Understanding reactivity

Mine Çetinkaya-Rundel

@minebocek
mine-cetinkaya-rundel
mine@stat.duke.edu

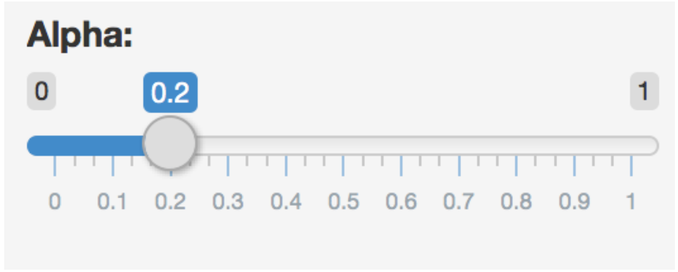
Reactivity 101

Reactions

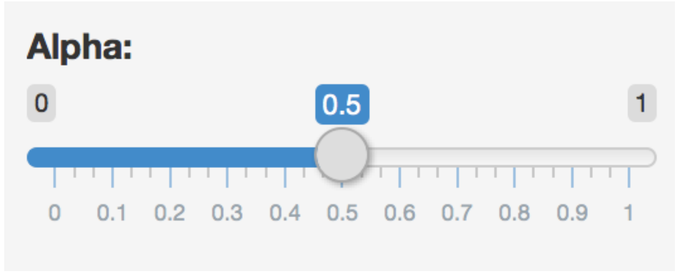
The **input$** list stores the current value of each input object under its name.

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

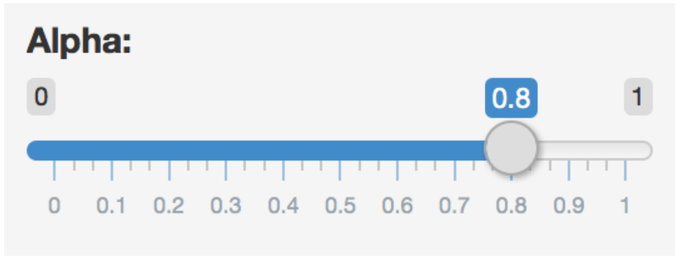
input$alpha



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-apps/movies-07.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

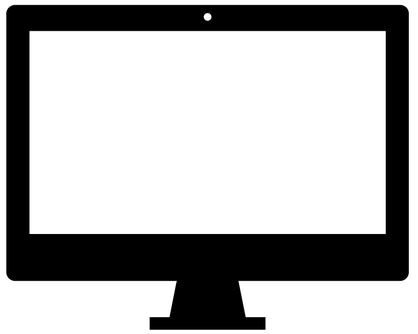
`movies-apps/movies-08.R`

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_type)
})
```

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 =
                                            col
    geom_point(…) +
    …
})
```

Cached - only re-run when inputs change

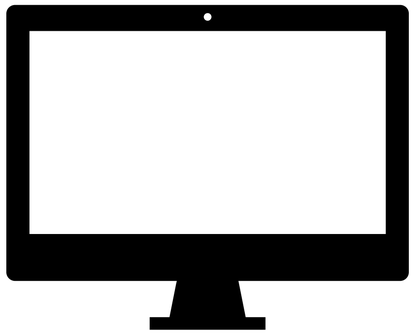
4. Use new data frame (which is reactive) also for printing number of observations

```
# 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

`movies-apps/movies-09.R`



DEMO

Also notice

- HTML tags for visual separation
- `req()`

When to use reactives

- 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

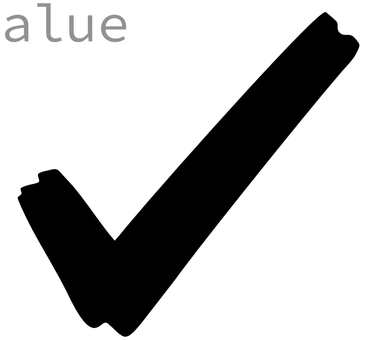
Suppose we want to plot only a random sample of movies, of size determined by the user. What is wrong with the following?

```
# Server
# Create a new data frame that is a sample of n_samp
# observations from movies
movies_sample <- 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(…)
})
```

```
# 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

Solution can also be found in `movies_10.R`.

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

Your turn

- Suppose we want the user to provide a title for the plot.
- Investigate and debug `movies_11.R` to add this functionality.
- Compare your code / output with the person sitting next to / nearby you.

3m 00s

Solution to the previous exercise

`movies-apps/movies-12.R`

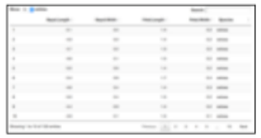
SOLUTION

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



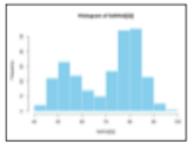
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)

works with

dataTableOutput(outputId, icon, …)

renderImage(expr, env, quoted, deleteFile)

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

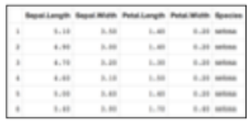


renderPlot(expr, width, height, res, …, env, quoted, func)

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

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

verbatimTextOutput(outputId)



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-apps/movies_12.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

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

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

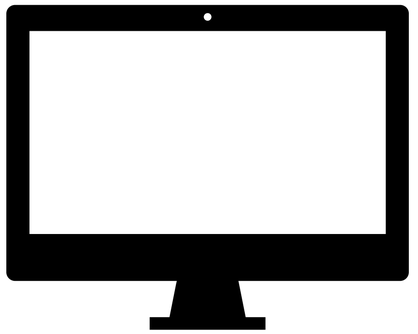
Stop-trigger-delay

Stop with `isolate()`

- Wrap an expression with `isolate()` to suppress its reactivity
- This will stop the currently executing reactive expression/observer/output from being notified when the isolated expression changes

Only update the alpha level when other inputs of the plot change

`movies-apps/movies-13.R`



DEMO

Delay with `eventReactive()`

- Calculate a value only in response to a given event with eventReactive()
- Two main arguments (the event to react to and the value to calculate in response to this event):

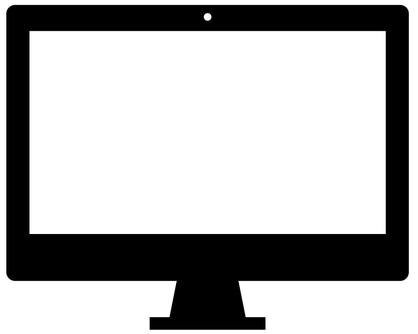
```
eventReactive(eventExpr, valueExpr, …)
```

simple reactive value - `input$click`,
call to reactive expression - `df()`,
or complex expression inside `{}`

the expression that produces the return value when `eventExpr` is invalidated

Remove the functionality for selecting types, instead randomly sample a user defined number of movies, but only sample and update outputs when an action button is pushed

`movies-apps/movies-14.R`



DEMO

Your turn

Update the previous app so that a sample with a default sample size is taken and plotted upon launch

3m 00s

Solution to the previous exercise

`movies-apps/movies-15.R`

SOLUTION

Trigger with `observeEvent()`

- Trigger a reaction (as opposed to calculate/recalculate a value) with observeEvent()
- Also two main arguments:

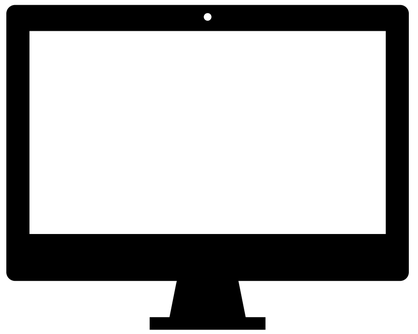
```
observeEvent(eventExpr, handlerExpr, …)
```

simple reactive value - `input$click`,
call to reactive expression - `df()`,
or complex expression inside `{}`

expression to call whenever `eventExpr` is invalidated

Add a button to write a csv
of the current random sample

`movies-apps/movies-16.R`



DEMO

Stop-delay-trigger

- `isolate()` is used to stop a reaction
- `eventReactive()` is used to create a calculated value that only updates in response to an event
- `observeEvent()` is used to perform an action in response to an event

Your turn

Debug the following app scripts:

- `review/whats-wrong.R`
- `review/mult-3.R`
- `review/add-2.R`

10m 00s