## render()

Render functions build output to display in the app. Render function make objects to display.

|  |  |
| --- | --- |
|  |  |

renderPlot({hist(rnorm(input$num))})

* Object will respond to every reactive value in the code.
* Code used to build (and rebuild) the object.

When notified it is invalid, the object created by the render function will rerun the entire block of code associated with it.

E.G In the two inputs example, there is 2 input objects. Everytime we update either one of the input files, the output app is updated. Think of it as the render function getting the input values.

## reactive()

Data=reactive({rnorm(input$num)})

Calling rnorm twice in this case as shown in the file 2 outputs. Instead, what we want is to output both objects with the same values. We want to avoid computation when the code is up to date. It just passes on the results.

* Reactive makes an object to use.
* Reactive expressions are themselves reactive.
* Call a reactive expression like a function.

Builds a reactive object (reactive expression):

1. You call a reactive expression like a function
2. Reactive expressions cache their values

## Isolate()

Creates output that is non-reactive. Returns the result as a non-reactive value.

isolate({rnorm(input$num)})

E.G. In our file 2\_1, we want the line to function such that it does not constantly update whenever we make any changes. Finding a way to prevent the field from updating immediately.

Note the changes we made in file 2\_1.

* The plot does not have dependency on the title.
* However, there is still reactivity with the num

## TriggerCode()

observeEvent triggers code to run on the server. For example, it helps to track a button.

observeEvent(input$clicks, {print(input$clicks)})

* Reactive values to respond to
* Code block to run whenever observer is invalidated
* Observer treats this as if it has been isolated with isolate()

Your app shouldn’t rely on action button

Use observe for more implicit syntax.

Observe({print(inpuit$clicks)})

* Observer will respond to every reactive value in the code.
* Code block to run whenever observer is invalidated.

## DelayReactions()

From file 2\_1, we want to create a button that changes the output when we press it.

Data= reactive(input$go, {rnorm(input$num)}

* Reactive values to respond to
* Code used to build object

## ManageState