My feeling is if you are interested in learning how to modularize R-Shiny code, then you likely already know how to work with R and do some modularizing by creating custom functions. However, just in case and to get started with the example function I intend to use, this is where I want to start.

Defining a custom function is fairly easy with the \*\*function\*\* command. It has and expects three parts, not counting the object you assign it to. The first of these parts is the list of arguments, those values passed to the function. The second is the body of the function, or the expression it is to execute when called. The third is value to be returned by the function when it finishes.

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

exFUN <- function(ARGUMENTS) {

BODY

return(VALUE)

}

```

The list of arguments is actually optional, so if the function does not require any inputs passed to it when it is called, you can leave that blank. You can also provide default values for arguments, as I will show in my example shortly.

The body of the function is necessary, if only because there would be no point to calling a function that does nothing.

The value to be returned both is and is not required. A function always returns a value, whether you specify one or not using the \*\*return\*\* command. If you do not include the \*\*return\*\* command, then the function will return whatever the last evaluated expression was. Depending then, \*\*return\*\* is optional, but it can be handy to explicitly state what the output is supposed to be. Also you can use \*\*return\*\* multiple times in the body of the function. When the function reaches one though, it will provide its value and stop, which can be handy if you want an early exit from the function, such as if a certain condition has been met.

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

exFUN <- function(n, r = 2) {

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

return(round(out, r))

}

```

In my \*\*exFUN\*\* function there are two arguments, \*\*n\*\* and \*\*r\*\* with \*\*r\*\* having a default value of 2. As you can see, the default value was provided by setting the argument’s object name equal to that value. The purpose of this function is to provide some number (\*\*n\*\*) of random values between 0 and 9, and round them to two decimals or whatever amount (\*\*r\*\*) you wish.

To identify the body of the function I have surrounded it in curly brackets, { }, indicating to R it should take this whole block of code as the function’s expression. If the body were written on a single line, this would not be necessary, as you can see here:

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

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

```

Though visually different, this is functionally identical to what I wrote above. I have removed the \*\*out\*\* object by providing its value directly to the \*\*round\*\* function, and I have dropped the \*\*return\*\* command, as it is not necessary. It is also not necessary in the earlier form, but I wished to be more explicit then.

Except for calling the function, which is the same as calling any function, there is only one point left to cover for custom functions, concerning this guide. R has the concept of environments, which are, very simply, collections of objects. Objects in one environment are separate from those in another. Functions create their own, temporary environments when they run, which can be useful because it means whatever is done by the body of the function will not influence anything outside of the function, unless explicitly told to. In the example below, an \*\*out\*\* object is created outside of the function and despite \*\*exFUN\*\* assigning a value to \*\*out\*\*, the original value assigned outside the function is returned when called.

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

out <- 0

exFUN <- function(n, r = 2) {

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

return(round(out, r))

}

exFUN(2)

out

```

If you wish to have the function actually change the value as it runs, you want to use the `{r}<<-` command as that will go up the environment tree until it either finds an object by the given name, or reaches the global environment, what you normally work in, and create the object there.

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

rm(out)

exFUN <- function(n, r = 2) {

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

return(round(out, r))

}

exFUN(2)

out

```

I have removed the \*\*out\*\* object in this example using \*\*rm\*\* to show how that object has been created and given a value, despite not existing within the global environment initially.

An alternative to using super-assignment is to explicitly give the environment the object is in.

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

rm(out)

DATA <- new.env()

DATA$out <- 0

exFUN <- function(n, r = 2) {

DATA$out <- runif(n, min = 0, max = 9)

return(round(DATA$out, r))

}

exFUN(2)

DATA$out

```