

# Introduction to R Shiny

---

Workshop on Data Visualization in R

**Lokesh Mano** • 18-Oct-2022

NBIS, SciLifeLab

# Contents

- Introduction to RShiny
- Code Structure
- App execution
- UI
- Server
- Reactivity
- Isolate reactivity
- `observeEvent()`
- Updating Widgets
- Error Validation
- Download Button
- Modularizing reactivity

## Life sciences



COVID-19 tracker



Exploring large hospital data for better use of antimicrobials



ShinyMRI - View MRI images in Shiny



A/B Testing Sample Size Calculator



ctmweb, a web app to analysis Animal tracking data



Visualizing Biodiversity in National Parks data



iSEE



MOTE: An Effect Size Calculator



Interactively view and subset phylogenetic trees

# What is shiny?

- Interactive documents & web applications
- Completely created using R
- Needs a live environment

## Usage

- [Standalone web applications](#)
- [Dashboard/Flexboard](#)
- Interactive RMarkdown
- Gadgets/RStudio extensions

## App structure

- UI Layout
- UI Inputs (Widgets)
- UI Outputs
- Renderer
- Builder

# Code structure

## One file format

*app.R*

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

## Two file format

*ui.R*

```
ui <- fluidPage()
```

*server.R*

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

# App execution

- Change to app directory, then run `runApp()`
- Use `shinyApp()`

```
shinyApp(  
  ui=fluidPage(),  
  server=function(input,output) {}  
)
```

- From Rmd file using `rmarkdown::run()`
- Running as a separate process from terminal

```
R -e "shiny::runApp('~/.shinyapp')"
```

# UI • Layout

```
shinyApp(  
  ui=fluidPage(  
    titlePanel("Title Panel"),  
    sidebarLayout(  
      sidebarPanel(  
        helpText("Sidebar Panel")  
      ),  
      mainPanel(tabsetPanel(  
        tabPanel("tab1",  
          fluidRow(  
            column(6,helpText("Col1")),  
            column(6,  
              helpText("Col2"),  
              fluidRow(  

```

## Title Panel



# UI • Widgets • Input

```
shinyApp(  
  ui=fluidPage(  
    fluidRow(  
      column(4,  
        fileInput("file-input", "fileInput:"),  
        selectInput("select-input", label="selectI  
        numericInput("numeric-input", label="numer  
        sliderInput("slider-input", label="sliderI  
        textInput("text-input", label="textInput")  
        textAreaInput("text-area-input", label="te  
        dateInput("date-input", label="dateInput")  
        dateRangeInput("date-range-input", label=""  
        radioButtons("radio-button", label="radioB  
        checkboxInput("checkbox", "checkboxInput"),  
        actionButton("action-button", "Action"),  
        hr(),  
        submitButton()  
      )  
    ),  
    server=function(input, output) {  
    }  
  })  
)
```

fileInput:

selectInput

numericInput

sliderInput

textInput

textAreaInput

dateInput

dateRangeInput

to

radioButtons

☒ A ☐ B ☐ C

☐ checkboxInput

Widgets gallery



# UI • Widgets • Outputs

```
shinyApp(  
  ui=fluidPage(fluidRow(column(5,  
    textInput("text_input",label="textInput",  
    hr(),  
    htmlOutput("html_output"),  
    textOutput("text_output"),  
    verbatimTextOutput("verbatim_text_output"),  
    tableOutput("table_output"),  
    plotOutput("plot_output",width="300px",height="300px"),  
  )),  
  server=function(input, output) {  
    output$html_output <- renderText({input$text_input})  
    output$text_output <- renderText({input$text_input})  
    output$verbatim_text_output <- renderText({input$text_input})  
    output$table_output <- renderTable({iris[1:3,1:3]})  
    output$plot_output <- renderPlot({  
      plot(iris[,1],iris[,2])  
    })  
  })  
})
```

textInput

<h3 style='color:red'>Red text</h3>

Red text

<h3 style='color:red'>Red text</h3>

<h3 style='color:red'>Red text</h3>

Sepal.Length	Sepal.Width	Petal.Length
5.10	3.50	1.40
4.90	3.00	1.40
4.70	3.20	1.30



# Dynamic UI

- UI elements are created conditionally using `uiOutput()` / `renderUI()`

```
shinyApp(  
  ui=fluidPage(  
    selectInput("data",label="Select data",  
               choices=c("mtcars","faithful","iris")),  
    tableOutput("table"),  
    uiOutput("ui")  
  ),  
  server=function(input, output) {  
  
    data <- reactive({ get(input$data, 'package:datasets') })  
  
    output$ui <- renderUI({  
      if(input$data=="iris") plotOutput("plot",width="400px")  
    })  
  
    output$plot <- renderPlot({hist(data()[, 1])})  
    output$table <- renderTable({head(data())})  
  })  
)
```

- Other options include `conditionalPanel()`, `insertUI()` and `removeUI()`

# Server

- `Server` is a function that assembles your `input` into `output` using R based code.
- Three rules to be followed to write a server function:

Rule 1: Save objects to display to `output$`



Rule 2: Build objects to display with `render*()`

- R-Code block (can even be an entire R script) between the braces `{}` inside the `render*()` function.

```
output$hist <- renderPlot({  
  title <- "histogram of 100 random numbers"  
  hist(rnorm(100), main = title)  
})
```

Different Render functions

Rule 3: Use input values with `input$`

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



```
input$num
```

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

# Reactivity

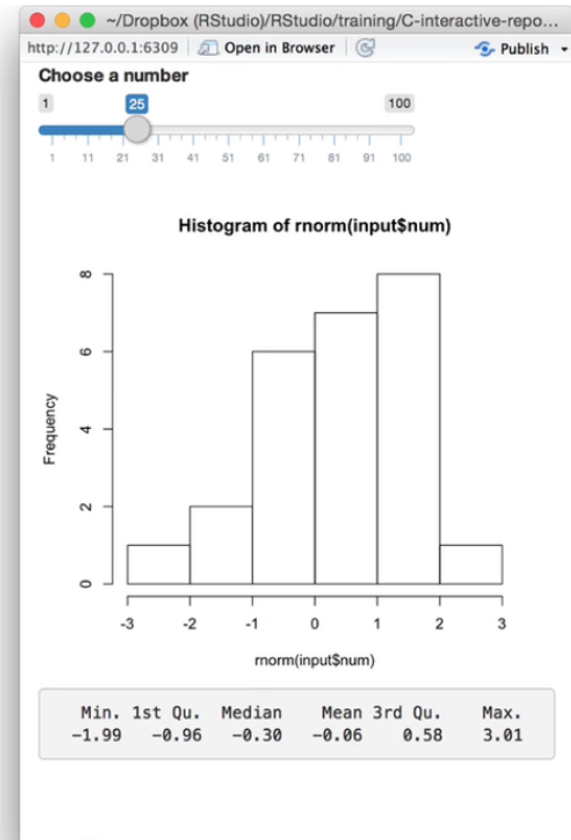
```
# 02-two-outputs

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)
```



# Reactivity

input\$num

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

```
output$stats <-  
  renderPrint({  
    summary(rnorm(input$num))  
  })
```



# Reactivity

input\$num

Can these describe  
the same data?

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

```
output$stats <-  
  renderPrint({  
    summary(rnorm(input$num))  
  })
```





# Reactivity



## reactive()

Builds a reactive object (reactive expression)

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

# Reactivity

```
# 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)
```



# Isolate reactivity

- Reactivity can be controlled.
- You will notice that as soon as you try to change the title, the histogram will update with new values

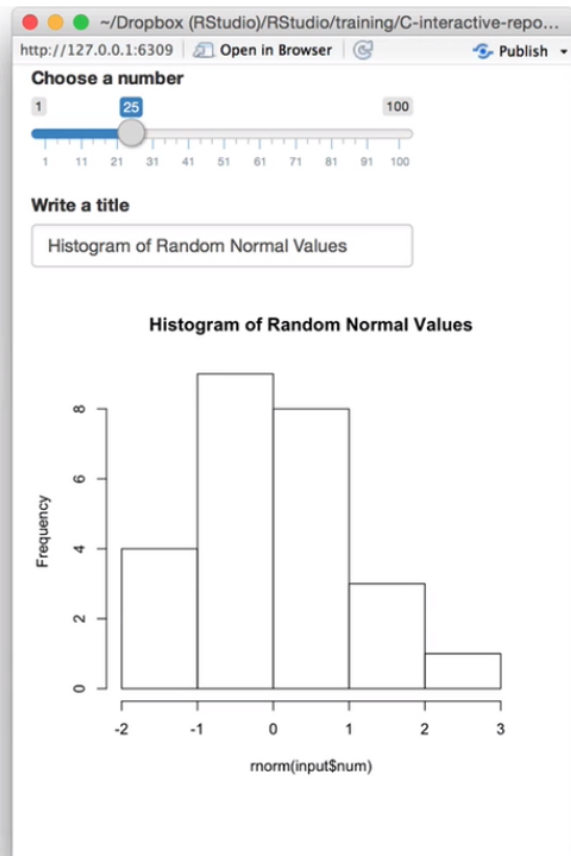
```
# 01-two-inputs

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)
```



# Isolate reactivity

```
# 01-two-inputs
```

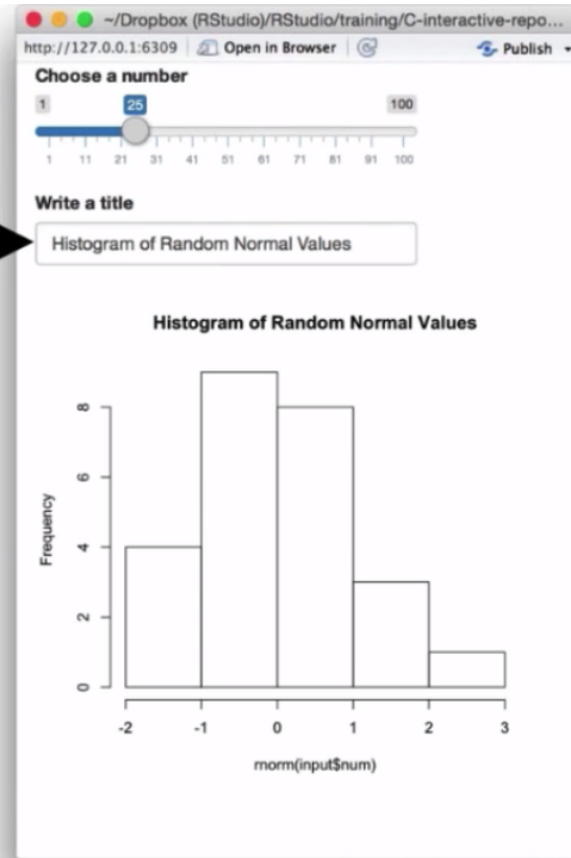
```
library(shiny)
```

```
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?**



## isolate()

Returns the result as a non-reactive value

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

# Isolate reactivity

```
# 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)
```



# observeEvent()

```
# 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)
```





# Updating widgets

- Widgets can be updated once initialised.
- Add third argument **session** to server function

```
server=function(input,output,session) {}
```

- Example of a typical UI

```
ui=fluidPage(  
  selectInput("select-input",label="selectInput",choices=c("A","B","C")),  
  numericInput("numeric-input",label="numericInput",value=5,min=1,max=10),  
  sliderInput("slider-input",label="sliderInput",value=5,min=1,max=10),  
)
```

- Update functions can be used to update input widgets
- Reactive observer `observe({})` monitors for a conditional change

```
server=function(input,output,session) {  
  observe({  
    if(something) {  
      updateSelectInput(session,"select-input",label="selectInput",choices=c("D","E","F"))  
      updateNumericInput(session,"numeric-input",label="numericInput",value=10,min=1,max=  
      updateSliderInput(session,"slider-input",label="sliderInput",value=8,min=1,max=10)  
    }  
  })  
}
```

# Error validation

- Shiny returns an error with missing or incorrect values

```
shinyApp(  
  ui=fluidPage(  
    selectInput("data_input",label="Select data",  
               choices=c("", "mtcars", "faithful", "iris")),  
  )  
)
```

- Errors can be handled in a controlled manner
- `validate()` can be used to check input
- `validate()` using `need()`

Select data

Error: invalid first argument  
Error: invalid first argument

```
shinyApp(  
  ui=fluidPage(  
    selectInput("data_input",label="Select data",  
               choices=c("", "mtcars", "faithful", "iris")),  
  )  
)
```

- `validate()` using custom function

Select data

Please select a data set  
Please select a data set

```
valfn <- function(x) if(is.null(x) | is.na(x) | x=="") return("Invalid input")  
shinyApp(  
  ui=fluidPage(  
    selectInput("data_input",label="Select data",  
               choices=c("", "mtcars", "faithful", "iris")),  
  )  
)
```

- `shiny::req()` checks input variable and silently stops execution

Select data

Input data is incorrect.  
Input data is incorrect.

# Download • Data

- Add button and `downloadHandler()` function

```
shinyApp(  
  ui=fluidPage(  
    selectInput("data_input",label="Select data",  
               choices=c("mtcars","faithful","iris")),  
    textOutput("text_output"),  
    downloadButton("button_download","Download")  
  ),  
  server=function(input, output) {  
    getdata <- reactive({ get(input$data_input, 'package:datasets') })  
    output$text_output <- renderText(paste0("Selected dataset: ",input$data_input))  
  
    output$button_download <- downloadHandler(  
      filename = function() {  
        paste0(input$data_input,".csv")  
      },  
      content = function(file) {  
        write.csv(getdata(),file,row.names=FALSE,quote=F)  
      })  
  })
```

- Run in system browser if Rstudio browser doesn't work
- See usage of download buttons

# Download • Plots

```
shinyApp(  
  ui=fluidPage(  
    selectInput("data_input",label="Select data",  
               choices=c("mtcars","faithful","iris")),  
    textOutput("text_output"),  
    plotOutput("plot_output",width="400px"),  
    downloadButton("button_download", "Download")  
  ),  
  server=function(input, output) {  
    getdata <- reactive({ get(input$data_input, 'package:datasets') })  
    output$text_output <- renderText(paste0("Selected dataset: ",input$data_input))  
    output$plot_output <- renderPlot({hist(getdata()[, 1])})  
  
    output$button_download <- downloadHandler(  
      filename = function() {  
        paste0(input$data_input, ".png")  
      },  
      content = function(file) {  
        png(file)  
        hist(getdata()[, 1])  
        dev.off()  
      })  
    })  
})
```

- Run in system browser if Rstudio browser doesn't work
- See usage of download buttons



# Thank you. Questions?

Slide courtesy: Roy Francis (NBIS, RaukR2021)

R version 4.1.3 (2022-03-10)

Platform: x86\_64-pc-linux-gnu (64-bit)

OS: Ubuntu 18.04.6 LTS

---

Built on: 📅 18-Oct-2022 at 🕒 09:57:13

2022 • SciLifeLab • NBIS