NC STATE UNIVERSITY
DEPARTMENT OF
STATISTICS

Dynamic UIs

Justin Post

Recap

`ui`

- Controls layout of app
- Basic layout uses a sidebar panel and a main panel
    - `bslib::page_sidebar()` or `shiny::sidebarLayout()`
- Use strings, formatted (html style) text, widgets (`*Input` functions), and output from `server` (`*Output` functions)

`server`

- Back-end for app
- Create outputs that react to inputs (must be a reactive context!)
    - `render*` functions, `reactive()`, and `reactiveValues()`
    - `observe()`, `observeEvent()`, and `eventReactive()`

Dynamic UI

- Often want to update UI based on user input!
- Methods for updating UI
  - `update*` functions
  - `renderUI()`/`uiOutput()`
  - `conditionalPanel()`

Using `update*` Functions

- Every input widget has a corresponding update function
  - `updateActionButton()`
  - `updateCheckboxInput()`
  - `updateNumericInput()`
  - …

Using `update*` Functions

- Every input widget has a corresponding update function
  - `updateActionButton()`
  - `updateCheckboxInput()`
  - `updateNumericInput()`
  - …
- Requires session argument on server() function

```
server <- function(input, output, session) {
  ---#  do stuff
}
```

- After all observers (reactive things) evaluate, updater sends message back to client

Using `update*` Functions

- Syntax of `update*` functions similar to the functions that created the inputs

```
numericInput(inputId,
             label,
             value,
             min = NA,
             max = NA,
             step = NA,
             width = NULL)

updateNumericInput(session,
                   inputId,
                   label = NULL,
                   value = NULL,
                   min = NULL,
                   max = NULL,
                   step = NULL)
```

Using `update*` Functions

- Syntax of `update*` functions similar to the functions that created the inputs
- Any arguments with `NULL` values ignored
- For `radioButtons()`, `checkboxGroupInput()`, and `selectizeInput()`, the set of choices can be cleared by using `choices = character(0)`

`updateSliderInput()` (First Attempt)

```r
ui <- fluidPage(
      ...
      sidebarPanel(
         sliderInput("bins", "Number of bins:",
                     min = 1, max = 50, value = 30),
         numericInput("maxBins", label = "Set Maximum Number of Bins",
                      value = 50, min = 1, max = 100)
      ),
      ...
)

server <- function(input, output, session) {
    ...
    updateSliderInput(session, "bins", max = input$maxBins)
}
```

What is our issue?

`updateSliderInput()` (Fixed)

```r
ui <- fluidPage(
      ...
      sidebarPanel(
         sliderInput("bins", "Number of bins:",
                     min = 1, max = 50, value = 30),
         numericInput("maxBins", label = "Set Maximum Number of Bins",
                      value = 50, min = 1, max = 100)
      ),
      ...
)
server <- function(input, output, session) {
    ...
    observe({
      updateSliderInput(session, "bins", max = input$maxBins)
      })
}
```

`update*` UI Functions

- Use the template app
- Try to add a numeric input for the user to specify the largest value of the slider
- Use the `updateSliderInput` function to update the max of the slider
- Add an `actionButton` to only update when pressed (so no intermediate updates while typing)

`renderUI()` and `uiOutput()`

- `renderUI()` and `uiOutput()` allow for flexible dynamic UI elements
- Recall: Shiny essentially writes HTML/JavaScript for us!

```r
print(fluidPage(titlePanel(title = "Hi"),
                sidebarLayout(sidebarPanel(), mainPanel())))
```

```
## <div class="container-fluid">
##   <h2>Hi</h2>
##   <div class="row">
##     <div class="col-sm-4">
##       <form class="well" role="complementary"></form>
##     </div>
##     <div class="col-sm-8" role="main"></div>
##   </div>
## </div>
```

`renderUI()` and `uiOutput()`

- `renderUI()` and `uiOutput()` allow for flexible dynamic UI elements
- Recall: Shiny essentially writes HTML/JavaScript for us!

```r
print(numericInput("id", "Label User Sees", value = 10))
```

```
## <div class="form-group shiny-input-container">
##   <label class="control-label" id="id-label" for="id">Label User Sees</label>
##   <input id="id" type="number" class="shiny-input-number form-control" value="10"/>
## </div>
```

`renderUI()` and `uiOutput()`

- `renderUI()` makes a **reactive version** of a function that generates HTML!
- Can have `renderUI()` return
    - A widget or other function that makes HTML
    - A shiny 'tag object' created via `shiny::tagList()`
- Use with `uiOutput()` in UI file

`renderUI()` and `uiOutput()` (updating a widget)

```r
ui <- fluidPage(
      ...
      sidebarPanel(
          uiOutput("slider"),
          numericInput("maxBins", label = "Set Maximum Number of Bins",
                       value = 50, min = 1, max = 100)
      ),
      ...
),
server <- function(input, output, session) {
    ...
    output$slider <- renderUI({
        sliderInput("bins", "Number of bins:", min = 1,
                    max = input$maxBins, value = 30)
    })
}
```

`renderUI()` and `uiOutput()` (outputting HTML)

```r
ui <- fluidPage(
      ...
      sidebarPanel(
           uiOutput("info"),
           numericInput("purchase", label = "How Many?",
                        value = 50, min = 0, max = 100)
      ),
      ...
),
server <- function(input, output, session) {
    ...
      output$info <- renderUI({
        text <- paste0("You have selected to buy ", input$purchase)
        h3(text)
      })
}
```

`renderUI()` and `uiOutput()` (using `tagList()`)

```r
ui <- fluidPage(
      ...
      sidebarPanel(
           uiOutput("info"),
           numericInput("purchase", label = "How Many?",
                        value = 50, min = 0, max = 100)
      ),
      ...
),
server <- function(input, output, session) {
    ...
      output$info <- renderUI({
        text <- paste0("You have selected to buy ", input$purchase)
        slidey <- sliderInput("bins", "Number of bins:", min = 1,
                    max = input$maxBins, value = 30)
        tagList(
          h3(text),
          slidey
          )
      })
}
```

`renderUI()` and `uiOutput()`

- Use the template app
- Try to add some dynamic updating text and things to the UI

`conditionalPanel()`

- Create a 'panel' that is only visible if a condition is met
- Condition can depend on input widget
    - Accessed differently! (Use a '.' not a '$' and javascript style comparisons)

conditionalPanel()

```r
...
sidebarPanel(
  selectInput("plotType", "Plot Type",
            c(Scatter = "scatter", Histogram = "hist")),

  # Only show this panel if the plot type is a histogram
  conditionalPanel(condition = "input.plotType == 'hist'",
          selectInput("breaks", "Breaks",
               c("Sturges", "Scott", "Freedman-Diaconis", "[Custom]" = "custom")),
      # Secondary conditonalPanel, Only show this panel if Custom is selected
      conditionalPanel(
          condition = "input.breaks == 'custom'",
          sliderInput("breakCount", "Break Count", min = 1, max = 200, value = 40)
      )
  )
)
```

`conditionalPanel()`

- Use the template app
- Try to add a new UI element using a conditional panel!

Dynamic UI Recap

- Often want to update UI based on user input!
- Methods for updating UI
  - `update*` functions
  - `renderUI()`/`uiOutput()`
  - `conditionalPanel()`