Creating web apps for interactive data visualization with R Shiny

A Tufts TIDAL workshop from the Data Intensive Studies Center (DISC)

Karin Knudson

karin.knudson@tufts.edu

https://karink520.github.io/intro-r-shiny-workshop/

Creating web apps for interactive data visualization with R Shiny

A Tufts TIDAL workshop from the Data Intensive Studies Center (DISC)

Karin Knudson

karin.knudson@tufts.edu

https://karink520.github.io/intro-r-shiny-workshop/

If you're here early, check out examples of Shiny apps at: https://shiny.rstudio.com/gallery/

“Shiny makes it **incredibly easy** to build **interactive web applications** with R. Automatic **"reactive" binding** between inputs and outputs and extensive **prebuilt widgets** make it possible to build **beautiful**, **responsive**, and **powerful applications with minimal effort**.”

— the Shiny documentation

TO DO:

1. Open the workshop page and keep the materials handy throughout the session:

   https://karink520.github.io/intro-r-shiny-workshop/

2. Open RStudio and install Shiny using the command:

```
install.packages(“shiny”)
```

STRUCTURE OF A SHINY APP

```
library(shiny)

ui <- ... # Appearance and layout of app

server <- ... # Code the server needs to run,
          e.g. draw plots and process data

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

A BASIC TEMPLATE

```
ui <- fluidPage(

)

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

}

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

LAYOUT AND PANEL FUNCTIONS TO STRUCTURE

- Content as arguments, separated by commas
- Start with `fluidPage()`, `navbarPage()`, or `fixedPage()`
- See `fluidRow()/fluidPage()` and `column()` docs for more flexibility laying out page on a 12-column grid
- Bootstrap framework

CREATE HTML WITH SHINY

Many html tags have their own Shiny helper function, e.g.:

`a, div, h1…h6, img, p, span`

HTML tags can also be created with `tags`, e.g.:

`tags$a, tags$code`

Some HTML tags have names that would conflict with native R functions so must be accessed within `tags`, rather than with their name alone

CUSTOMIZING CSS

Put a CSS file in `www` subdirectory (contents sent to user’s browser)

```
ui = fluidPage( includeCSS(”stylesheet.css"),
      …
```

ADDING A THEME

```
install.packages("shinythemes")
```

Add a theme argument to fluidPage, e.g:

```
    ui = fluidPage(theme = shinytheme("darkly"),
          …
```

More about themes: https://rstudio.github.io/shinythemes/

BASIC REACTIVITY WITH *INPUT, *OUTPUT, RENDER*

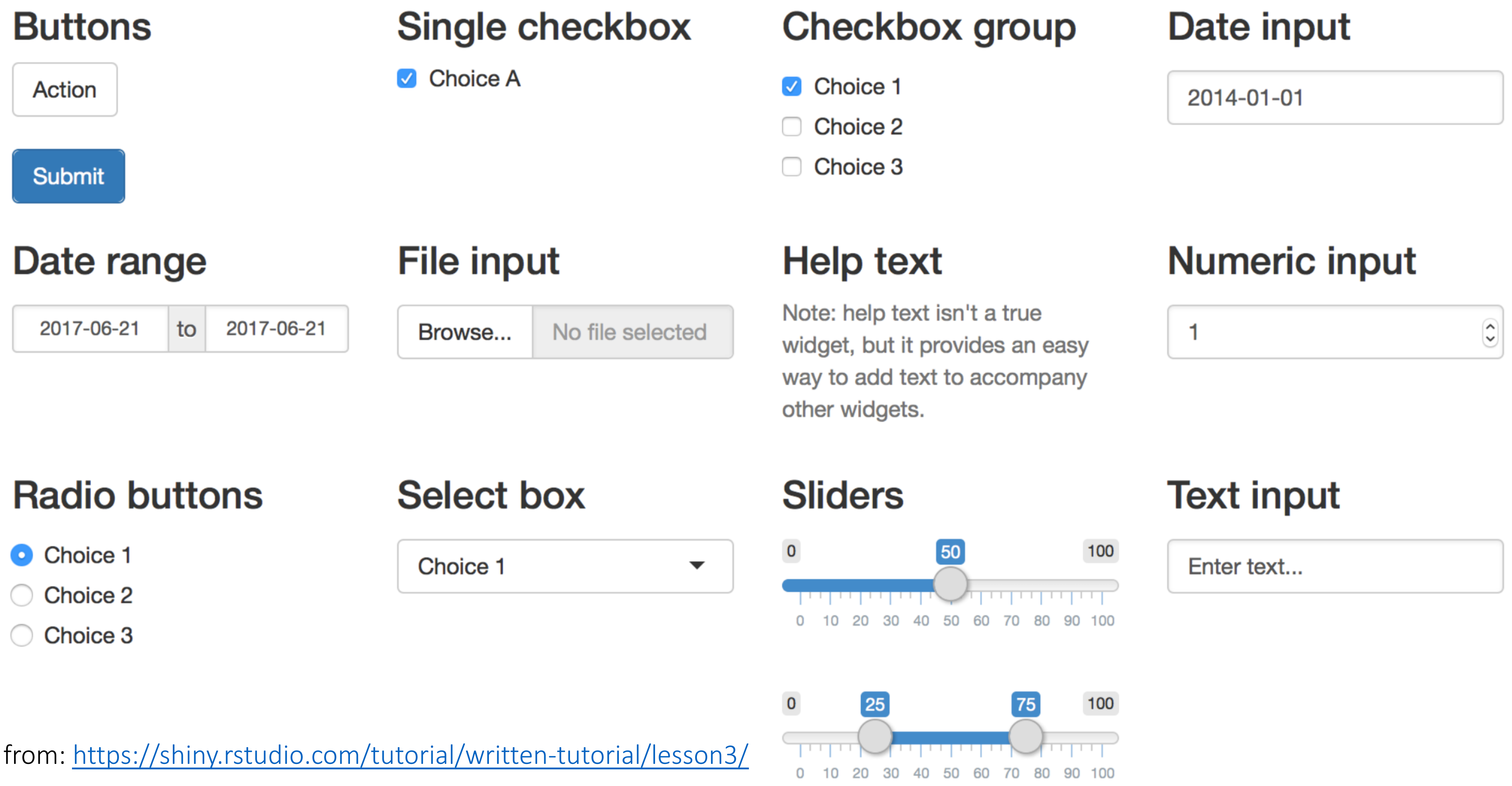
Add inputs (e.g. premade widgets) and outputs to `ui`, each with an id

Inputs and outputs are passed to the server function in a list, accessed by their ids

Connect inputs and outputs with `render*` in the server function

When an input used within `render*` changes, code block re-runs.

WIDGET EXAMPLES



Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

image from: https://shiny.rstudio.com/tutorial/written-tutorial/lesson3/

YOU TRY (EXERCISES)

QUESTIONS, RECAP, PAUSE

NEXT: A DEEPER DIVE INTO REACTIVITY

Including additional functions for more control over reactive behavior

REACTIVITY

- Track dependencies
- Update output when (ideally **only** when!) values that it depends on change
- Reactive code runs in an order depending on the *reactive graph,* the structure of how inputs and outputs are connected

REACTIVITY

- Elements of `input` are *reactive values* created by `*Input` functions. Reactive values get used inside reactive functions like `render*`

- When reactive values are changed they *invalidate*, and code that depends on them may re-run, depending on how it is written.

- Recall that code in a render function reruns whenever anything inside it invalidates

REACTIVE()

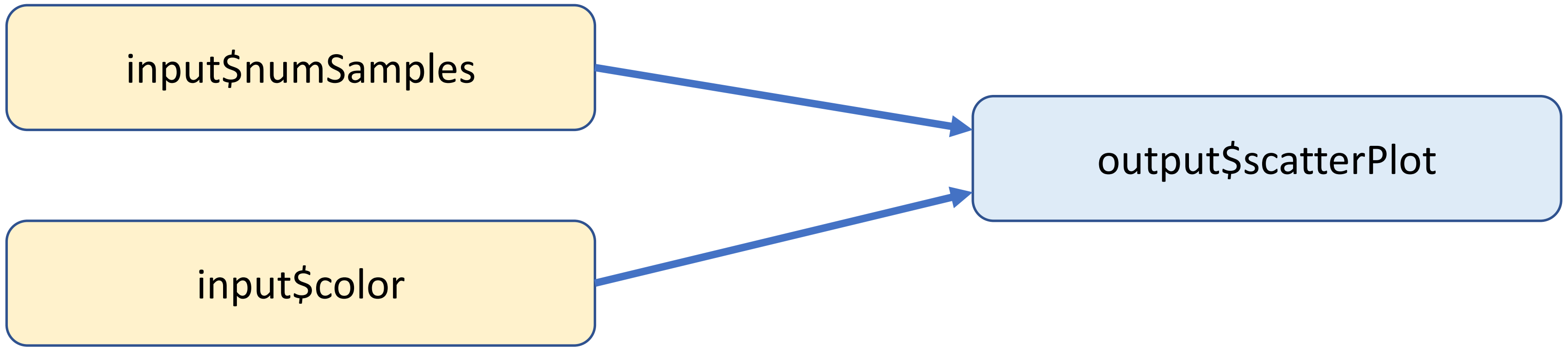
Creates and returns a reactive expression

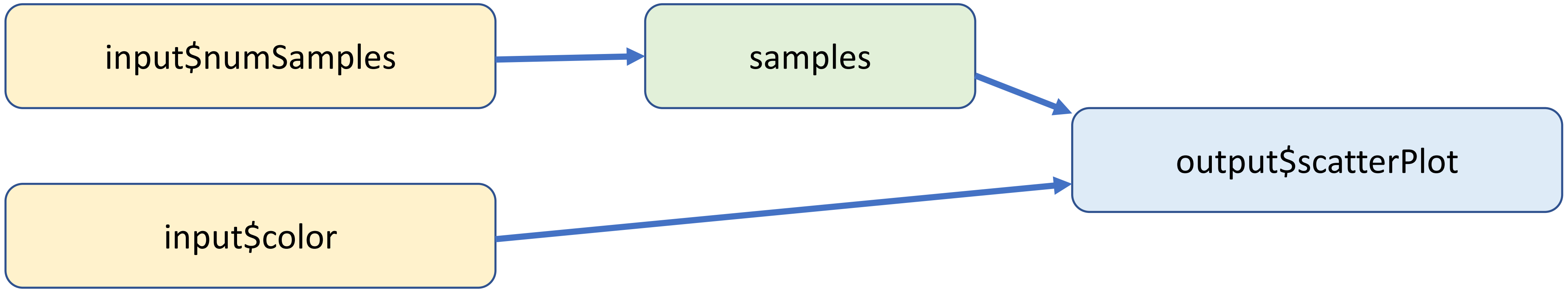
Reactive expression caches its value; reruns if any values it depends on invalidate

Reactive expressions are accessed with function notation - with ():

```
name_of_reactive_expression()
```

Reactive expressions and values must be used inside reactive functions


input$numSamples
input$color
output$scatterPlot


input$numSamples
samples
input$color
output$scatterPlot

EVENTREACTIVE()

```
eventReactive(reactiveValue, {code} )
```

Creates and returns a reactive expression

Only reruns if one of the specified reactive values invalidates

Common use case: delaying events

ISOLATE()

Creates a non-reactive value

	Run some code on the server	Create and return something to display
Rerun when *any* reactive value(s) within it change	observe()	render*()
Rerun when *certain* reactive value(s) within it change	observeEvent()	N/A (but you could use isolate() within render* to prevent invalidation of reactive values inside isolate from making the code in render* rerun.)

	Create and return a reactive expression
Invalidates when *any* reactive values within it invalidate	reactive()
Invalidates when *certain reactive values within it change	eventReactive()
Automatically invalidates after a specified time interval (ms)	reactiveTimer()

Create a non-reactive value
isolate()

Create a list of reactive values
reactiveValues()

DEPLOYING YOUR APP

Shinyapps.io (easiest option, includes free and paid tiers)

Shiny Server (host yourself)

RStudio connect (paid option with more features)

RESOURCES - SHINY

- Tutorials and more: https://shiny.rstudio.com/tutorial/
- Function reference: https://shiny.rstudio.com/reference/shiny/1.4.0/
- “Cheat Sheet”: https://rstudio.com/resources/cheatsheets/
- Book by Hadley Wikham: https://mastering-shiny.org/

RESOURCES – DATA VISUALIZATION

- Fundamentals of Data Visualization by Claus Wilke (general principles, but examples are created in R with ggplot2)

- Data Visualization: A Practical Introduction by Kieran Healy (more explicitly R and ggplot2 focused)