Understanding reactivity

Daniel Kaplan

Dtkaplan

dtkaplan@gmail.com

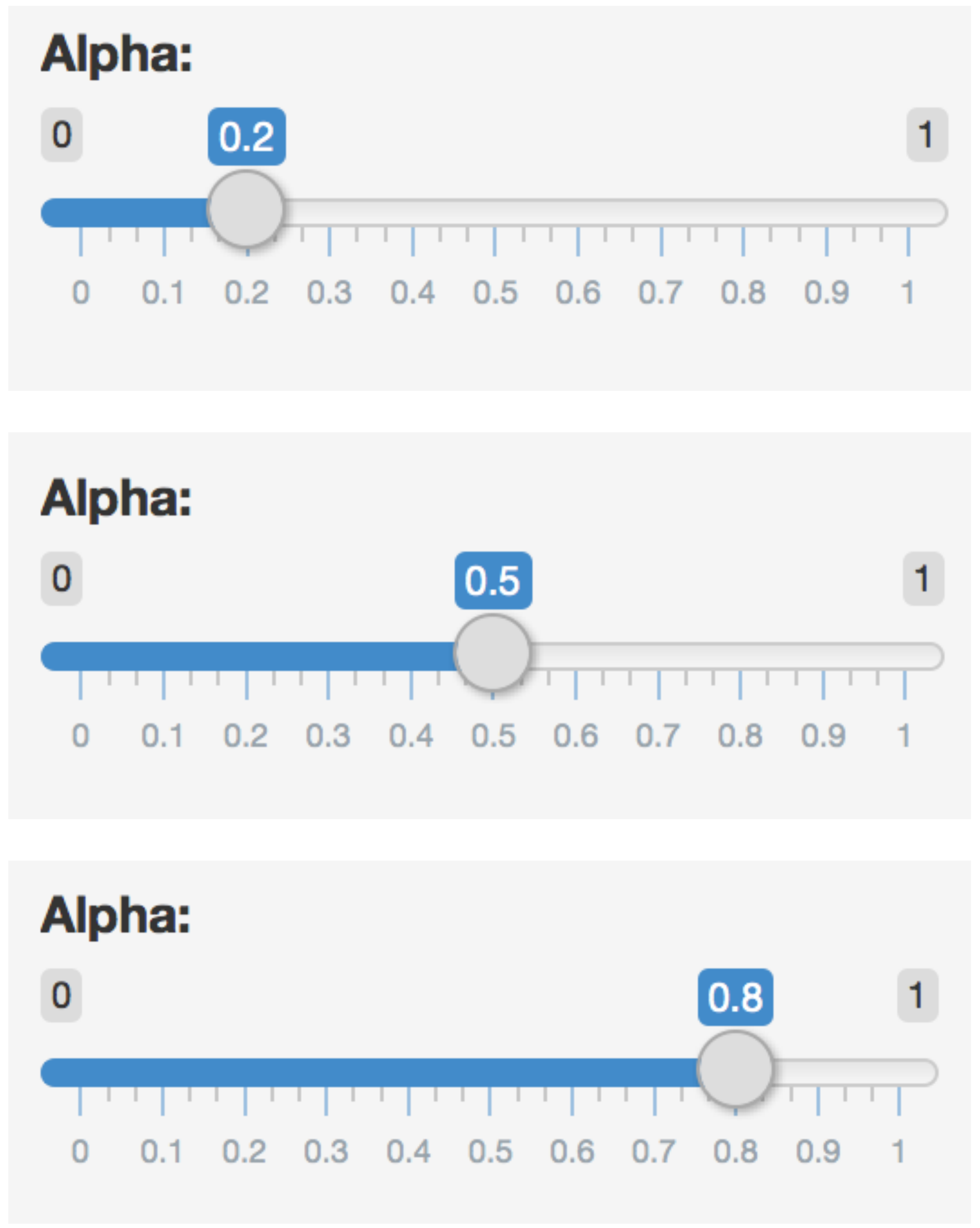
Reactivity 101

Reactions

The **input$** list stores the current value of each input object under its name.

```
# Set alpha level
sliderInput(inputId = "alpha",
         label = "Alpha:",
         min = 0, max = 1,
         value = 0.5)
```

input$alpha



input$alpha = 0.2

input$alpha = 0.5

input$alpha = 0.8

Reactivity 101

Reactivity automatically occurs when an input value is used to render an output object

```
# Define server function required to create the scatterplot
server <- function(input, output) {
     # Create the scatterplot object the plotOutput function is expecting
    output$scatterplot <- renderPlot(
     ggplot(data = movies, aes_string(x = input$x, y = input$y,
                                                  color = input$z)) +
            geom_point(alpha = input$alpha)
    )
}
```

Your turn

- Start with movies_05.R
- Add a new sliderInput defining the size of points (ranging from 0 to 5)
- Use this variable in the geom_ of the ggplot function as the size argument
- Run the app to ensure that point sizes react when you move the slider
- Compare your code / output with the person sitting next to / nearby you

5m 00s

Solution to the previous exercise

apps/movies/movies-06.R

✓

SOLUTION

Reactive flow

Suppose you want the option to plot only certain types of movies as well as report how many such movies are plotted:

1. Add a UI element for the user to select which type(s) of movies they want to plot
2. Filter for chosen title type and save as a new (reactive) expression
3. Use new data frame (which is reactive) for plotting
4. Use new data frame (which is reactive) also for reporting number of observations

DEMO

1. Add a UI element for the user to select which type(s) of movies they want to plot

```
# Select which types of movies to plot
checkboxGroupInput(inputId = "selected_type",
                   label = "Select movie type(s):",
                   choices = c("Documentary", "Feature Film", "TV Movie"),
                   selected = "Feature Film")
```

2. Filter for chosen title type and save the new data frame as a reactive expression

```
# Before app
library(tidyverse)

# Server
# Create a subset of data filtering for chosen titl
movies_subset <- reactive({
  req(input$selected_type)
  filter(movies, title_type %in% input$selected_typ )
})
```

Creates a **cached expression** that knows it is out of date when input changes

3. Use new data frame (which is reactive) for plotting

```
# Create the scatterplot object the plotOutput function is expecting
output$scatterplot <- renderPlot({
  ggplot(data = movies_subset(), aes_string(x = input$x, y = inp
                                           color = input$z)) +
    geom_point(…) +
    …
})
```

Cached - only re-run when inputs change

4. Use new data frame (which is reactive) also for printing number of observations

```
# UI
mainPanel(
  …
  # Print number of obs plotted
  uiOutput(outputId = "n"),
  …
  )
# Server
output$n <- renderUI({
  types <- movies_subset()$title_type %>%
    factor(levels = input$selected_type)
  counts <- table(types)

  HTML(paste("There are", counts, input$selected_type, "movies in this 
dataset.<br>"))
})
```

Putting it altogether

apps/movies/movies-07.R

DEMO

(also notice the HTML tags,
added for visual separation, in the mainPanel)

When to use reactive

- By using a reactive expression for the subsetted data frame, we were able to get away with subsetting once and then using the result twice
- In general, reactive conductors let you
  - not repeat yourself (i.e. avoid copy-and-paste code) which is a maintenance boon)
  - decompose large, complex (code-wise, not necessarily CPU-wise) calculations into smaller pieces to make them more understandable
- These benefits are similar to what happens when you decompose a large complex R script into a series of small functions that build on each other

Your turn

- For consistency, in movies_07.R, there should be at least one more spot on the app where the new movies_subset dataset should be used, instead of the full movies dataset
  - Hint: Does the data table match the plotted data?
- Find and fix
- Run the app to confirm your fix is working
- Compare your code / output with the person sitting next to / nearby you

3m 00s

Solution to the previous exercise

apps/movies/movies-08.R

✓

SOLUTION

Suppose we want to plot only a random sample of movies, of size determined by the user. What is wrong with the following?

```
# Server
# Create a new data frame that is a sample of n_samp
# observations from movies
movies_sample <- reactive({
  req(input$n_samp)      # ensure availability of value
  sample_n(movies_subset(), input$n_samp)
})

# Plot the sampled movies
output$scatterplot <- renderPlot({
  ggplot(data = movies_sample(),
         aes_string(x = input$x, y = input$y, color = input$z)) +
    geom_point(…)
})
```

?

Solution can also be found in **movies_09.R.**

Note that **output$n** and **output$datatable** are also updated in the script.

Implementation

Implementation of reactives

- **Reactive values** – reactiveValues():
  - e.g. input: which looks like a list, and contains many individual reactive values that are set by input from the web browser
- **Reactive expressions** – reactive(): they depend on reactive values and observers depend on them
  - Can access reactive values or other reactive expressions, and they return a value
  - Useful for caching the results of any procedure that happens in response to user input
  - e.g. reactive data frame subsets we created earlier
- **Observers** – observe(): they depend on reactive expressions, but nothing else depends on them
  - Can access reactive sources and reactive expressions, but they don't return a value; they are used for their side effects
  - e.g. output object is a reactive observer, which also looks like a list, and contains many individual reactive observers that are created by using reactive values and expressions in reactive functions

Suppose we want the user to provide a title for the plot. What is wrong with the following, and how would you fix it? See **movies_10.R**.

```
# UI
textInput(inputId = "plot_title",
          label = "Plot title",
          placeholder = "Enter text"),

# Server
output$pretty_plot_title <- toTitleCase(input$plot_title)

output$scatterplot <- renderPlot({
  ggplot(data = movies_sample(),
         aes_string(x = input$x, y = input$y, color = input$z)) +
    geom_point(alpha = input$alpha, size = input$size) +
    labs(title = output$pretty_plot_title)
})
```

?

Suppose we want the user to provide a title for the plot. What is wrong with the following, and how would you fix it? See **movies_10.R**.

```
# UI
textInput(inputId = "plot_title",
          label = "Plot title",
          placeholder = "Enter text"),

# Server
pretty_plot_title <- reactive({ toTitleCase(input$plot_title) })

output$scatterplot <- renderPlot({
  ggplot(data = movies_sample(),
         aes_string(x = input$x, y = input$y, color = input$z)) +
    geom_point(alpha = input$alpha, size = input$size) +
    labs(title = pretty_plot_title())
})
```

?

apps/movies/movies-11.R

Reactive expressions vs. observers

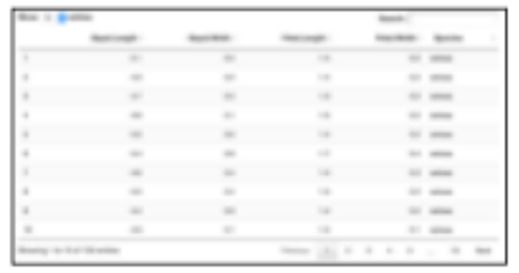
- Similarities: Both store expressions that can be executed
- Differences:
  - Reactive expressions return values, but observers don't
  - Observers (and endpoints in general) eagerly respond to reactives, but reactive expressions (and conductors in general) do not
  - Reactive expressions must not have side effects, while observers are only useful for their side effects

Render functions

Render functions

```
render*({ [code_chunk] })
```

- Provide a code chunk that describes how an output should be populated
- The output will update in response to changes in any reactive values or reactive expressions that are used in the code chunk



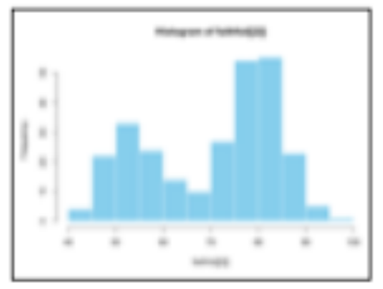
DT::**renderDataTable(expr**, options, callback, escape, env, quoted**)**

works with

dataTableOutput(outputId, icon, …**)**

renderImage(expr, env, quoted, deleteFile**)**

imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline**)**



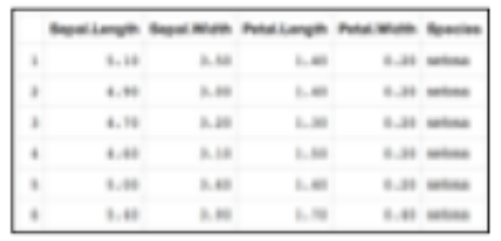
renderPlot(expr, width, height, res, …, env, quoted, func**)**

plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline**)**

```
'data.frame':   3 obs. of  2 variables:
 $ Sepal.Length: num  5.1 4.9 4.7
 $ Sepal.Width : num  3.5 3 3.2
```

renderPrint(expr, env, quoted, func, width**)**

verbatimTextOutput(outputId)



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa

renderTable(expr,…, env, quoted, func**)**

tableOutput(outputId)

foo

renderText(expr, env, quoted, func**)**

textOutput(outputId, container, inline**)**

renderUI(expr, env, quoted, func**)**

uiOutput(outputId, inline, container, …**)**
& **htmlOutput(outputId**, inline, container, …**)**

Recap

render*({ [code_chunk] })

- These functions make objects to display
- Results should always be saved to output$
- They make an observer object that has a block of code associated with it
- The object will rerun the entire code block to update itself whenever it is invalidated

Your turn

- Run the app in movies_11.R.
- Try entering a few different plot titles and observe that the plot title updates however the sampled data that is being plotted does not.
- Given that the renderPlot() function reruns each time input$plot_title changes, why does the sample stay the same?

3m 00s

Because the data frame that is used in the plot is defined as a reactive expression with a code chunk that does not depend on input$plot_title.

✓

SOLUTION