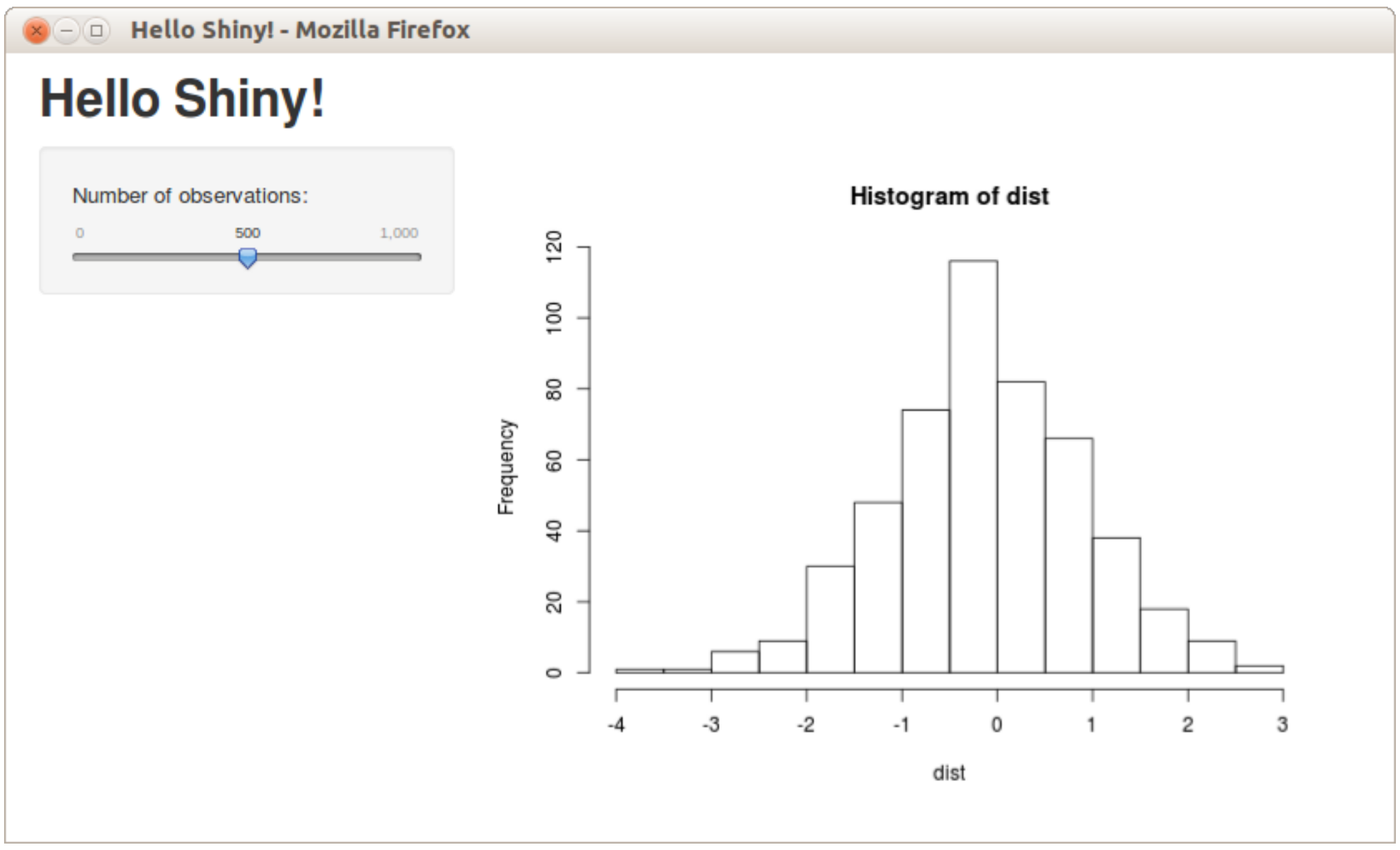
This tutorial is deprecated. Learn more about Shiny at our new location, shiny.rstudio.com.



The Hello Shiny example is a simple application that generates a random distribution with a configurable number of observations and then plots it. To run the example, type:

```
> library(shiny)
> runExample("01_hello")
```

Shiny applications have two components: a user-interface definition and a server script. The source code for both of these components is listed below.

In subsequent sections of the tutorial we'll break down all of the code in detail and explain the use of "reactive" expressions for generating output. For now, though, just try playing with the sample application and reviewing the source code to get an initial feel for things. Be sure to read the comments carefully.

The user interface is defined in a source file named ui.R:

ui.R

```
library(shiny)

# Define UI for application that plots random distributions
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("Hello Shiny!"),

  # Sidebar with a slider input for number of observations
  sidebarPanel(
    sliderInput("obs",
                "Number of observations:",
                min = 1,
                max = 1000,
                value = 500)
  ),

  # Show a plot of the generated distribution
  mainPanel(
    plotOutput("distPlot")
  )
))
```

The server-side of the application is shown below. At one level, it's very simple–a random distribution with the requested number of observations is generated, and then plotted as a histogram. However, you'll also notice that the function which returns the plot is wrapped in a call to `renderPlot`. The comment above the function explains a bit about this, but if you find it confusing, don't worry–we'll cover this concept in much more detail soon.

server.R

```
library(shiny)

# Define server logic required to generate and plot a random distribution
shinyServer(function(input, output) {

  # Expression that generates a plot of the distribution. The expression
  # is wrapped in a call to renderPlot to indicate that:
  #
  #  1) It is "reactive" and therefore should be automatically
  #     re-executed when inputs change
  #  2) Its output type is a plot
  #
  output$distPlot <- renderPlot({

    # generate an rnorm distribution and plot it
    dist <- rnorm(input$obs)
    hist(dist)
  })
})
```

The next example will show the use of more input controls, as well as the use of reactive functions to generate textual output.

← Previous Next →