With the Commons trick, as I call it, all of the goals for the applet have been achieved, but there is still some room for improvement and additional tricks. This first one is to create a UI element to show how many random numbers are present in the tables. In other words, the sum of the \*\*randCount\*\* inputs, and while that sounds simple enough, it actually is not because each is within separate namespaces. Fortunately there are some means of getting information out of namespaces, with some being more effective and appropriate than others. This section is going to be concerned with the more appropriate methods, and by appropriate I mean not defying the nature of namespaces.

Before getting too far into extracting the information, it makes sense to get the UI element taken care of first.

```{r echo = TRUE, eval = FALSE}

ui <- fluidPage(

titlePanel(NULL, "Shiny Modularization Example"),

textOutput('randCount'),

…

)

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

…

observe({

…

output$randCount <- renderText({

paste0("There are ", sum(COUNT), " random numbers")

})

})

}

```

Removing the code we have already seen or I am about to cover, this is what we have. In the UI I have a simple \*\*textOutput\*\* element, meant to be placed at the top of the page, and then in \*\*server\*\*’s \*\*observe\*\* the calls to the modules and the text are prepared. The \*\*COUNT\*\* object is to be a list of the \*\*randCount\*\* values, and so is what we need to create.

Now, knowing that \*\*function\*\* and \*\*moduleServer\*\* return values, it would be tempting to do something like the code I am about to show you, but this does not work:

```{r echo = TRUE, eval = FALSE}

exampleServer <- function(name) { roundTerm <- reactive(input$roundTerm)

moduleServer(name, function(input, output, session) {

exTAB <- reactive(exFUN(input$randCount))

output$TAB <- renderTable(exTAB(), digits = reactive(roundTerm()), striped = reactive(input$striping))

input$randCount

})

}

```

While this will return the value of \*\*randCount\*\* each time you call \*\*exampleServer\*\*, making a list very easy to generate, it also causes a bug. For some reason, it forces \*\*randCount\*\* to reset to 3 each time you change it, completely negating that input. I am unsure why this is, but that is what happens and so makes for an unacceptable solution. The idea is not completely wrong though, it just cannot be applied in this way.

```{r echo = TRUE, eval = FALSE}

exampleCount <- function(name) {moduleServer(name, function(input, output, session) {

input$randCount

})}

observe({

lapply(modList(), exampleServer)

output$modules <- renderUI( lapply(modList(), exampleUI) ) output$randCount <- renderText({

COUNT <- unlist(lapply(modList(), exampleCount))

paste0("There are ", sum(COUNT), " random numbers")

})

})

```

By putting the same thing in a separate module, it will work and without the bug. Clearly the bug is from some interaction being triggered with the rest of \*\*exampleServer\*\*, so by using the separate \*\*exampleCount\*\* we have a solution. We then just need to call \*\*exampleCount\*\* like we do the other modules, with \*\*lapply\*\*, and after an application of \*\*unlist\*\*, we have a workable \*\*COUNT\*\*. If you would rather avoid \*\*unlist\*\*, you can use \*\*sapply\*\* instead, but I am keeping to \*\*lapply\*\* for now. The reason for \*\*unlist\*\* is \*\*lapply\*\* will return an object of class \*\*list\*\* but \*\*sum\*\* requires a vector of numbers, which \*\*unlist\*\* provides. The difference between \*\*lapply\*\* and \*\*sapply\*\* is the latter automatically simplifies the output of the former, which achieves the same goal. (If you look at the documentation of the two functions, \*\*sapply\*\* is described as a wrapper of \*\*lapply\*\* with its \*\*simplify\*\* argument enabled.)

While we could stop here, I actually want to take this a step further, to something more complicated but also more powerful. Also it will explain the name of this section: \*\*reactiveValuesToList\*\*.

The objects \*\*input\*\* and \*\*output\*\* in Shiny are \*\*reactiveValues\*\*, so basically lists but each element is a reactive. With the function \*\*reactiveValuesToList\*\* it is possible to create a non-reactive version of these objects. Though we only want one value out of \*\*input\*\*, you may find a situation that you want more than one, and this function is a way to get them.

```{r echo = TRUE, eval = FALSE}

exampleCount <- function(name) {moduleServer(name, function(input, output, session) {

# input$randCount

reactiveValuesToList(input)[["randCount"]]

})}

```

It is worth noting this \*\*input\*\* object will only contain those objects within the namespace passed to \*\*moduleServer\*\*. With \*\*reactiveValuesToList\*\* it produces a list, and then we can select specific objects by their names. It is necessary to use double brackets here, but inside them you can place a vector of object names. You can also be a little more creative to selectively remove those elements you do not want, which might be preferable for some situations.

While this certainly works, and works within the confines of how Shiny modules are likely expected to work, this is not the only way to get information out of a module. This other method I have found is the topic of the next section.