With the basics of functions and namespaces covered, next will be how to actually modularize Shiny code, but first I want to cover the example applet I will be using. Though I have changed it a little, the \*\*exFUN\*\* used before will still be at the heart of this applet.

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

library(shiny)

# exFUN <- function(n, r = 2) runif(n, min = 0, max = 9) |> round(r)

exFUN <- function(n) runif(n, min = 0, max = 9)

ui <- fluidPage(

titlePanel(NULL, "Shiny Modularization Example"),

fluidRow(

column(2, numericInput(inputId = 'roundTerm', label = "Rounding Term", value = 2)),

),

fluidRow(

column(2, numericInput(inputId = 'randCount', label = "Count", value = 3, min = 0)),

column(2, checkboxInput(inputId = 'striping', label = "Table Striping")),

),

tableOutput('TAB'),

)

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

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

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

}

shinyApp(ui, server)

```

Though I do wish to assume anyone reading this guide already has a working knowledge of Shiny, I still want to take the time to run through it.

Starting at the top we have first the call to load the Shiny library and then two definitions for \*\*exFUN\*\*, with the first commented out. That first version similar to what I used before, but I made one change for myself. This was to call \*\*round\*\* with a pipe, and this is just to help me remember what the pipe character is for base R; \*\*|>\*\*. The second version of this function is without \*\*round\*\* at all because for the applet I want to achieve rounding a different way, but I will get to that latter. This version of the \*\*exFUN\*\* will just produce \*\*n\*\* random numbers between 0 and 9.

Before going further, I want to explain why I decided to make \*\*exFUN\*\* essentially a wrapper for \*\*runif\*\*; to simulate different datasets being processed by the modules. The specific data itself is inconsequential for this guide, but the fact the modules we will be making can each work with different pieces of data does matter.

With the custom function made, next we create the Shiny UI, which uses the \*\*fluidPage\*\* format and has five elements to it, for now. I find fluid page is a nice format to work from most of the time, but have used others when appropriate. The first of the elements is a call to \*\*titlePanel\*\* with the second argument, the \*\*windowTitle\*\* argument set just so the browser shows something better than the host address. The \*\*title\*\* argument being \*\*NULL\*\* means no additional text is placed on the page.

The second element is a \*\*numericInput\*\* but I have it in a \*\*column\*\* of width 2 in a \*\*fluidRow\*\*. That likely seems odd and unnecessary, and it is for now but will be needed later. I have already built the final, modularized applet and this version is from cutting that one down. Ultimately this \_Rounding Term\_ input will have another control next to it in the row, so it is ready for it before it needs to be.

Next we have another \*\*fluidRow\*\* made because the remaining two inputs I want in a row together, and as it happens both fit nicely in columns of width 2.

The first of these inputs is \*\*randCount\*\*, a numeric that will control how many random numbers are generated. This is why the \*\*label\*\* is \_Count\_ and the \*\*min\*\* is 0. The default \*\*value\*\* is 3 because I think that is fair to start at.

The second input is a \*\*checkboxInput\*\* for \*\*striping\*\*, which is for toggling striping within a table on and off. Functionally, this is largely unnecessary as a control for the user, but for this guide it can be an additional example of a control. For clarity, the value of \*\*checkboxInput\*\* is either \*\*TRUE\*\* or \*\*FALSE\*\*, which is entirely appropriate for this option that is also just a toggle.

Finishing off the UI is the \*\*tableOutput\*\* element for rendering a table. The name of the table is, quite appropriately and simply, \*\*TAB\*\*.

With the UI portion finished, we come to the server portion, which is quite simple, at least for now. Its first line is to create the \*\*exTAB\*\* object, for example table. It is a \*\*reactive\*\* of \*\*exFUN\*\* that has been given \*\*randCount\*\* from the numeric input covered earlier. It may seem odd that I would create this object here, as I could just as easily call \*\*exFUN\*\* within the \*\*renderTable\*\* command in the next line, but I have a reason and it relates back to why I changed \*\*exFUN\*\* for this applet by removing rounding. It also relates to why I have been using \*\*runif\*\*, a random number generator.

In this applet, the processing of \*\*exFUN\*\* will be very quick, but in other applets I have made the processing can be much slower. This makes changes to the number of digits for rounding something laborious, as each time the number changes, the whole table must be recalculated. By separating the processing of \*\*exFUN\*\* such that it is only dependent on \*\*randCount\*\*, changing the rounding or the striping will not impact it. If you adjust those two inputs, the random values will be unchanged, except in appearance.

Looking at the creation of \*\*output$TAB\*\* as a \*\*renderTable\*\* object, you see it simply takes the \*\*exTAB\*\* reactive as the expression, and then two optional arguments, both with their values within calls to \*\*reactive\*\*.

The reason for the calls to \*\*reactive\*\* is so the table will react to changes in the values of the inputs I am passing to the arguments. Also an error will be thrown if these inputs, which are already reactives, are not used within a reactive context. It would be nice if \*\*renderTable\*\* would except these as reactives implicitly, but that is not how it works.

The first of these arguments is \*\*digits\*\* that controls the number of digits shown within the table, like rounding. It is not actually rounding though, because the behavior when you go to a negative value is quite different from what \*\*round\*\* does when given a negative value. If \*\*digits\*\* is given a negative value, the table shows the results in scientific notation with the magnitude of \*\*digits\*\* being the precision. Changing the table to use \*\*round\*\* would be quite easy, as would applying a minimum to \*\*roundTerm\*\* so it cannot be negative.

The second argument is for controlling if the table is stripped or not, meaning the odd rows have a light gray background while the even rows are left white (or transparent and white is the normal background). As this is a basic toggle, the \*\*checkboxInput\*\* for \*\*striping\*\* was the most appropriate input type.

Except for the call to \*\*shinyApp\*\* that finishes off this applet. In the following sections we will be modularizing this for the purpose of having multiple tables of random numbers, each with its own count of numbers and own striping controls. The rounding will actually be handled globally, so you can see how that is done, as well as a little trick I realized.