App components:

1. User Interface (UI) - Create UI for the app. The UI will be a web document written in html, but you don’t need to know html as we will generate it from R.
2. Server instructions – create a set of instructions for the server to follow. These instructions will tell the user what to do when they change the different input objects.

**BUILDING THE USER INTERFACE**

App template

library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

shinyApp(ui = ui, server = server)

This is the template that you will always be using. *Running this creates an empty web app.*

Terminology:

* fluidPage() <- In fluid web design, the widths of page elements are set proportional to the width of the screen or browser window. A fluid website expands or contracts based on the width of the current viewport. Fluid design helps make websites more usable across device types with varying screen dimensions.
* server <- What Is a Web Server? Web servers are software or hardware (or both together) that stores and delivers content to a web browser at a basic level. The servers communicate with browsers using Hypertext Transfer Protocol (HTTP).

Think of your app in terms of **inputs** and **outputs**

* Input - user input
* Output - the output is presented on the UI and depends on the input

Add them to your app as arguments to fluidPage(). Anything added to fluidPage will appear on the UI.

ui <- fluidPage(

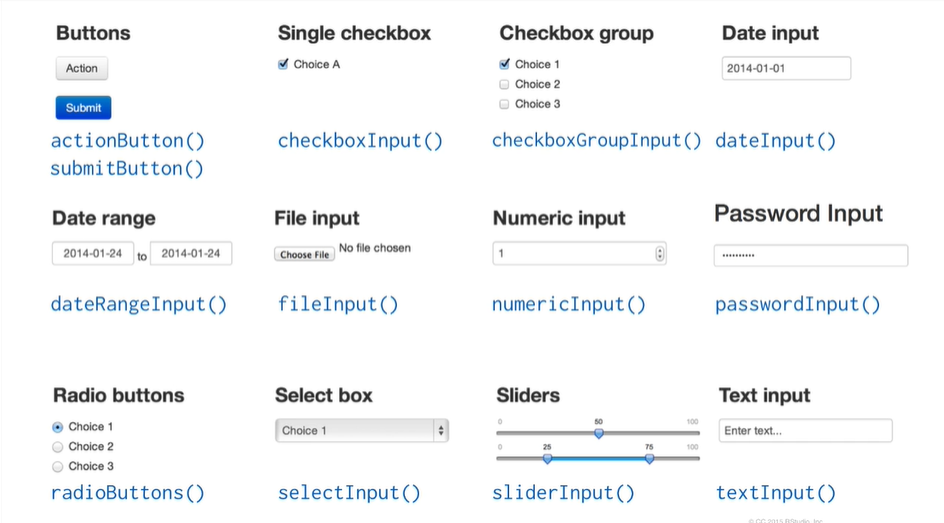
# Input() functions,

# Output() functions

)

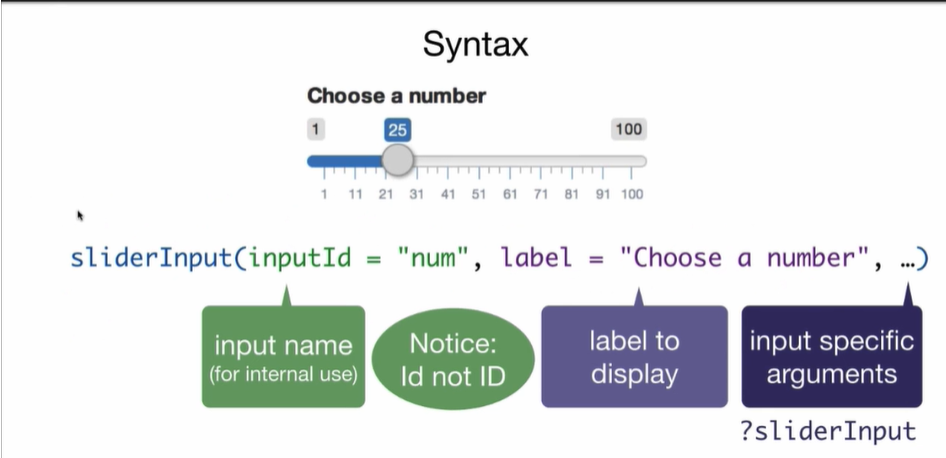
Inputs

Shiny provides a dozen input functions that come with the shiny library.



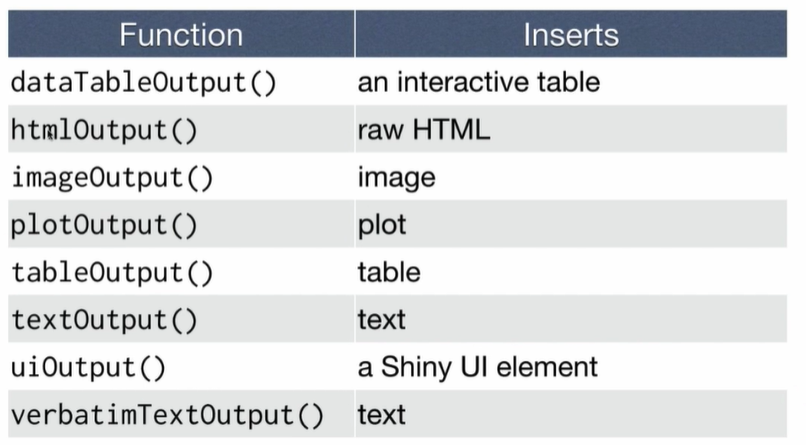
Each of the input functions take the same basic syntax:

* They all begin by taking an argument called **inputId** which allows you to assign a name to the input.
* The second argument is a **label** argument which describes what the input object does. If you don’t want to include a label just use empty quotation marks like this “”
* Finally, you can use additional arguments. To know which specific arguments to use with the specific input object, you can look at the help page for that input object.



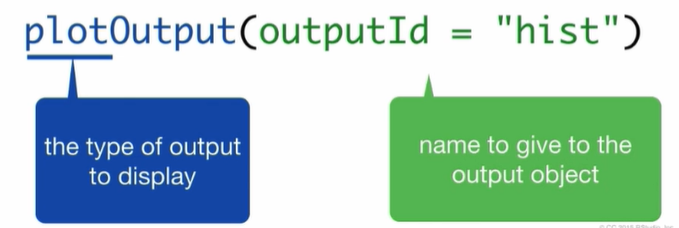
Outputs

Outputs are things like plots, tables, text, etc. Shiny provides several output functions that places outputs into your app. Notice that the output to get is placed just before the Output() function. For example, if you needed a plot, then **plotOutput()**.



To display output, add it to **fluidPage()** with an Output() function. The output syntax is simple:

* Choose an output function
* The first argument of the output function is **OutputId** which is used to name the output object



**Example**

ui <- fluidPage(sliderInput(inputId = “number”, label = “Choose a number”, value = 25, min = 0, max = 100), plotOutput(outputId = “hist”)

)

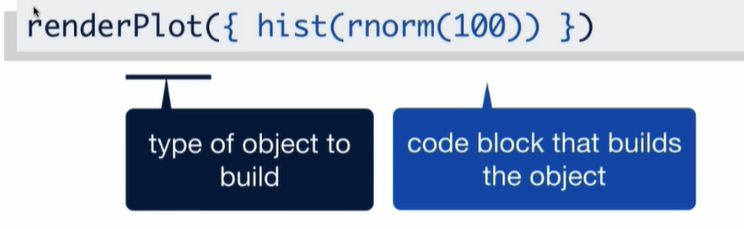
*This will not display a plot but will add space in the UI for an R object. To display the object we must build it in the server.*

**SERVER: ASSEMBLE INPUTS INTO OUTPUTS**

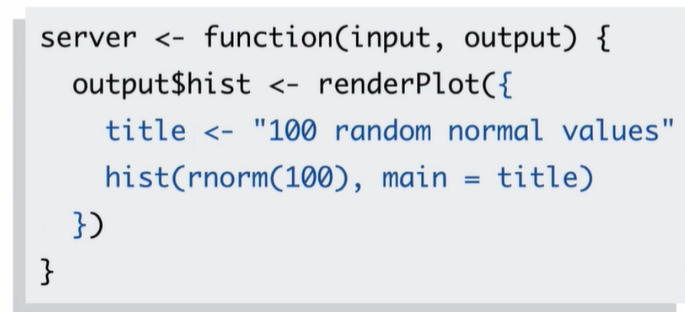
Three rules for writing the server function:

1. If you are building an output object, save that to **output$** (e.g., output$hist). Make sure that the object name is the same as it appears in fluidPage().
2. What you save into the output object should be something build with the **render** function. Render builds reactive output to display on the UI.

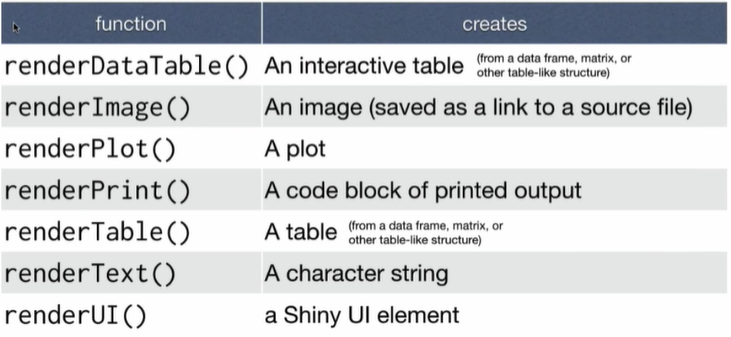
For example:



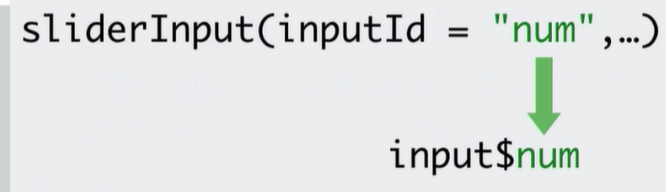
In RShiny, braces are used to pass the code as a unified code block. They allow you to put as many lines of code as you want into the same code block to pass to renderPlot. You could put an entire Rscript in those braces.



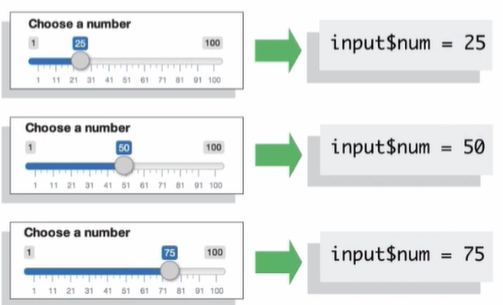
A family of render functions…

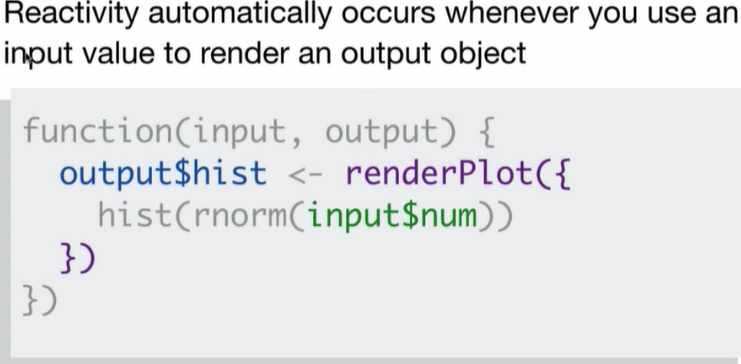


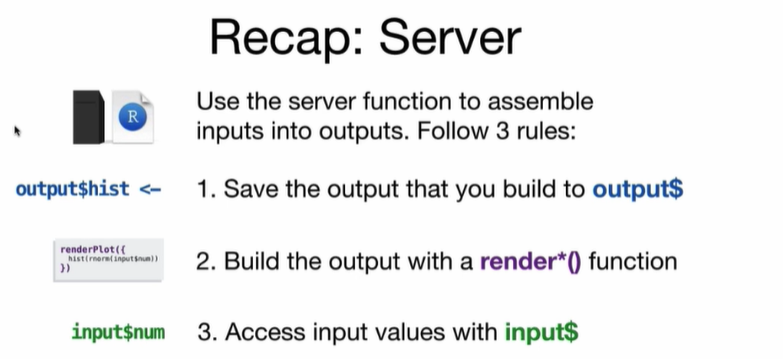
1. Use input values with **input$**



The input value changes whenever a user changes the input.







[Shiny - Tutorial (rstudio.com)](https://shiny.rstudio.com/tutorial/) at 24 mins (moving beyond the basics)