Introduction

CASE STUDIES: BUILDING WEB APPLICATIONS WITH SHINY IN R

Dean Attali
Shiny Consultant

datacamp

Course overview

- Assumes basic Shiny knowledge
- Review important Shiny concepts
- Develop multiple apps for real-life scenarios
- Repetitive practice of essential features to increase familiarity
- Learn new features and best practices

Shiny app template

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

- Load the shiny package
- Create a webpage with `fluidPage()` - UI of a Shiny app
- Create server portion of the app - where application logic lives
- Combine UI + server into a Shiny app and run it

Adding text to Shiny

- Add text as argument to `fluidPage()`

```
ui <- fluidPage(
    "Hello there"
)
```

- Result: webpage with text "Hello there"
- `fluidPage()` accepts arbitrary # of arguments

```
ui <- fluidPage(
    "Hello",
    "there"
)
```

Formatted text

`h1()`

`h2()`

`strong()`

`em()`

Primary Header

Secondary header

Bold

Italicized (*em*phasized)

Formatted text

```
ui <- fluidPage(
    h1("SHINY COURSE"),
    "by",
    strong("Dean Attali"),
)
```

SHINY COURSE

by **Dean Attali**

Sidebar layout

Sidebar layout

Sidebar layout

Sidebar Layout


```
ui <- fluidPage(
    sidebarLayout(
        sidebarPanel(
            "This is the sidebar"
        ),
        mainPanel(
            "Main panel goes here"
        )
    )
)
```

Let's practice!

CASE STUDIES: BUILDING WEB APPLICATIONS WITH SHINY IN R

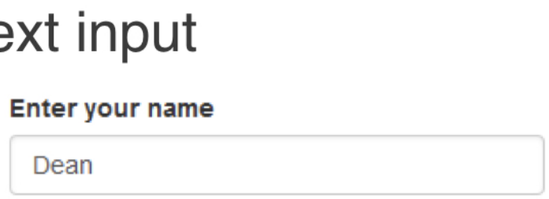
Inputs and outputs

CASE STUDIES: BUILDING WEB APPLICATIONS WITH SHINY IN R

Dean Attali
Shiny Consultant

datacamp

Inputs



Inputs

Text input

Enter your name

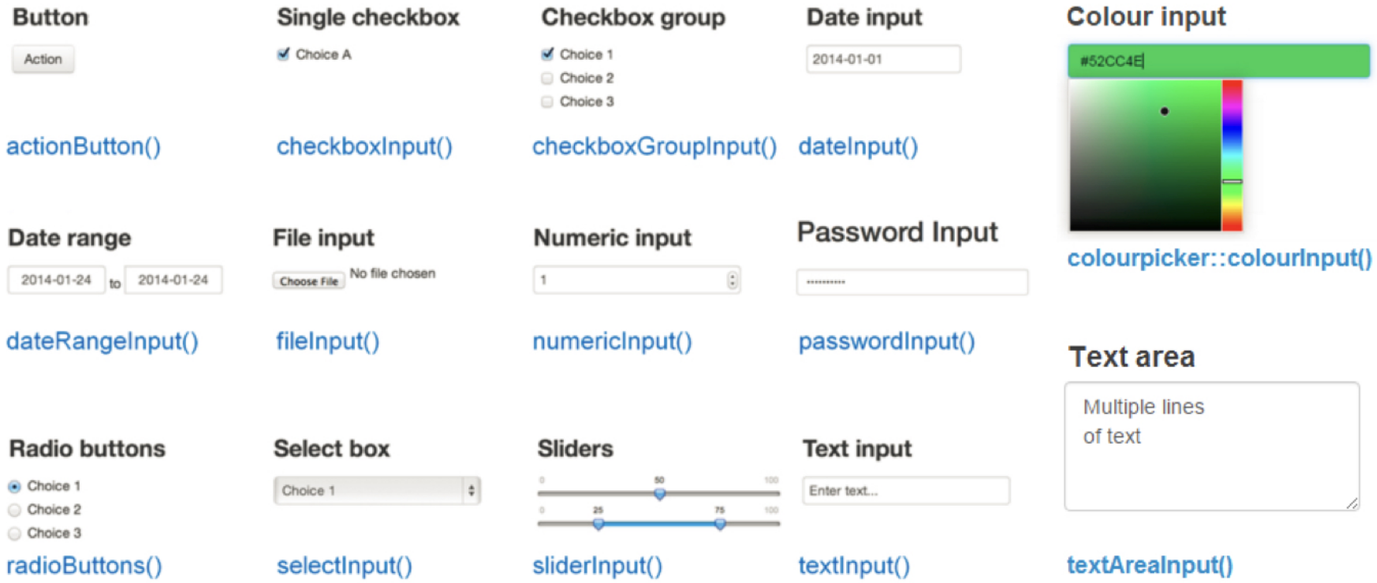
Dean

Numeric input

How many siblings?

4

Inputs


Button
Action
actionButton()
Single checkbox
Choice A
checkboxInput()
Checkbox group
Choice 1
Choice 2
Choice 3
checkboxGroupInput()
Date input
2014-01-01
dateInput()
Colour input
#52CC4E
colourpicker::colourInput()
Date range
2014-01-24
to
2014-01-24
dateRangeInput()
File input
Choose File
No file chosen
fileInput()
Numeric input
1
numericInput()
Password Input
passwordInput()
Text area
Multiple lines
of text
textAreaInput()
Radio buttons
Choice 1
Choice 2
Choice 3
radioButtons()
Select box
Choice 1
selectInput()
Sliders
sliderInput()
Text input
Enter text...
textInput()

Building inputs

```
ui <- fluidPage(
    textInput(inputId = "name", label = "Enter your name",
              value = "Dean"),
    numericInput(inputId = "sibs", label = "How many siblings?",
                 value = 4, min = 0)
)
```

- Input functions: `*Input(inputId, label, ...)`
- `inputId` = Unique ID
- `label` = Text to describe input
- `...` = Additional input-specific parameters

Outputs

- Plots, tables, text - anything R creates & users see
- Two steps:
  1. Create placeholder for output (in UI)

```
ui <- fluidPage(
 "Plot goes here:",
 plotOutput(outputId = "my_plot")
)
```

  2. Write R code to generate output (in server)

The server

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

  # Code for building outputs

}
```

- `input`
  - Read values from here (inputs user modifies)
- `output`
  - Write values to here (outputs e.g. plots, tables)

Building outputs

```
ui <- fluidPage(
    numericInput("num", "Number of rows", value = 10, min = 0),
    tableOutput("my_table")
)
server <- function(input, output) {
    output$my_table <- renderTable({
        head(iris, n = input$num)
    })
}
```

- 3 Rules to build output object:
  1. Build object inside render function (`renderPlot()` , `renderText()` , etc)
  2. Save object to `output$<outputId>`
  3. Use `input$<inputId>` to access value of input

Let's practice!

CASE STUDIES: BUILDING WEB APPLICATIONS WITH SHINY IN R

Reactivity 101

CASE STUDIES: BUILDING WEB APPLICATIONS WITH SHINY IN R

Dean Attali
Shiny Consultant

datacamp

Reactivity basics

- Shiny uses **reactive programming**
- Outputs **react** to changes in input
- When value of variable `x` changes, anything that relies on `x` is re-evaluated
- Contrast with regular R:

```
x <- 5
y <- x + 1
x <- 10
```

- What is the value of y? 6 or 11?

Reactive variables

- All inputs are reactive
- `input$<inputId>` inside render function will cause output to re-render

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

- `output$my_plot` depends on `input$num`
  - `input$num` changes ⇒ `output$my_plot` reacts

Reactive contexts

- Reactive values can only be used inside **reactive contexts**
- Any `render*()` function is a reactive context
- Accessing reactive value outside of reactive context ⇒ error

```
server <- function(input, output) {
    print(input$num)
}
```

```
ERROR: Operation not allowed without an active reactive context.
```

Observe a reactive variable

- `observe({ ... })` to access reactive variable

```
server <- function(input, output) {
    observe({
        print( input$num )
    })
}
```

- Useful for debugging, track reactive variable
- Each reactive variable creates a dependency

```
observe({
    print( input$num1 )
    print( input$num2 )
})
```

Create a reactive variable

- `reactive({ ... })` to create reactive variable
- Wrong:

```
server <- function(input, output) {
    x <- input$num + 1
}
```

```
ERROR: Operation not allowed without an active reactive context.
```

- Correct:

```
server <- function(input, output) {
  x <- reactive({
      input$num + 1
  })
}
```

Reactive variables

- Access custom reactive variable like a function:
  - add parentheses `()`

```
server <- function(input, output){
    x <- reactive({
        input$num + 1
    })
    observe({
        print( input$num )
        print( x() )
    })
}
```

Let's practice!

CASE STUDIES: BUILDING WEB APPLICATIONS WITH SHINY IN R