SHINY INTRO

Check out

https://shiny.rstudio.com

for additional information and tutorials

Contents

- Building Basic Shiny Apps
- Sharing Shiny Apps
- Customize Reactions

Building a Basic Shiny App

- Start up RStudio
- Create a new R script (via File – New File – R Script).
- Save this new R script and give it a name, like “test”. RStudio will then call it “test.R”.
- Enter the following lines (also available in shinyApp1.R):

```
library(shiny)
ui <- fluidPage("Hello World!")
server <- function(input, output){}
```

- Finally, enter this line:

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

- Note that, if you save the file after typing the last line, some things change in RStudio. For example, instead of “Run”, you now see “Run App”.
- Because of that last line, you can now no longer run a selection of your code; you can only run the app in its totality.
- I recommend that during development you put all your code in a separate R script, leaving just the library statement and the shinyApp statement in your Shiny App.

Creating a Basic Shiny App - 2

- Click on “Run App”.
- Also try “Open in Browser”.
- Exit out or press the “Stop” button in your RStudio console.

- You have created your first Shiny App!

Format of a Shiny App

- UI – The user interface, contains all the input and output fields (via input/output functions) and their position on the screen (in html).

- Server – Function that populates the output fields using the input fields

- shinyApp – Function that puts it all together

Try: type “UI” in your console.

Important Functions

Inputs: actionButton(),submitButton(),checkboxInput(), checkboxGroupInput(),dateInput(), dateRangeInput(), fileInput(),numericInput(),passwordInput(),radioButtons(), selectInput(), sliderInput(),textInput()

Outputs: dataTableOutput(), htmlOutput(), imageOutput(), plotOutput(), tableOutput(), textOutput(), uiOutput(), verbatimTextOutput()

Renders: renderDataTable(), renderImage(), renderPlot(), renderPrint(), renderTable(), renderText(), renderUI()

A More Interesting App

```
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))
  })
}
```

Run this and see what it does.

Available in shinyApp2.R

What do These Important Functions Do?

- Input and output functions create a field on the screen. Input fields can be seen immediately; type the function with a question mark in front of it in the console to see how to create this type of field.
- Example: `?sliderInput`
- The first argument to the input/output functions is the name of the field. When referring to a field, use its name with respectively input$ or output$ in front of it.
- Example: `numericInput("num","Number",2)` creates an input field called input$num with label “Number” and initial value 2.
- Example: `plotOutput("hist")` creates space for an output field called output$hist, which will contain a plot.

What do These Important Functions Do? - 2

- Output fields need to be populated using the “render” functions. This is done inside the “server” function.

  Example:

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

  This will create a histogram using a number of random normally distributed values; the number used is obtained from the “input$num” field.

NOTE: The plotOutput() function in the “ui” field only creates the ***space*** for an output field; it must be populated inside the “server” function.

What do These Important Functions Do? - 3

- “renderPlot” is a function whose argument is a block of code. The output of this block of code should be a plot. Aside from this requirement, in this block you can do anything you can do in other R functions; populate local variables, call other functions, loops, etc., as long as its output (i.e. the last statement) is a plot.
- The same concept holds for all other render functions.
- Render functions will run every time an input field that it uses, is changed.
- In our “more interesting example”, renderPlot will run each time input$num changes value . We say that renderPlot ***reacts*** to changes in input$num.
- NOTE that input values can only be accessed in special reactive functions (like the render functions).

Sharing Your App

- Create a directory on your computer and put all files related to this app in this directory.
- Call the main R-script (the one that contains `shinyApp(ui = ui, server = server)`) "app.R".
- Sign up for an account at shinyapps.io. Go to your username, then "tokens" and "show". Then click "Copy to clipboard" and "ok".
- In RStudio, type

  ```
  install.packages("rsconnect")
  library(rsconnect)
  ```

  followed by what is on your clipboard, which should be something like this

  ```
  rsconnect::setAccountInfo(name='name',token='token',
                            secret='secret')
  ```

  finally:

  ```
  rsconnect::deployApp('directory path')
  ```

- Now when you run your app in Rstudio, you see a "publish" option in the top right corner. Choosing this will publish your app to shinyapps.io.
- Find the URL for your app via your account; click Applications – running or sleeping – name (of your directory). You can then run your app via that URL.

Customize Reactions

Reactive functions: All render functions (as before), plus reactive(), isolate(), observeEvent(), observe(), eventReactive(), reactiveValues()

- Every reactive function takes a block of code (enclosed in curly braces) as input. This block of code will run every time one or more input values change.
- isolate() is actually a non-reactive function. It takes a reactive input and stops it from being reactive.

Another Shiny App

```
ui <- fluidPage(
  sliderInput(inputId="num",
    label="Choose a number",
    value=25,min=1,max=1000),
  textInput(inputId= "title",
    label ="Write a title",
    value="Random Normal Values"),
  actionButton(inputId="go",
    label= "Update"),
plotOutput("hist"),
  verbatimTextOutput("stats")
)
```

```
server <- function(input, output){
  data<-eventReactive(input$go,
                {rnorm(input$num)})
  output$hist<-renderPlot({
    title<-paste(isolate(input$num),
            isolate({input$title}))
    hist(data(), main=title)
  })
  output$stats<-renderPrint({
    summary(data())
  })
}
```

Available in shinyApp3.R

Customize Reactions - 2

- reactive () is used to avoid running the same code multiple times. It is assigned to a variable.
- Since the variable is a function, it must be called as a function. So use parentheses!
- Example: `data<-reactive({rnorm(input$num)})`
- Render functions can now call `data()` instead of `rnorm(input$num)`, and if multiple render functions are involved, they will all use the same data (which is desirable, since rnorm gives different results each time it is run).

Customize Reactions - 3

- isolate () is used if we do not want the code block to run when a certain input field is changed.
- Example:

```
output$hist<-renderPlot({
    title<-paste(isolate(input$num),
                 isolate({input$title}))
    hist(data(), main=title)
})
```

- This block of code will not run if input$num and/or input$title are changed. Only when the data() function output changes will this code run (and then it will use the latest values of input$num and input$title).

Customize Reactions - 4

- observe() will run any time one of the input values in its block of code is changed. Unlike reactive(), it is not assigned to a variable so there is no output value that can be accessed in other functions.
- observeEvent () is like observe(), but is used if we want to run its block of code only when certain input values change; it takes TWO arguments. The first contains the input values it will react to, and the second one is the code block.

Customize Reactions - 5

- eventReactive() is used if we want to avoid running the same code multiple times, and also only want to run it when certain input fields have changed. Like observeEvent() it takes two arguments; the first contains the input values it will react to, and the second one is the code block. However, unlike observeEvent() and like reactive(), its value is assigned to a variable which can be accessed elsewhere in the code.
- Example:

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

- This block of code will only run when input$go changes (in this case it is a button that can be pressed). If input$go changes, rnorm is run using the latest value of input$num. Its output will be available as data() elsewhere in the code.

Customize Reactions - 6

- reactiveValues() is used if you need to create a list of variables that are updated using code, and should trigger updates elsewhere when changed.
- Example:

```
rv<-reactiveValues(num=5)
```

- The field rv$num can now be accessed anywhere in the server code. For example, it may be used in one of the render functions. But now when somewhere in the code this value is changed, for example via `rv$num<-6`, the code block of the render function that uses it, will be rerun.