Introduction to Shiny

Justin Millar
March 28, 2018

What is Shiny?

Shiny is an R package that can create interactive applications straight from R code

Shiny applications hosted and accessed online

- Free Shiny app hosting at www.shinyapps.io
- Shiny code can also be hosted on GitHub and run locally in the R console using the `shiny::runGitHub()` function

No HTML/CSS/JavaScript required!

- Everything is written in the same R code you already know and love

Some of our lab's uses for Shiny

Organizing R outputs (i.e. data tables and plots)

Creating decision tools for aiding policy design

Making interactive maps/plots

Teaching aids

Creating searchable data tables

Check out www.showmeshiny.com for some serious impressive Shiny apps!

The basics of a Shiny app

There are two main components of a Shiny app

- The `ui`, or user interface, which designs the layout of the app
- The `server`, which operates your R code

There are two approaches for organizing Shiny app

- Using one script, `app.R`, which contains two variables, `ui` and `server`, for defining these components and ends with the `shinyApp(ui, server)` function
- Using two separate scripts, `ui.R` and `server.R`, for defining the user interface and server operations

The app can be run using the `runApp()` function, or with the “Run App” button in RStudio

The UI

Characteristics of the UI

- Designing the layout of the app
  - Explanatory text
  - Panels/tabs
  - Fancy stuff (e.g. Bootstrap themes)
- Defining the Input variables
  - Variable type/structure
  - Restrictions on inputs
  - Naming variables for the server
- Displaying the Output from the server
  - Primarily where each output will be displayed/organized

```
# 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")

    )
  )
)
```

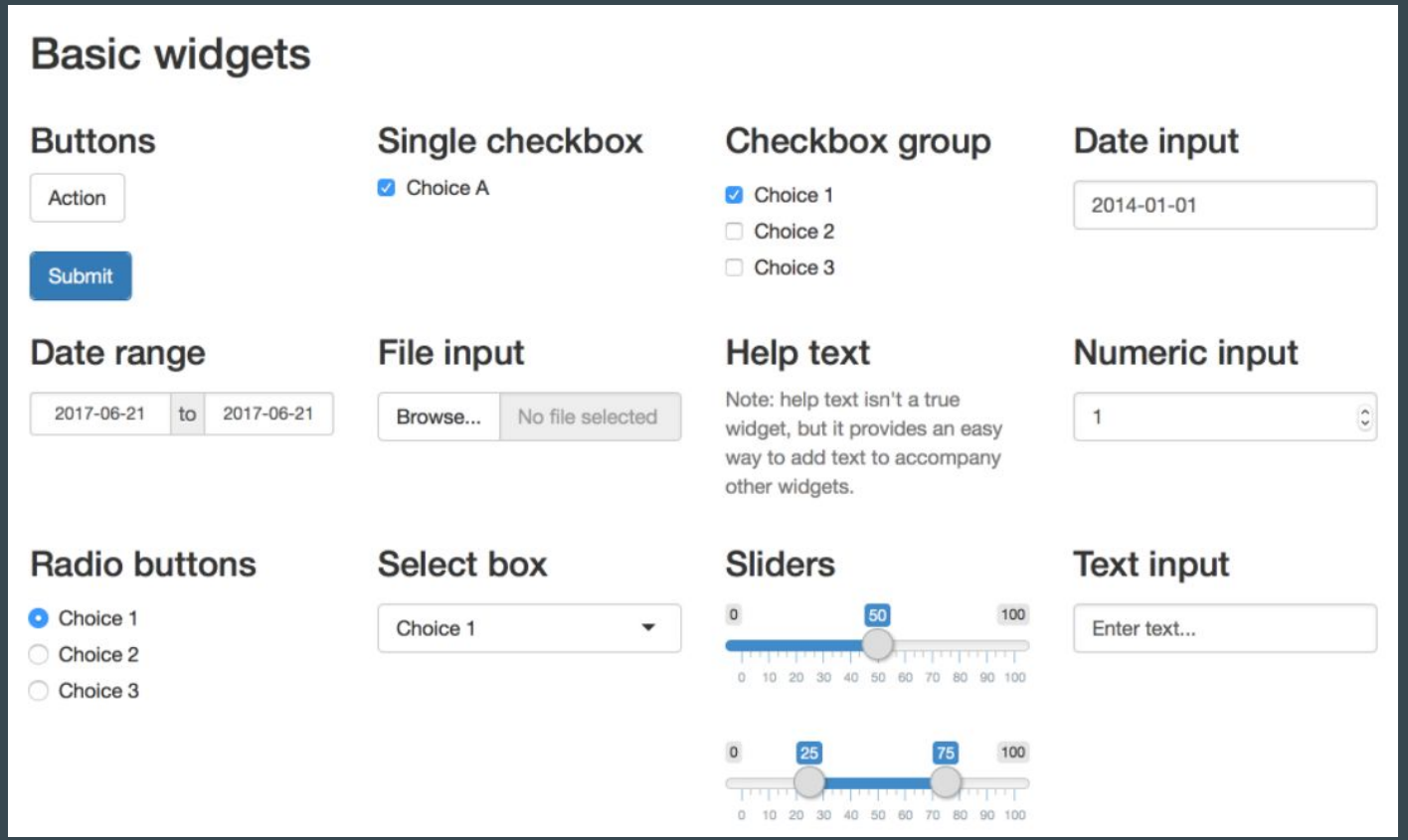
The Server

Notice that the `server` variable is a function with the arguments `(input, output)`

- The server will define each output with unique variable names that can be called in the UI
  - Defined as `output$varName`
- We need to use specific functions for different types of output
  - `renderPlot()`, `renderDataTable()`
- These functions can take inputs from the UI using `input$inputId`
- The `{}` in `renderPlot()` tell the server that this object is **reactive**, and should automatically update whenever the `input$var` is changed

```
# 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")

    })

}
```

Control widgets



Check out the gallery

Reactivity and Reactive Expressions

Reactivity is what allows your Shiny app to update based on new information from the UI

The `render` functions in the server automatically track which `input$` variables are included for each `output$`, and update accordingly

Think of the `output$...` as a list of different objects you want to create with R, created by a distinct `render` function

We can create individual a reactive expression using the `reactive({})` function

Reactive expressions take widget input(s) and return a value, and will update whenever the input value changes

These variables can also used in server outputs

Reactive expression cache information, which can significantly speed up computation for your app

Think of reactive expressions as a chain that connects `input` values to `output` objects

Click here for a better, in-depth explanation of reactivity

Where should I write my code?

Various components of your R code may be:

- Generally applicable to the entire app
- Only involved in the `server` section
- Used in a specific `output$...`

There are different places within your script that you can put in R code

So where should it all go?

```
# a place to put code

ui <- fluidpage(

)

server <- function(input, output) {

  # Another place to put code

  output$map <- renderPlot({

    # A third place to put code

  })
}

shinyApp(ui, server)
```

```
# A place to put code

ui <- fluidpage(

)

server <- function(input, output) {

  # Another place to put code

  output$map <- renderPlot({

    # A third place to put code

  })
}

shinyApp(ui, server)
```

Run once when app is launched

```
# A place to put code

ui <- fluidpage(

)

server <- function(input, output) {

  # Another place to put code

  output$map <- renderPlot({

    # A third place to put code

  })
}

shinyApp(ui, server)
```

Run once each time a user visits the app

```
# A place to put code

ui <- fluidpage(

)

server <- function(input, output) {

  # Another place to put code

  output$map <- renderPlot({

    # A third place to put code

  })
}

shinyApp(ui, server)
```

Run once each time a user changes a widget that output$map depends on

Further Resources

https://shiny.rstudio.com/tutorial/

- Test out the included examples ->

http://shiny.rstudio.com/gallery/

https://www.showmeshiny.com/

https://deanattali.com/blog/building-shiny-apps-tutorial/

```
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
```

Link to GitHub with demonstration examples