**Building Web Applications in R with Shiny**

**Reactive Programming**

**Reactive Sources**

User input that comes through a browser interface.

Sources can only be parents.

**Reactive Conductors**

Reactive component between a source and an endpoint.

Conductors can be both a dependent (child) and have dependents (parent).

**Reactive Endpoints**

Something that appears in the user’s browser window (e.g. a plot or a table)

Endpoints can only be children.

One reactive source can be connected to multiple endpoints. Also one reactive endpoint can be connected to multiple sources.

**reactive**() when the same dataset is used in multiple outputs, it makes sense to make our code more efficient by using a reactive data frame. reactive() is a reactive conductor. reactive() is an implementation of reactive conductors.

Example: movies\_selected <- reactive({

req(input$selected\_var)

movies %>% select(input$selected\_var)

})

correlated

output$moviestable <- DT::renderDataTable({

req(input$selected\_var)

DT::datatable(data = movies\_selected() %>% select(input$selected\_var),

options = list(pageLength = 10),

rownames = FALSE)

})

Why use reactives?

1. By using a reactive expression for the subsetted data frame, you can subset once and reuse the subsetted result multiple times. This makes code far more efficient by doing away with the need for repeating code
2. Decompose large, complex calculations into smaller pieces to make them more understandable. This also allows breaking down large complex R scripts into a series of small functions that build on each other.

**Reactive Programming (continued)**

**Functions versus Reactives**

Each time you call a function, R will evaluate it. On the contrary, reactive expressions are lazy in that they only get executed when their input changes. Even if you call a reactive expression multiple times, it only re-executes when its input(s) change.

Using many reactive expressions can create a complicated dependency structure in your app.

**reactlog** is a graphical representation of this dependency structure. To view reactlog:

1. In a fresh R session, run options(shiny.reactlog = TRUE)
2. Then, launch you app as you normally would
3. In the app, press Ctrl + F3

**observeEvent**() and **updateNumericInput**() example:

observeEvent(input$selected\_type, {

n\_max <- nrow(movies\_selected())

updateNumericInput(session, “n\_samp”, max = n\_max)

updateNumericInput(session, “n\_samp”,

label = paste0(“Sample size (max = “, n\_max, “):”))

})

**sample\_n**()

sample\_n(movies\_selected(), input$n\_samp)

**reactiveValues**() is an implementation of reactive sources.

**reactive**() is an implementation of reactive conductors. reactive() can access values or other reactive expressions and they return a value. reactive() is useful for caching the results of any procedure that happens in response to user input.

**observe()** is an implementation of reactive endpoints. An output$\* object is an observer. observe() can access reactive sources and reactive expressions, but they don’t return a value. observe() is used for its side effects, which primarily involves sending data to the web browser.

**Reactives versus Observers**

Both store expressions that can be executed. However only reactive expressions return values, whereas observers do not return values. reactive() is used for calculating values without side effects. observe() is used for performing actions with side effects. Do not use an observe() when calculating a value and especially do not use reactive() for performing actions with side effects.