INTRODUCTION TO R SHINY

ACTS-DBDA
Sep 2021, CDAC Bangalore

INTRODUCTION

- Shiny is an R package that makes it easy to build **interactive web applications** (apps) straight from R.

- To install shiny package
  - `install.packages("shiny")`

- To load shiny package
  - `library("shiny")`

- Example (11 in-built examples are available)
  - `runExample("01_hello")`

STRUCTURE OF A SHINY APP

- Three main components in building shiny apps
    1. UI Function
        1. Handles the user interface layout and appearance of the app.
    2. Server Function
        1. Contains the instructions (actual code) needed by computer to build app.
    3. Call to Shiny app function
        1. The shinyapp() function creates application from an UI/Server pair

- UI Function (**ui.R**)
- Server Function (**server.R**)
- If you are building both components in same file, then (**app.R**)

STRUCTURE OF A SHINY APP

- Template

```
# load shiny library
library(shiny)
Ui <- fluidpage()
# Piece of code here
Server <- function(input, output) {
# Piece of code here
}
shinyApp(ui = ui, server = Server)
```

BASIC EXAMPLE

- Template

```
# load shiny library
library(shiny)
Ui <- fluidpage("Hello World")
# Piece of code here
Server <- function(input, output) {
# Piece of code here
   session$onSessionEnded(stopApp)
}
shinyApp(ui = ui, server = Server)
```

ADDING MORE TO BASIC EXAMPLE

- Using the same template

```
library(shiny)
UI <- fluidPage(
  titlePanel(title = "Demo for Shiny"),
  sliderInput(inputId = "num", label = "Number of Observations",
              min = 1, max = 200, value = 50),
  plotOutput("hist")
)
Server <- function(input, output, session){
  output$hist <- renderPlot({
    title <- "Hist for 100 numbers"
    hist(rnorm(input$num), main = title)
    })
  session$onSessionEnded(stopApp)
}
shinyApp(ui = UI, server = Server)
```

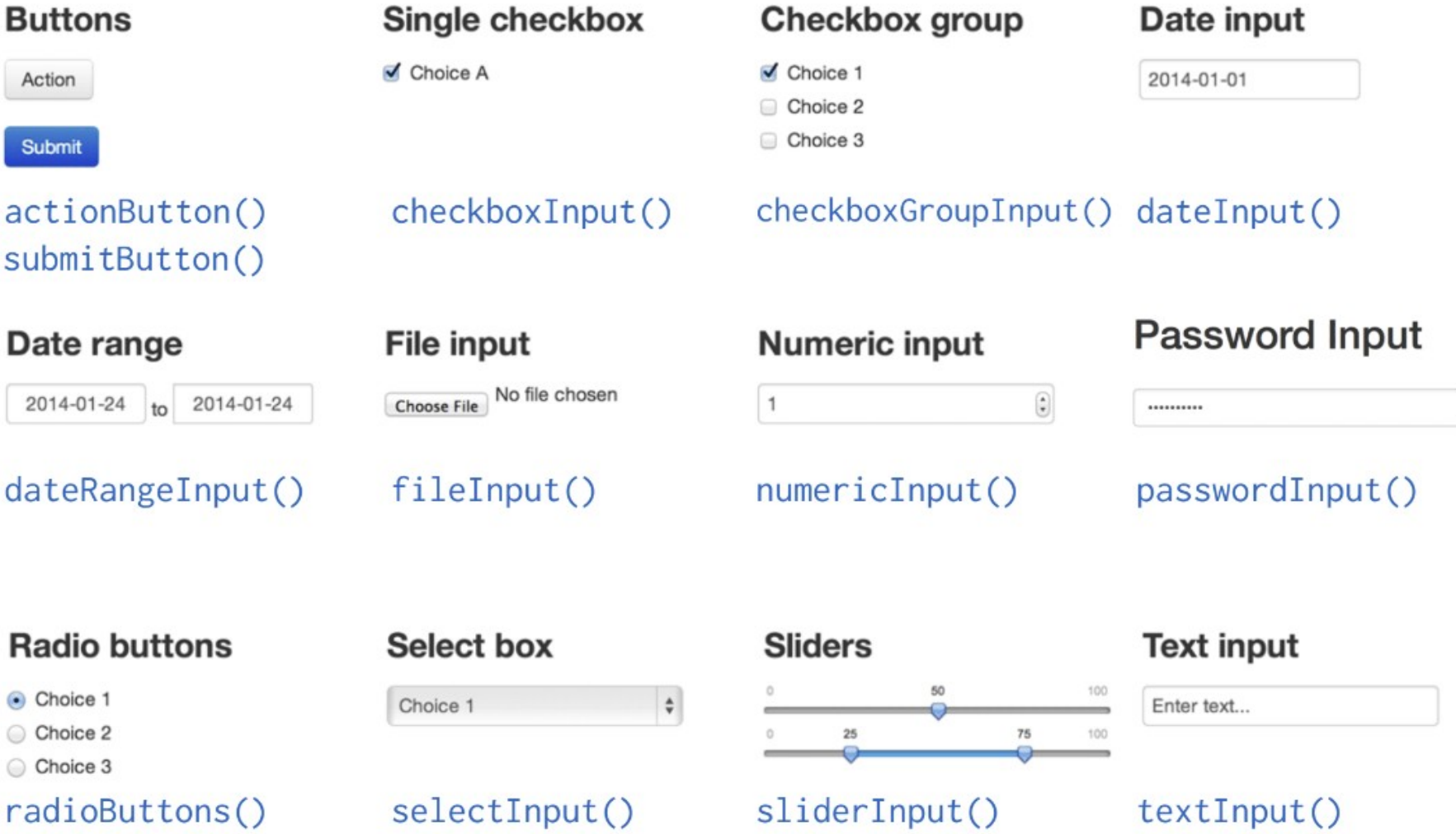
INPUTS

```
library(shiny)
UI <- fluidPage(
  titlePanel(title = "Demo for Shiny"),
  sliderInput(inputId = "num", label = "Number of Observations",
              min = 1, max = 200, value = 50),
  plotOutput("hist")
```

- To create inputs, use *Input() function

INPUT FUNCTIONS

- Input functions: These are used in fluidPage function in UI component



OUTPUTS

```
library(shiny)
UI <- fluidPage(
  titlePanel(title = "Demo for Shiny"),
  sliderInput(inputId = "num", label = "Number of Observations",
              min = 1, max = 200, value = 50),
  plotOutput("hist")
```

- To display output on UI, add it to fluidPage() with an Output() function
- Output() – adds space on UI for R object
- From plotOutput("hist"), we have to give "hist" as a name to output object

OUTPUT FUNCTIONS

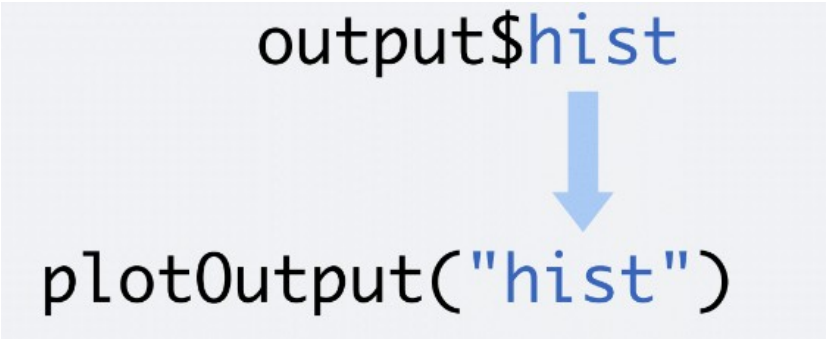
- Output functions: According to this function, output is shown on UI

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

SERVER CODE: ASSEMBLING INPUTS AND OUTPUTS

- Three basic rules to write server function.

```
Server <- function(input, output, session){
    # Some piece of code here
    # Some piece of code here
}
```



- Rule 1: Save objects that has to be displayed on UI to output$

```
Server <- function(input, output, session){
    output$hist <- code
    # Some piece of code here
}
```

SERVER CODE: ASSEMBLING INPUTS AND OUTPUTS

- Rule 2: Build objects that has to be display with render*().
- For Building, you need
  - Kind of object to build
  - Code to build that object


```
Server <- function(input, output, session){
    output$hist <- renderPlot({hist(rnorm(100))
    })
    # Some piece of code here
}
```

SERVER CODE: ASSEMBLING INPUTS AND OUTPUTS

- The render*() functions used to render different types of outputs on Webpage

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

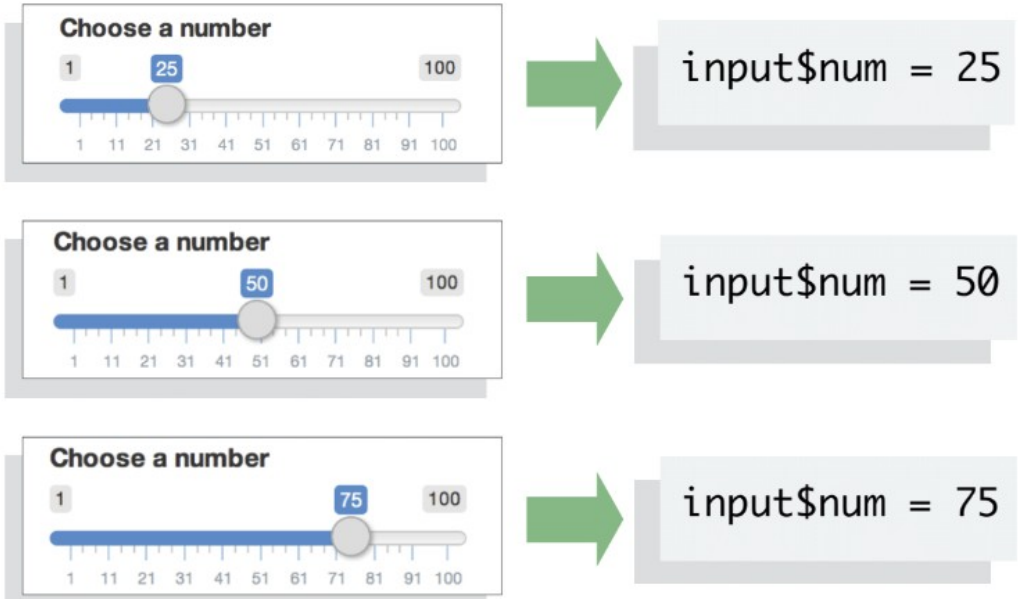
SERVER CODE: ASSEMBLING INPUTS AND OUTPUTS

- Rule 3: Access input values with input$

```
Server <- function(input, output, session){
    output$hist <- renderPlot({hist(input$num))
    })
    # Some piece of code here
}
```

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

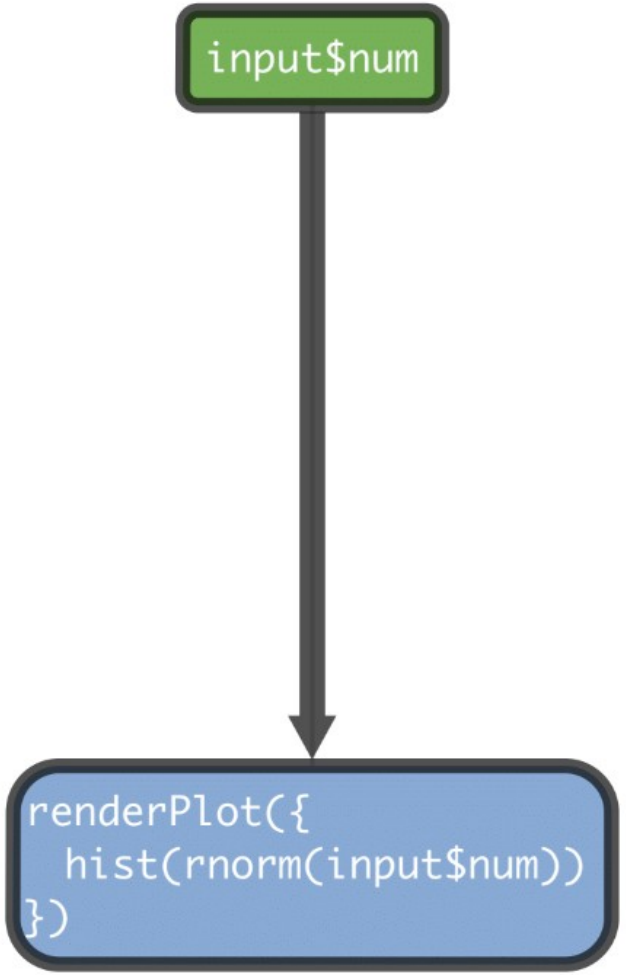
```
input$num
```



INTERACTIVE

- Interaction happens when input value is rendered to output object

```
Server <- function(input, output, session){
    output$hist <- renderPlot({hist(input$num))
    })
    # Some piece of code here
}
```



RECAP

Use the server function to assemble inputs into outputs. Follow 3 rules:

```
output$hist <-
```

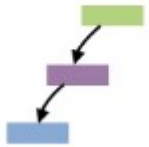
1. Save the output that you build to **output$**

```
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**

PUBLISH YOUR APPLICATION

- Publish to Shinyappsio.
- Include app.r, datasets, images, css, helper scripts etc.

```
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))
  })
}
```

Reference

- https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/

THANK YOU