To be honest, there is little purpose to modularizing just parts of the UI or just parts of the server portion for an applet, as the modules for both should be made to work together. However, this is a convenient place to divide the content of this guide, so that is how it is going to be. Luckily of the two, the UI is the easier to modularize, at least in my opinion.

There are two things we will need before getting started; an idea of the UI without modularization, and the goal for the UI with it. We already have the UI without modularization, so here is that code again:

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

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'),

)

```

The goals I have for the modularization are for each module to contain a table of random numbers and the control for how many random numbers to generate, and then for the number of these modules to be controlled. The \*\*roundTerm\*\* input I want to be global, so it will not be within a module, but the \*\*striping\*\* input can be. Actually having one of these inside and the other outside the UI module is helpful because of what I want to show later with the server modules. In total that is three UI elements I want to have modularized, so we can remove those from the UI and place them in a custom function. There is more involved than this for modularizing, but this is a good place to start.

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

exampleUI <- function() {

tagList(

fluidRow(

column(2, numericInput(inputId = 'randCount', label = paste0("Count ", name), value = 3, min = 0)),

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

),

tableOutput('TAB'),

)

}

ui <- fluidPage(

titlePanel(NULL, "Shiny Modularization Example"),

fluidRow(

column(2, numericInput(inputId = 'modCount', label = "Module Count", value = 2, min = 1)),

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

),

exampleUI(),

exampleUI(),

)

```

If you try running this code, it will work and, technically, it is modular in that we have created this \*\*exampleUI\*\* function, but it will not work properly. If you try running it and creating a table within the server function, nothing will happen because Shiny does not allow an object name to be repeated. Truthfully, this scenario of calling \*\*exampleUI\*\* twice in the UI should throw an error, leaving me somewhat puzzled it has not, but at the same time it does not actually function, so the error is present, nonetheless.

To solve this issue we need to go back to the second section of this guide and apply a namespace here. That will allow us to call this module as much as we wish, and with unique object IDs so the server code will work with it. Of course we will need the server code to also use namespaces, but that is for the next section.

Adding namespacing to this \*\*exampleUI\*\* function is quite easy and straightforward.

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

exampleUI <- function(name) {

ns <- NS(name)

…

}

```

I am skipping over the contents of the function for now, because I want to focus on what has happened here first. We have given the function an argument, \*\*name\*\*, and then passed it to the \*\*NS\*\* function. As you may remember, \*\*NS\*\* can take two arguments, but when only the first one is given, the \*\*namespace\*\* argument, it will return a function that generates the object IDs for that namespace. That function has been assigned to \*\*ns\*\*, but any ID will do.

As this is something the official guide on modularization has wrong, I want to emphasize that we are passing \*\*NS\*\* a value for its \*\*namespace\*\* argument. The second argument is named \*\*id\*\* and the guide uses that as the argument name when constructing UI modules even though it refers to the \*\*namespace\*\* argument, which only tempts confusion.

By the way, something else the guide states that is incorrect is that you must end the names of these modules, such as \*\*exampleUI\*\*, with “UI” or “Server” if this were a server function. I have tested both and this does not appear to be the case, so you should be safe to name these however you see appropriate. In this case \*\*exampleUI\*\* is appropriate, and so the convention that may have been necessary for an earlier version of Shiny, continues.

With \*\*ns\*\* created, the next step is to show how it is used in the rest of the \*\*exampleUI\*\* function.

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

exampleUI <- function(name) {

ns <- NS(name)

tagList(

fluidRow(

column(2, numericInput(inputId = ns('randCount'), label = paste0("Count ", name), value = 3, min = 0)),

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

),

tableOutput(ns('TAB')),

)

}

```

For all three UI elements, their IDs are first passed to \*\*ns\*\*, thereby creating the appropriate namespaced IDs to pass to the element. You can also see I am using the \*\*name\*\* argument to better identify the \*\*randCount\*\* input with a unique label. Remember, \*\*exampleUI\*\* is still a normal function, so you can do various things to it, just as you would a normal function. There are some limitations from not working in the server portion of the applet though, but there are still some things you can achieve here.

Technically at this point we have successfully modularized the UI and all that is left to show you is calling \*\*exampleUI\*\* when building the UI, but we have not yet achieved the goals I laid out. Still, showing the UI built for what we have now will be good, as chances are this is how such modularization will be used.

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

ui <- fluidPage(

titlePanel(NULL, "Shiny Modularization Example"),

fluidRow(

column(2, numericInput(inputId = 'modCount', label = "Module Count", value = 2, min = 1)),

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

),

exampleUI('module 1'),

exampleUI('module 2'),

)

```

As you can see I have \*\*exampleUI\*\* called twice with different names, and yes, the spaces in the names appear to work. (I only just tried it and so it might not be completely reliable, but so far it is.) Though I have both at the end of the UI, they could be anywhere in it, like any other element.

While this implementation satisfies the first half of my goals, by modularizing the table and count of random numbers within a table, I also want to control the number of these modules. That is what the new \*\*modCount\*\* input is for. Unfortunately there is no way to directly have the number of calls to \*\*exampleUI\*\* controlled by \*\*modCount\*\* within the UI, as the processing of \*\*modCount\*\* must be done in the server section. This calls for the use of \*\*uiOutput\*\*, which allows one to build UI components in the server section, and then pass them to the UI for rendering. While it works, it is not the best system and has some limitations, but that is a completely separate discussion.

That brings this section of the guide to an end, as the UI portion is done. Next up is the server portion that is a bit more complicated, but once you get a handle on its complexities, there are some neat things you can do, though those will be for the Advanced sections.