615 Shiny HW_code

Xiaohan Shi

2024-11-16

##Question 1: What is the difference between Hadley 1 and Hadley 2?

Answer: Both versions demonstrate how to build a basic Shiny application and gradually optimize the logical organization of your code. Hadley_1 is a simply version that let users select a dataset (the datasets package from R) and display the summary statistics and raw data table for that dataset. Hadley_2 is a updated version that allows users to separate UI and server logic, making code cleaner and easier to extend

##Exercise 2.3.5

1

Which of textOutput() and verbatimTextOutput() should each of the following render functions be paired with?

renderPrint(summary(mtcars)): verbatimTextOutput()

renderText("Good morning!"): textOutput()

renderPrint(t.test(1:5, 2:6)): verbatimTextOutput()

renderText(str(lm(mpg ~ wt, data = mtcars))): textOutput()

2

```
# Define UI
ui <- fluidPage(
  # Add an accessible description for the plot
  tags$p("Scatterplot of five random numbers for visually impaired users."),
  plotOutput(
    "plot",
    width = "700px",
    height = "300px"
  )
)

# Define Server
server <- function(input, output, session) {
  output$plot <- renderPlot({
    plot(1:5, main = "Scatterplot of Five Random Numbers", xlab = "Index", ylab = "Value")
  }, res = 96)
}

shinyApp(ui, server)
```

3

```
ui <- fluidPage(
  DTOutput("table")
)

server <- function(input, output, session) {
  output$table <- renderDataTable(
    mtcars,
    options = list(
      pageLength = 5,
      searching = FALSE,
      ordering = FALSE,
      paging = TRUE,
      info = FALSE
    )
  )
}
shinyApp(ui, server)
```

4

```
ui <- fluidPage(
  reactableOutput("table") # Use reactableOutput for Reactable tables
)

server <- function(input, output, session) {
  output$table <- renderReactable({
    reactable(
      mtcars,
      pagination = TRUE,
      searchable = FALSE,
      sortable = FALSE
    )
  })
}
shinyApp(ui, server)
```

3.3.6

1

```
ui <- fluidPage(
  textInput("name", "What's your name?"),
  textOutput("greeting")
)
server <- function(input, output, session) {
  output$greeting <- renderText({
    paste0("Hello ", input$name)
  })
}
shinyApp(ui, server)
```

2

1.

```
server1 <- function(input, output, session) {
  c <- reactive(input$a + input$b)
  e <- reactive(c() + input$d)
  output$f <- renderText(e())
}

Graph:

input$a ─┐
         │
input$b ─┼─> c() ──> e() ──> output$f
         │
input$d ─┘
```

2.

```
server2 <- function(input, output, session) {
  x <- reactive(input$x1 + input$x2 + input$x3)
  y <- reactive(input$y1 + input$y2)
  output$z <- renderText(x() / y())
}

Graph:
input$x1 ─┐
input$x2 ─┼─> x() ──┐
input$x3 ─┘          │
                     ├─> output$z
input$y1 ─┐          │
input$y2 ─┼─> y() ──┘
```

3.

```
server3 <- function(input, output, session) {
  d <- reactive(c() ^ input$d)
  a <- reactive(input$a * 10)
  c <- reactive(b() / input$c)
  b <- reactive(a() + input$b)
}

input$a ─┐
         │
         └─> a() ──┐
                   ├─> b() ──┐
input$b ──────────────┘       │
                              ├─> c() ──> d()
input$c ──────────────────────────┘       │
                                         │
input$d ─────────────────────────────────────────┘
```

3

choose unique names for your reactive expressions that don’t conflict with base R functions. For example:

```
my_var <- reactive(df[[input$var]])
my_range <- reactive(range(my_var(), na.rm = TRUE))
```

Now ‘my_var()’ and ‘my_range()’ are custom names for your reactive expressions, avoiding the conflict with the base functions var() and range().