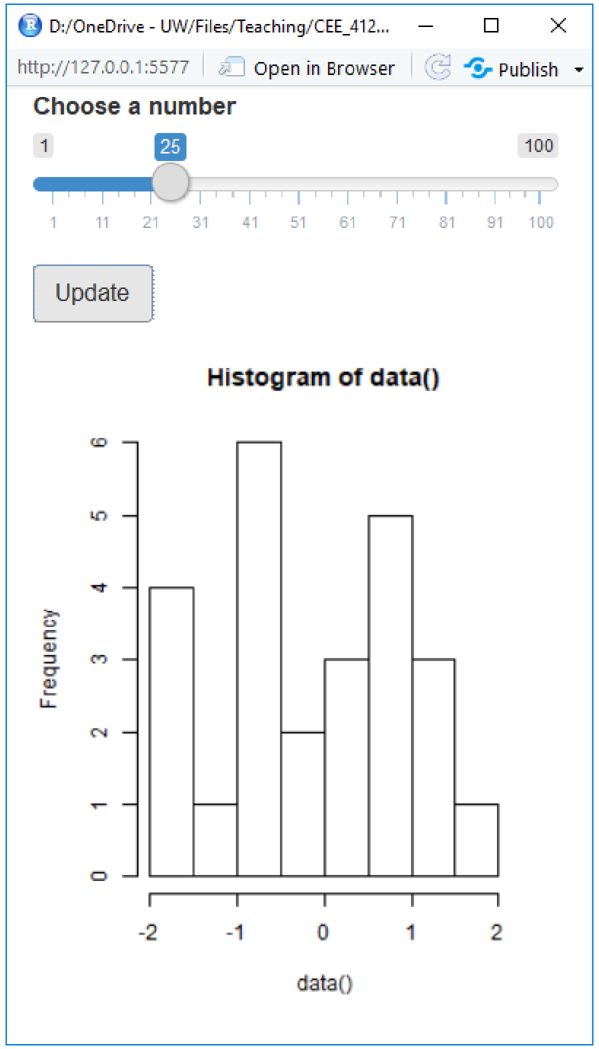
eventReactive()

Demo code file:
Exercises → Exercise 4 → Scripts → part_2_demo_6_eventReactive.R

- Create a new R file and run the following code:

```
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)
```

Run



Update your data (reactive values)

- Use reactiveValues()
  - It can create **a list of reactive values** to manipulate programmatically.

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

Elements to add to the list respond to

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

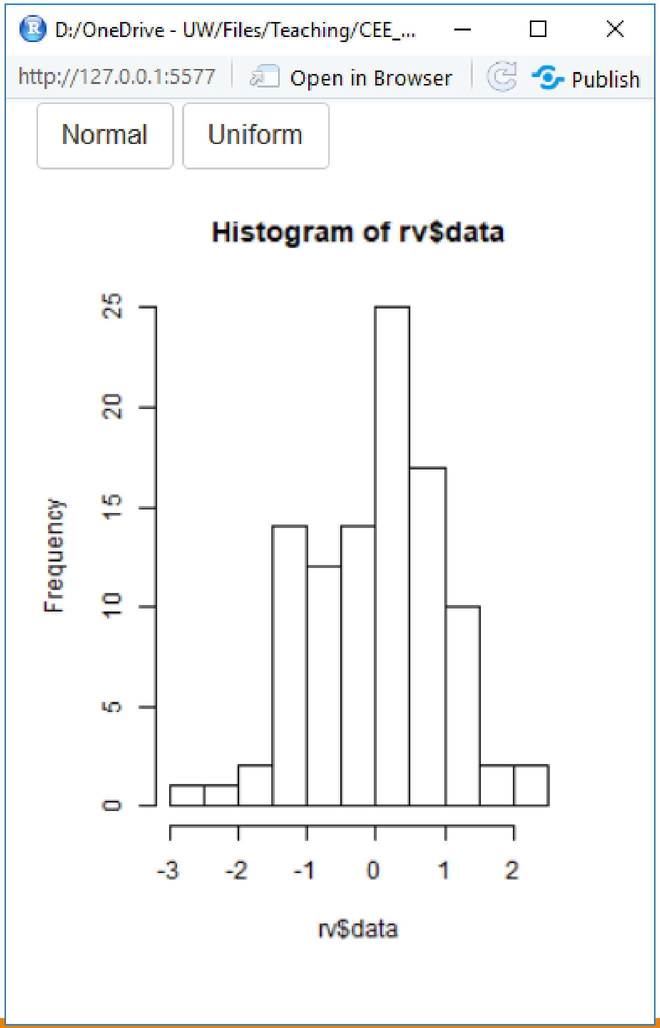
reactiveValues()

Demo code file:
Exercises → Exercise 4 → Scripts → part_2_demo_7_reactiveValues.R

- Create a new R file and run the following code:

```
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)
```

Run →


D:/OneDrive - UW/Files/Teaching/CEE_...
http://127.0.0.1:5577
Open in Browser
Publish
Normal
Uniform
Histogram of rv$data
Frequency
rv$data
0
5
10
15
20
25
-3
-2
-1
0
1
2