With the basics of modularizing Shiny code covered, time for some more advanced tricks I have discovered. Or at least tricks that are not directly covered by the official documentation and guides, but can come in handy. First is a trick I call Commons, because the idea is to make something common to each instance of a server module. In fairness, this one really is not that involved, but it can still be a useful trick to know.

In the prior sections I stated I wanted the ability to control the rounding for the tables of random numbers, but had not set it up yet. Part of the reason for that is I want this rounding to be a global input, rather than an input you will find within the UI modules. Due to how \*\*moduleServer\*\* works, we cannot just call \*\*input$roundTerm\*\* because it will look within a namespace that does not contain it. This means we need to pass the value into the module somehow. The most obvious means is with an argument, like this:

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

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

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

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

})

}

exampleServer(‘module 1’, reactive(input$roundTerm))

```

Though this works, I do not much like this form because you have to always provide the \*\*reactive(input$roundTerm)\*\* argument when \*\*exampleServer\*\* is called. You can give \*\*roundTerm\*\* a default value of that reactive, but I still do not like it because this does not really deserve an argument. The goal is not to be able to control the rounding whenever the module is called, but to have it controlled by the global input I created.

The solution I found here is not hinted at in the official modularization guide, but something near to it is. The arguments of the outer function are passed on to \*\*moduleServer\*\*, and if they are, why not anything else within the outer function?

```{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))

})

}

```

By moving the assignment of this new \*\*roundTerm\*\* object within the outer function, it will be available to the \*\*moduleServer\*\* and without having to explicitly pass it anything. I call any such objects I place in the outer function ahead of \*\*moduleServer\*\* the commons, because they will be common to every instance of that module.

In this situation, this really is not that advanced a trick, but it is still pretty powerful for other situations. Or rather, the realization that you can have code in the outer function before \*\*moduleServer\*\* can be powerful, as it opens up various possibilities. Remember, this code will be executed within the function’s temporary environment, and so does not need to influence anything in the parent environment, but it will still have access to the objects of its parent environment. Perhaps bridges would be a better title for this trick, but I will stick with commons as its original and still primary purpose for me is to make something commonly accessible to \*\*moduleServer\*\*.