Exercise IV – R Shiny

Part 1 – Build a Shiny App

CEE412 / CET522

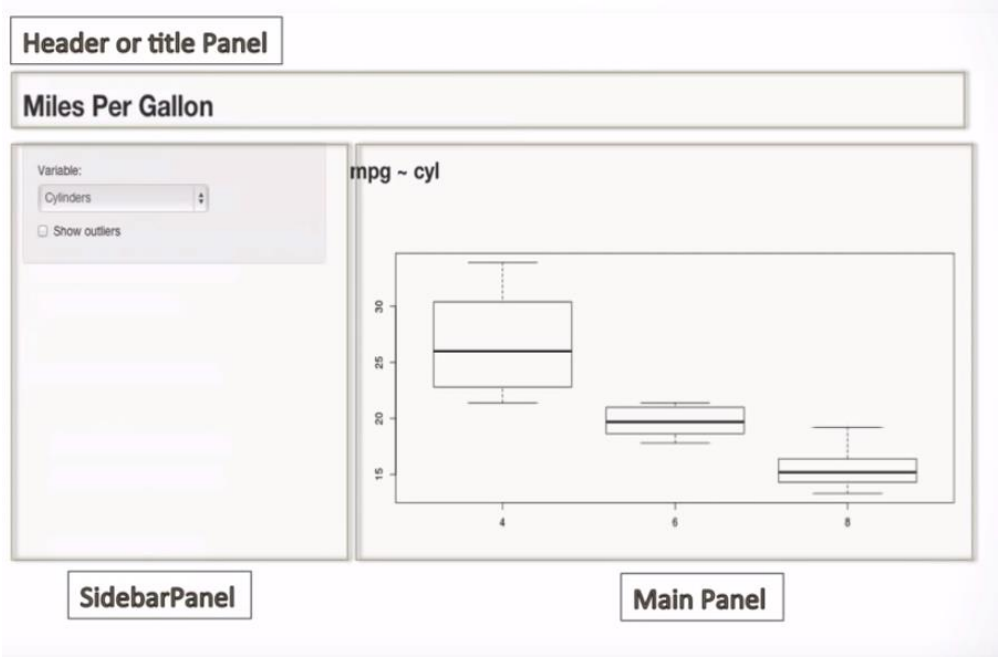
TRANSPORTATION DATA MANAGEMENT AND VISUALIZATION

WINTER 2020

Run a Shiny App

Introduction

- The aim of this exercise is to provide a brief introduction to R Shiny package and introduce application development.
- Following is an example of an application developed in Shiny based on the introduction provided to the Shiny application structure.



Run a Shiny Demo

- Open Rstudio and install the shiny package.

```
install.packages("shiny")
```

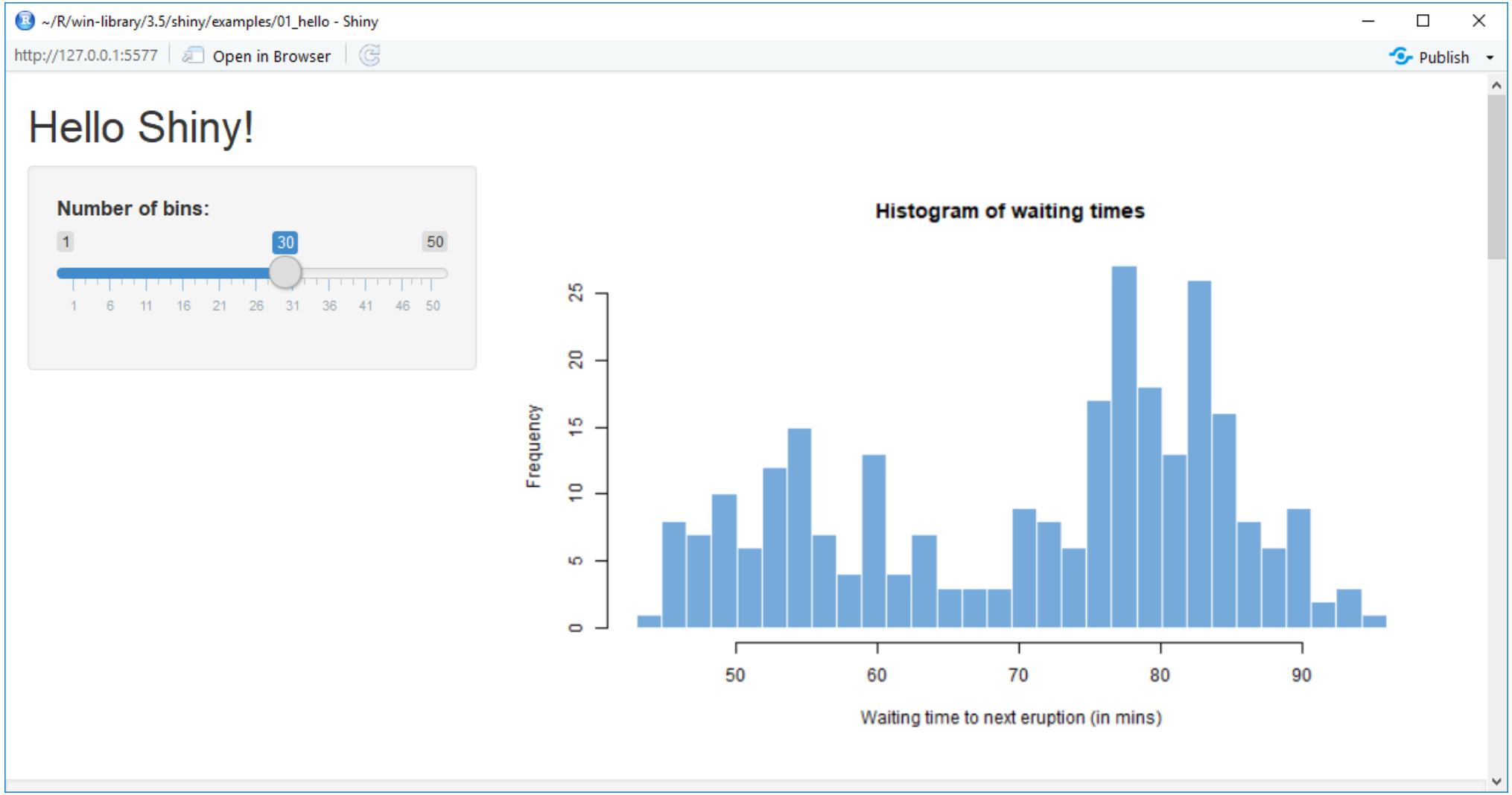
- Run a Shiny demo, you will get a window, as shown in the next slide.

```
library(shiny)
runExample("01_hello")
```

- Other demos:

```
runExample("01_hello") # a histogram
runExample("02_text") # tables and data frames
runExample("03_reactivity") # a reactive expression
runExample("04_mpg") # global variables
runExample("05_sliders") # slider bars
runExample("06_tabsets") # tabbed panels
runExample("07_widgets") # help text and submit buttons
runExample("08_html") # Shiny app built from HTML
runExample("09_upload") # file upload wizard
runExample("10_download") # file download wizard
runExample("11_timer") # an automated timer
```

Demo of 01_hello

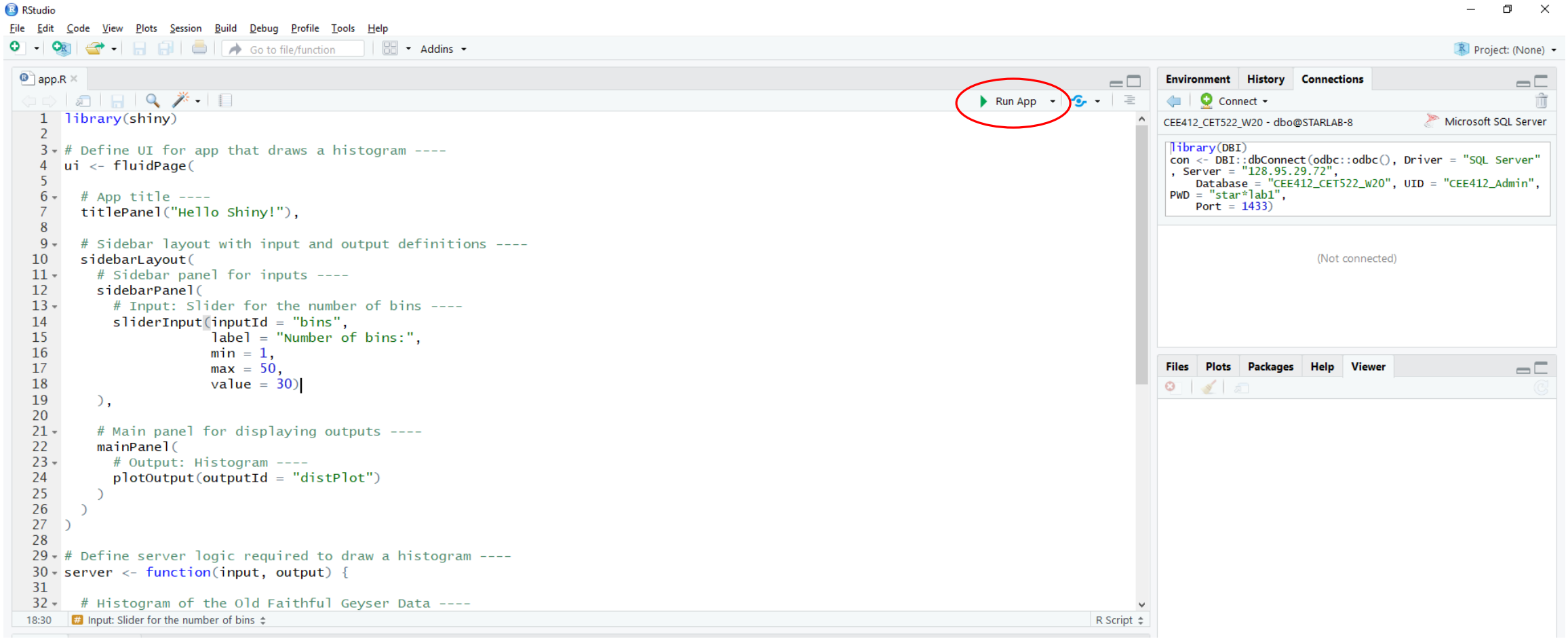


Let's check the details of the Demo

- Structure of a Shiny App
  - A user interface object (ui): controlling the layout and appearance of your app.
  - A server function (server): containing the instructions that your computer needs to build your app.
  - A call to the shinyApp function: creating Shiny app objects from an explicit UI/server pair.
- The ui object and server function of the demo are shown in the next slides. You can also find the source code in the scripts folder:
  - Exercises → Exercise 4 → Scripts → part_1_demo_1_hello.R
- Open this file and run it.

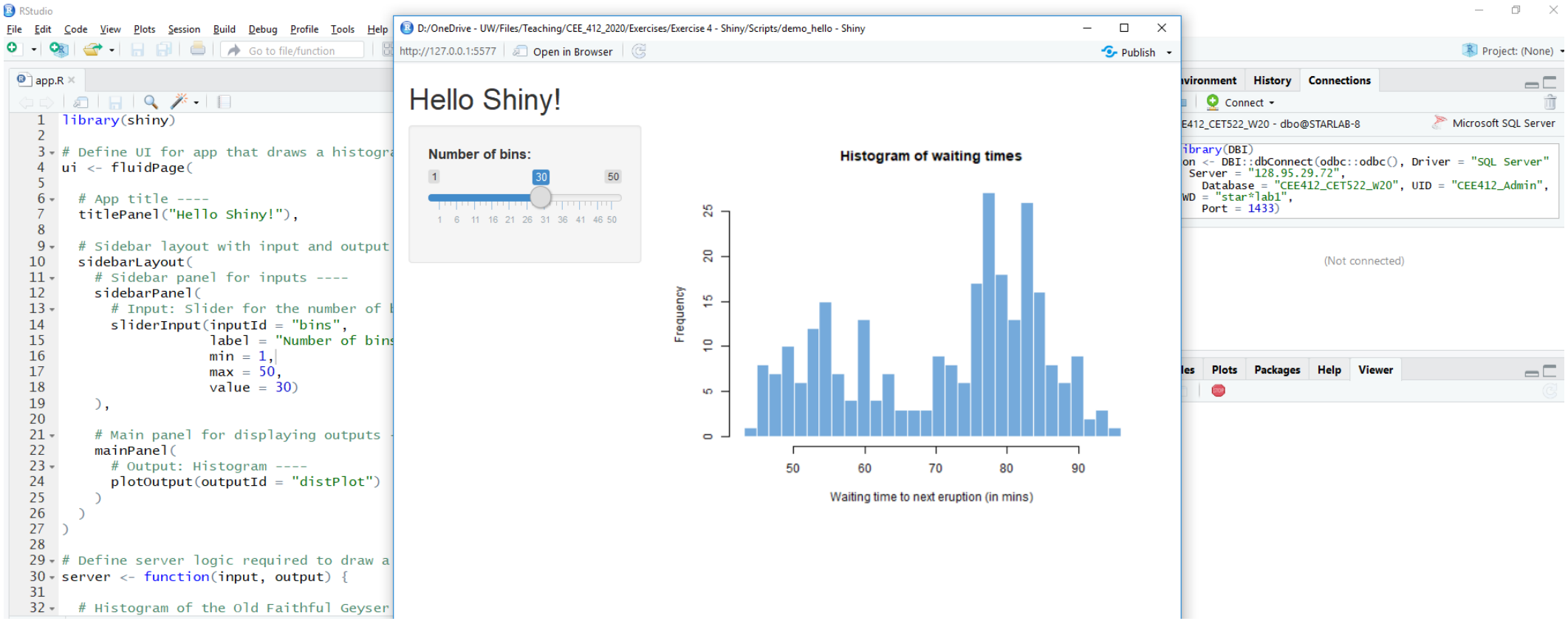
Run a Shiny App

Open this file and run it by clicking on "Run App"



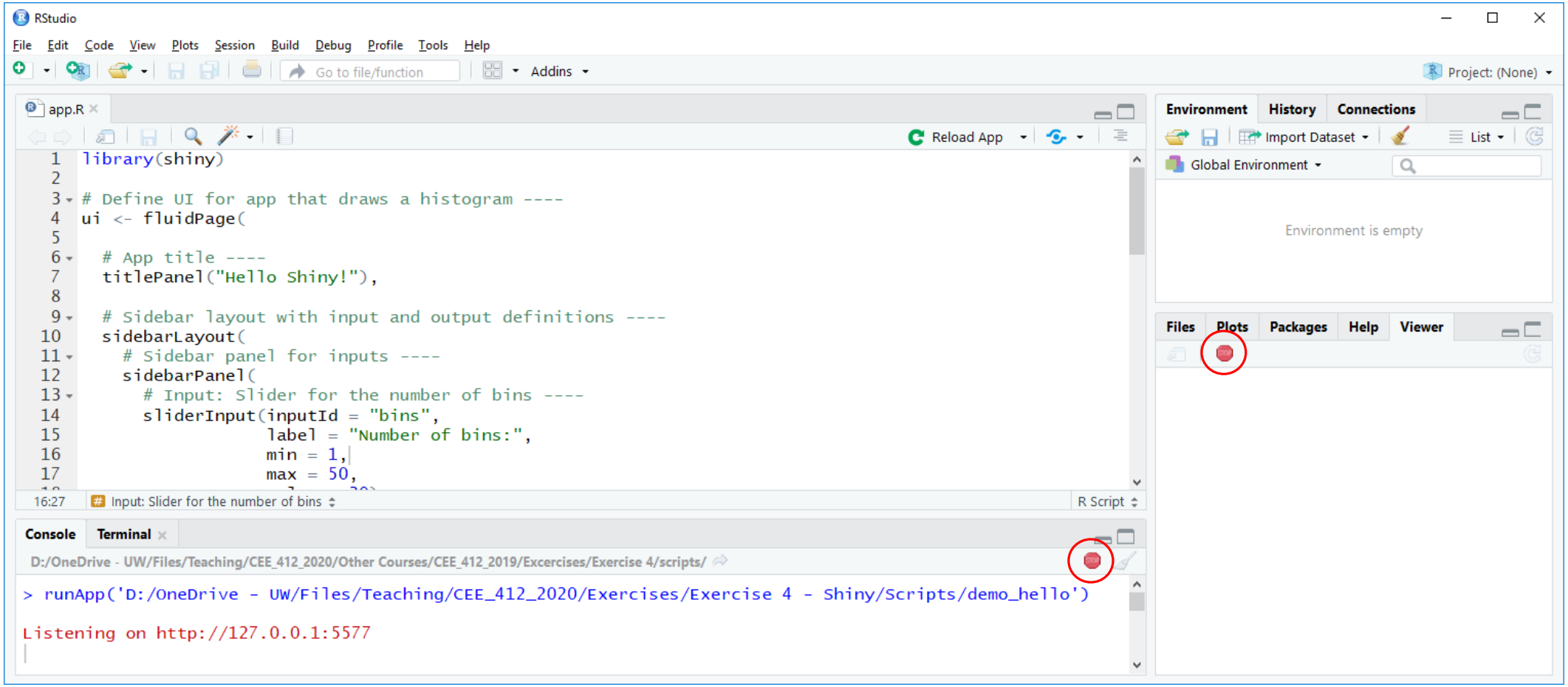
Run a Shiny App

A new window with the application will pop up.



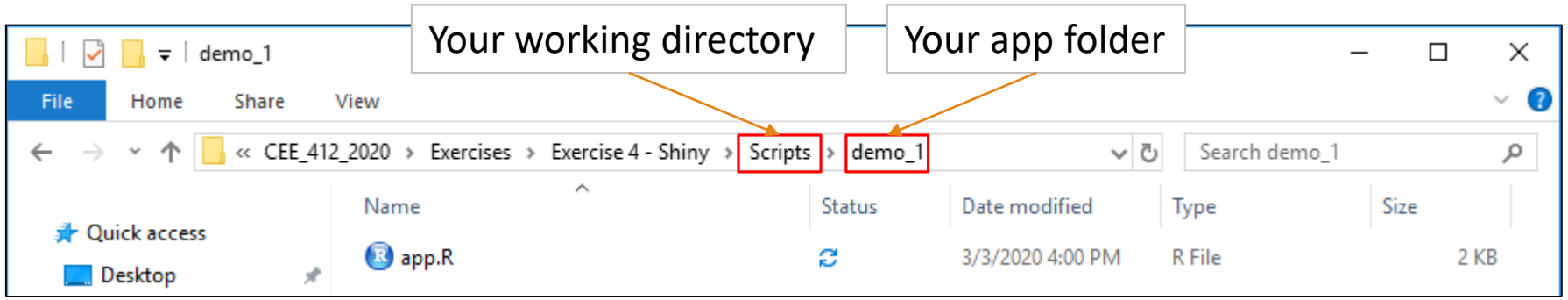
Run a Shiny App

Terminate the Shiny App by clicking on the "stop icon"



Run a Shiny App

- You can create a new Shiny app by creating a new script named as app.R. Copy all code from the demo_hello.R to app.R and save the script in a directory (for example, …/demo_1/).
- The app can be run with runApp("demo_1"). If you want to use this way to run the App, please remember to set your working directory to the one that contains the demo_1 folder (using the setwd() command).
  - For example:



Source Code of demo_hello

```r
# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(
    # Sidebar panel for inputs ----
    sidebarPanel(
      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

    # Main panel for displaying outputs ----
    mainPanel(
      # Output: Histogram ----
      plotOutput(outputId = "distPlot")
    )
  )
)
```

```r
# Define server logic required to draw a histogram ----
server <- function(input, output) {

  # Histogram of the Old Faithful Geyser Data ----
  # with requested number of bins
  # This expression that generates a histogram is wrapped in a call
  # to renderPlot to indicate that:
  #
  # 1. It is "reactive" and therefore should be automatically
  #    re-executed when inputs (input$bins) change
  # 2. Its output type is a plot
  output$distPlot <- renderPlot({

    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    hist(x, breaks = bins, col = "#75AADB", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")

  })

}
```

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

File location: Exercises → Exercise 4 → Scripts → demo_hello → app.R)

Notes

- Note: Prior to version 0.10.2, Shiny did not support single-file apps and the ui object and server function needed to be contained in separate scripts called ui.R and server.R, respectively. This functionality is still supported in Shiny, however exercises in this class and much of the supporting documentation focus on single-file apps.

Shiny App Structure

Simplest App Template

- A shortest viable Shiny app

```
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

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

Run

- Every app contains ui, server, and shinyApp
- How can we build the connection between them?

Connecting UI and Server

- In UI: add elements to your app as arguments to fluidPage()
  - Create reactive inputs with an ***Input()** function
  - Display reactive results with an ***Output()**function

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

- In Server: assemble inputs into outputs
  - Assemble outputs from inputs in the server function

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



}
```

Inputs in UI

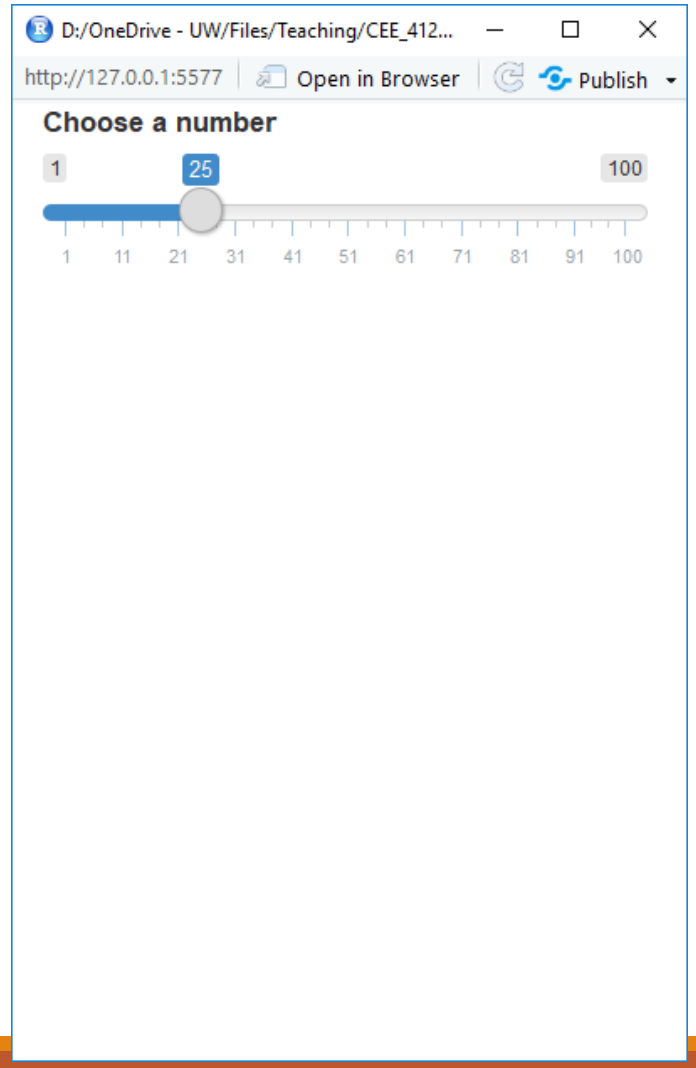
- Create an input with an ***Input()** function.
  - Example: a sliderInput

```
sliderInput(inputId = "num",
            label = "Choose a number",
            value = 25, min = 1, max = 100)
```

  - Create a new script and run the following code

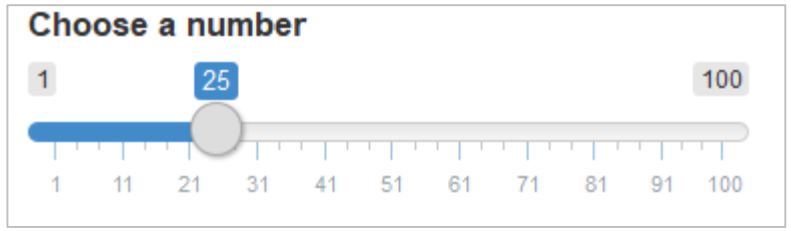
```
library(shiny)
ui <- fluidPage(
        sliderInput(inputId = "num",
            label = "Choose a number",
            value = 25, min = 1, max = 100)
)

server <- function(input, output) {}

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

Run



Inputs in UI

- Syntax of input functions




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

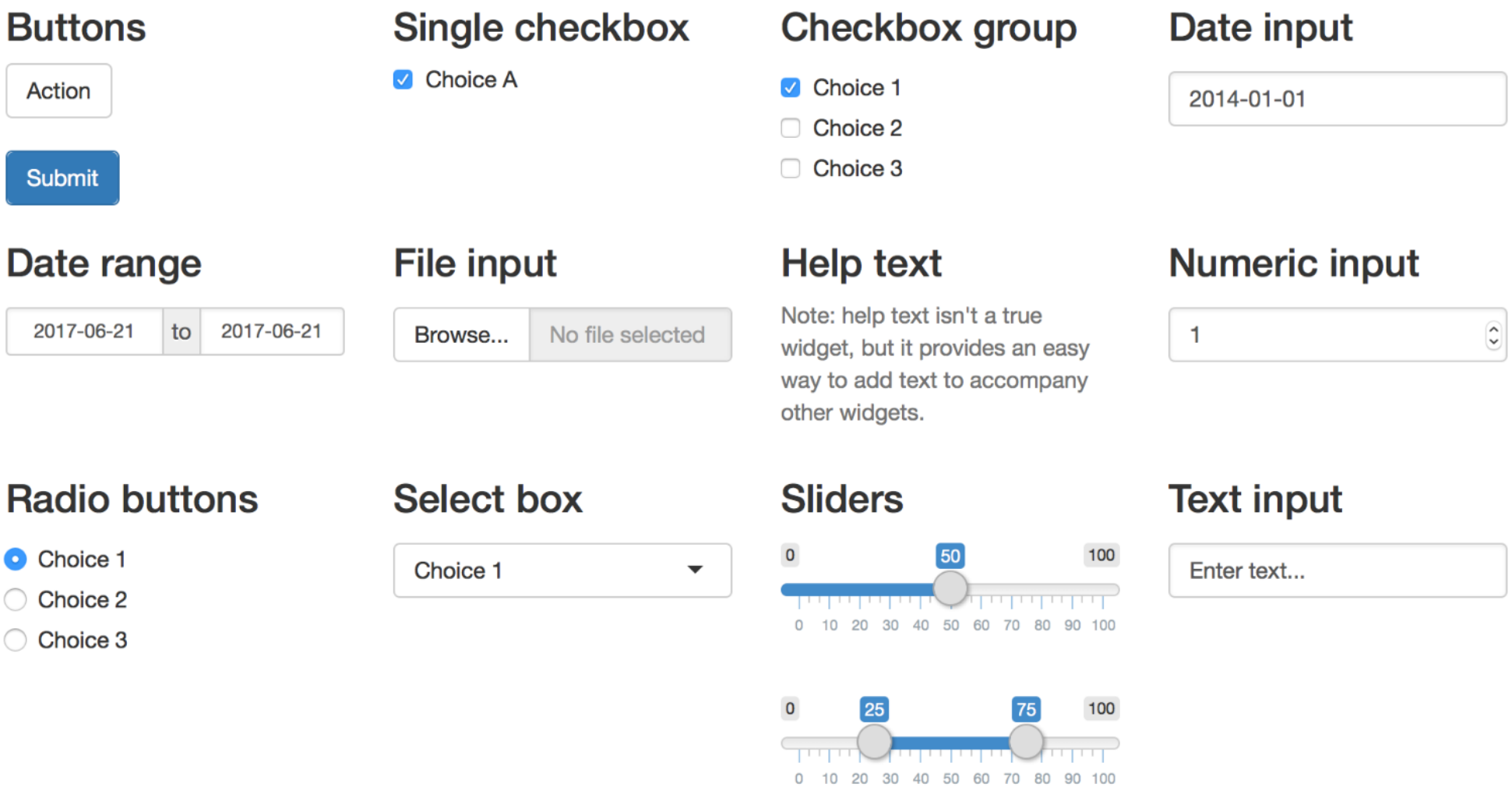
Input name
(for internal use)

Label to display

Input specific arguments

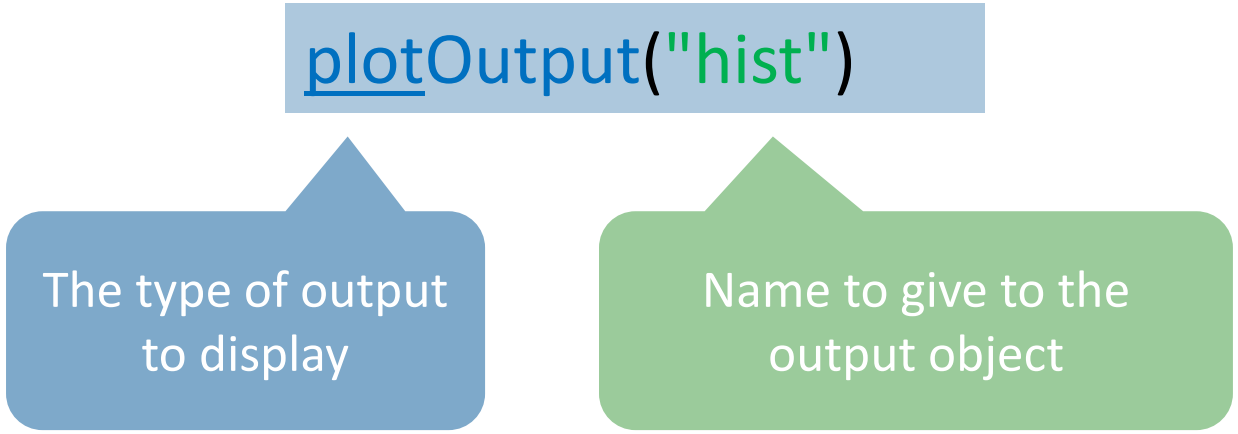
- Find more input functions, i.e. control widgets…
  - https://shiny.rstudio.com/tutorial/written-tutorial/lesson3/

Inputs in UI


Buttons
Action
Submit
Single checkbox
Choice A
Checkbox group
Choice 1
Choice 2
Choice 3
Date input
2014-01-01
Date range
2017-06-21
to
2017-06-21
File input
Browse...
No file selected
Help text
Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.
Numeric input
1
Radio buttons
Choice 1
Choice 2
Choice 3
Select box
Choice 1
Sliders
0
50
100
0 10 20 30 40 50 60 70 80 90 100
0
25
75
100
0 10 20 30 40 50 60 70 80 90 100
Text input
Enter text...

Outputs in UI

- To display output, add it to fluidPage() with an *Output() function



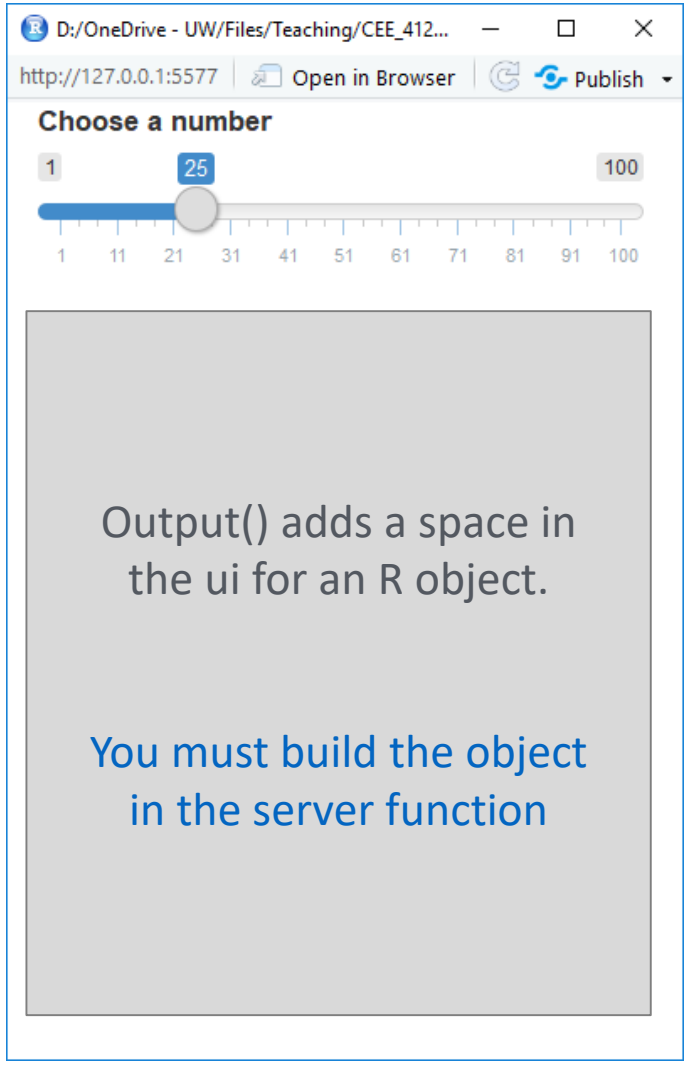
Outputs in UI

- Run the following code

```
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)
```

Remember the comma between arguments

Run



Type of Inputs and Outputs

- Input

function	widget
actionButton	Action Button
checkboxGroupInput	A group of check boxes
checkboxInput	A single check box
dateInput	A calendar to aid date selection
dateRangeInput	A pair of calendars for selecting a date range
fileInput	A file upload control wizard
helpText	Help text that can be added to an input form
numericInput	A field to enter numbers
radioButtons	A set of radio buttons
selectInput	A box with choices to select from
sliderInput	A slider bar
submitButton	A submit button
textInput	A field to enter text

- Output

Output function	Creates
dataTableOutput	DataTable
htmlOutput	raw HTML
imageOutput	image
plotOutput	plot
tableOutput	table
textOutput	text
uiOutput	raw HTML
verbatimTextOutput	text

Server Function

- Use 3 rules to write the server function
  1. Save objects to display to output$

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

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

Please ensure the output plots have the same name

Server Function

- Use 3 rules to write the server function
  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.

render function	creates
renderDataTable	DataTable
renderImage	images (saved as a link to a source file)
renderPlot	plots
renderPrint	any printed output
renderTable	data frame, matrix, other table like structures
renderText	character strings
renderUI	a Shiny tag object or HTML

Server Function

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

- **render*():** build reactive output to display in UI

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

Type of object to build

Code block that builds the object

- Rendering a histogram of 100 random normal values
  - Title: title of the hist
  - rnorm(): norm distribution

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

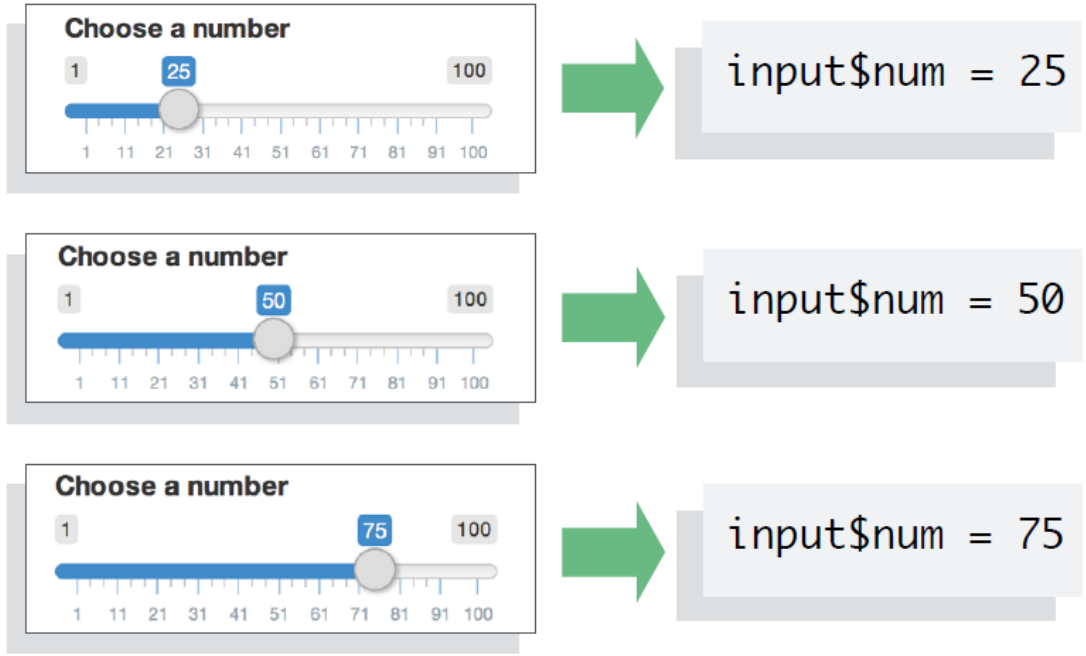
Server Function

- Use 3 rules to write the server function
  3. Access **input** values with input$

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

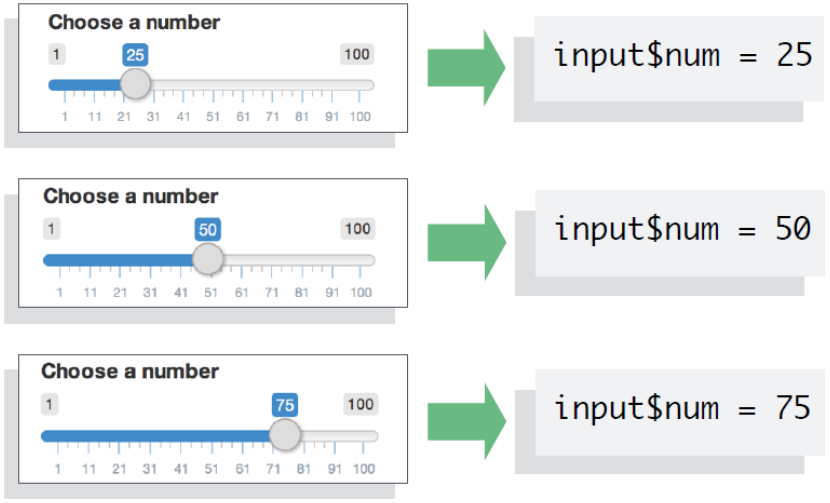
Please ensure the input variables have the same name

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



Server Function

- The input value changes whenever a user changes the input.



- Output will automatically update if you follow the 3 rules
  - Reactivity automatically occurs whenever you use an input value to render an output object

Recap: Server Function

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**

Run the demo

- Demo file:
  - Exercises → Exercise 4 → Scripts → part_1_demo_2_hist.R
  - Code:

```
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)
```

Run