

COMPUTACIÓN ESTADÍSTICA CON R

CLASE 5

RUBÉN SOZA

INTRODUCCIÓN A SHINYR

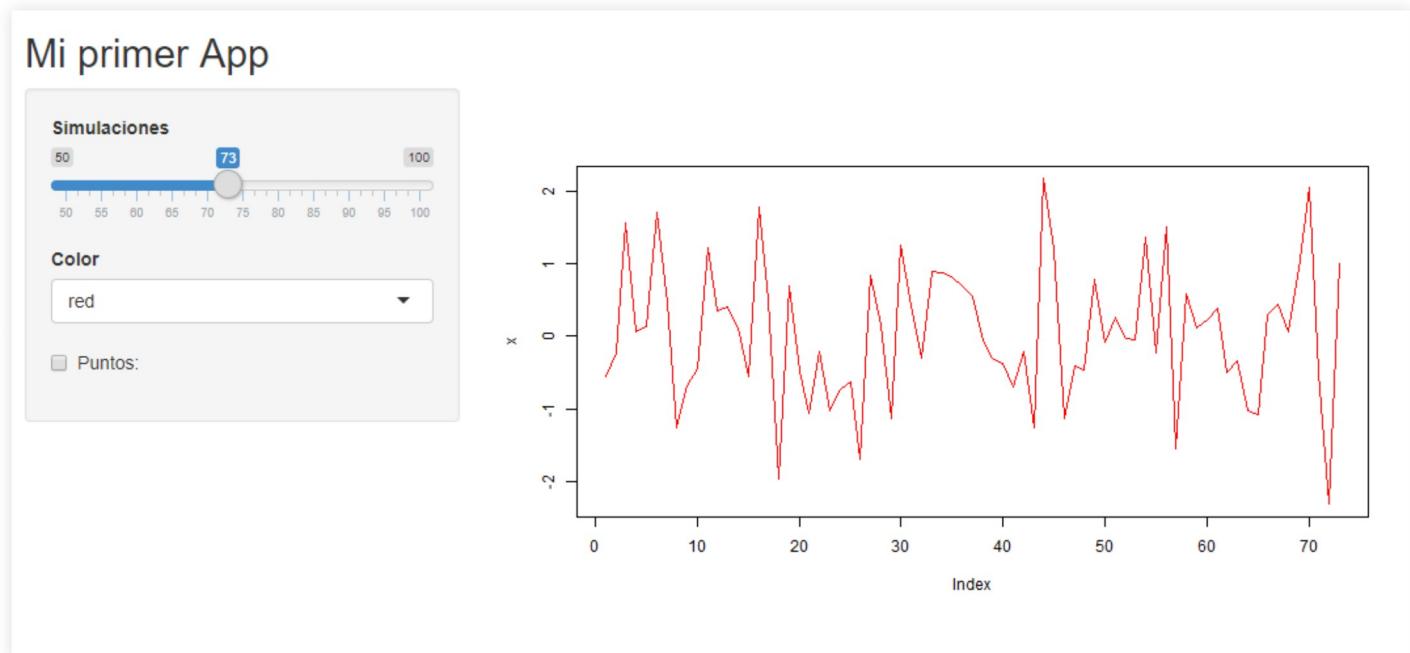
ANTES DE PARTIR

Necesitaremos algunos paquetes:

```
install.packages(  
  c("tidyverse", "shiny", "shinythemes", "shinyWidgets",  
    "shinydashboard", "DT", "leaflet", "plotly")  
)
```

¿QUÉ ES UNA APP(LICACIÓN) WEB?

(Wikipedia:) Herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador.



APLICACIÓN WEB

Las apps poseen **inputs** y **outputs**:

Veamos el siguiente link: <https://ruben-soza.shinyapps.io/actividades/>

LA ESTRUCTURA DE UNA SHINYAPP

```
library(shiny)

ui <- fluidPage()

server <- function(input, output) { }

runApp(list(ui = ui, server = server))
```

LA ESTRUCTURA DE UNA SHINYAPP

```
library(shiny)  
ui <- fluidPage()  
server <- function(input, output) {}  
runApp(list(ui = ui, server = server))
```

- » Se define una interfaz de usuario (user interface). En adelante **ui**
- » En este caso es una página fluida vacía **fluidPage()**
- » En el futuro acá definiremos diseño/estructura de nuestra aplicación (*layout*). Que se refiere la disposición de nuestros **inputs** y **outputs**

LA ESTRUCTURA DE UNA SHINYAPP

```
library(shiny)  
ui <- fluidPage()  
server <- function(input, output) {}  
runApp(list(ui = ui, server = server))
```

- » Se define el **server** en donde estará toda la lógica de nuestra aplicación.
- » Principalmente serán instrucciones que dependeran de **inputs**

LA ESTRUCTURA DE UNA SHINYAPP

```
library(shiny)  
ui <- fluidPage()  
server <- function(input, output) {}  
runApp(list(ui = ui, server = server))
```

- » **runApp** es la función que crea y deja corriendo la app con los parámetros otorgados.
- » **No siempre** tendremos que escribirla pues veremos que RStudio al crear una shinyApp nos pondrá un botón para *servir* la aplicación

EJERCICIO:

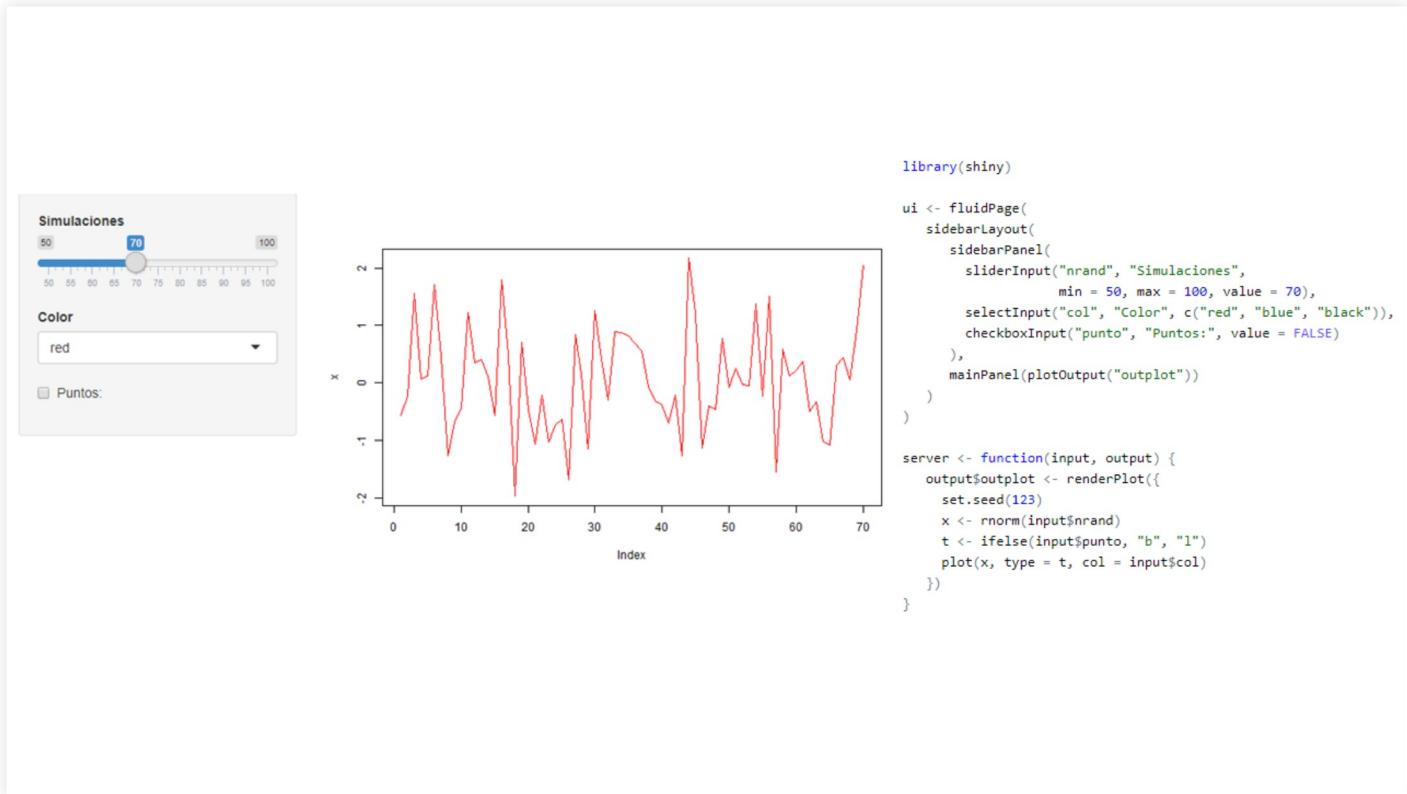
Hacer funcionar el código `mi_primer_shiny.R` en R Rstudio:

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                  min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

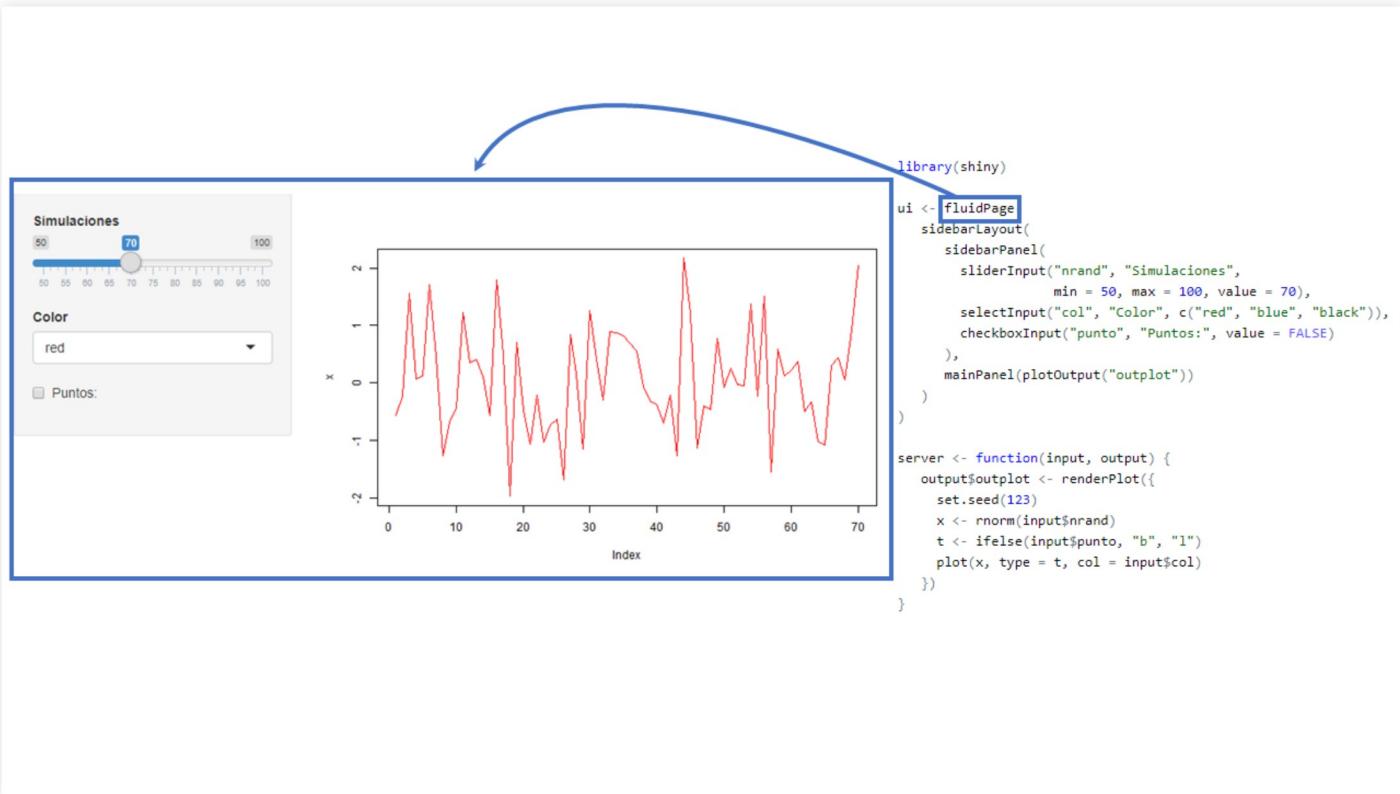
server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
  })
}
```

FUNCIONAMIENTO DE UNA APP DE SHINY

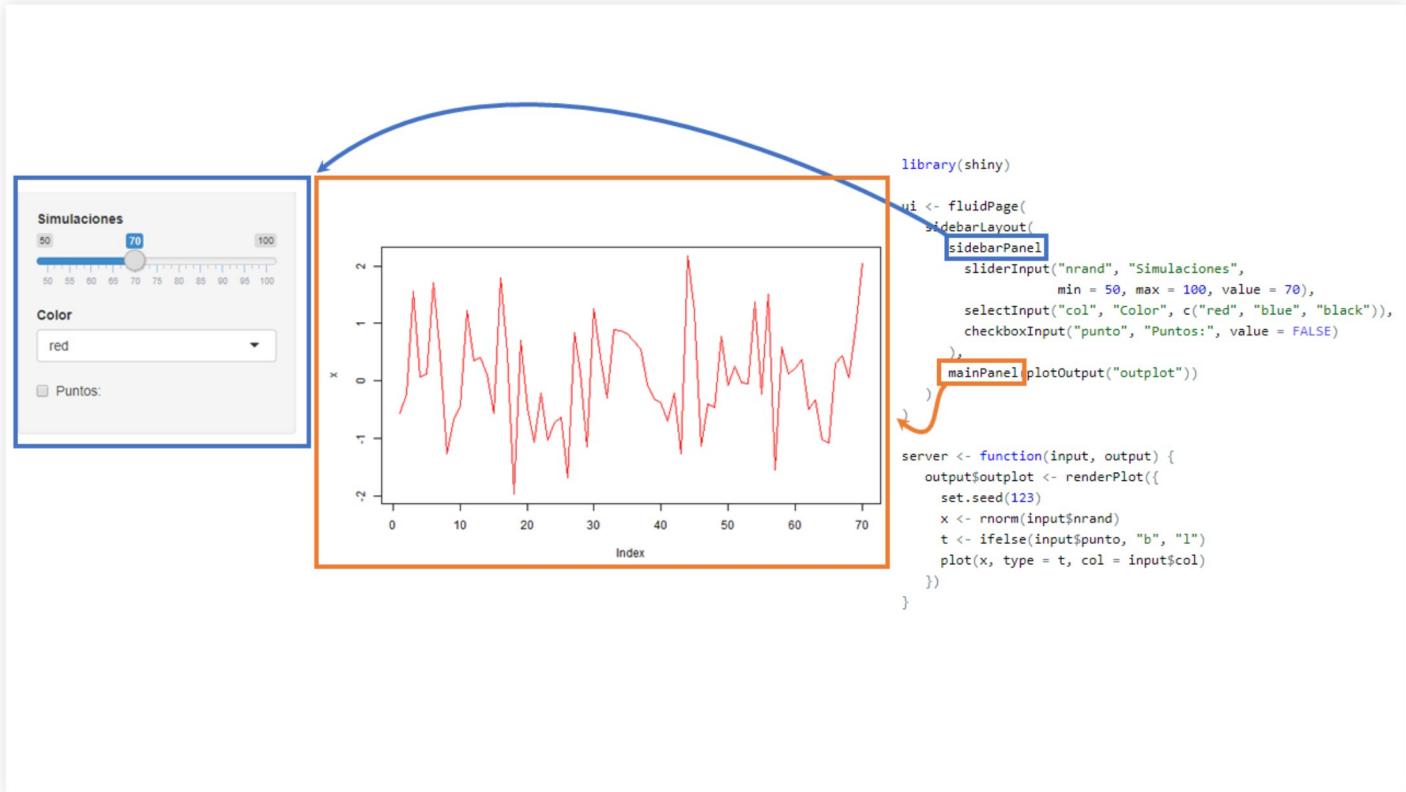
APP



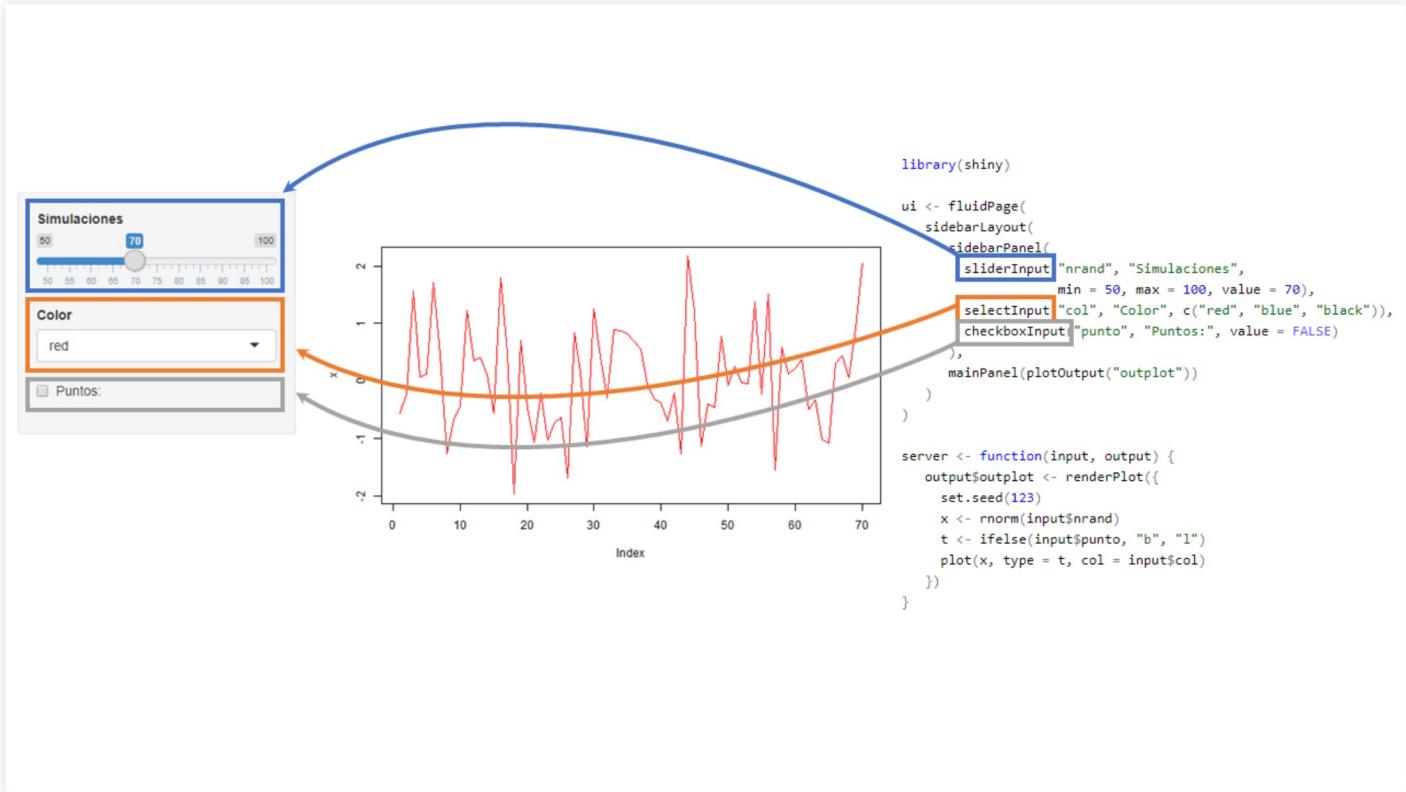
CONTENEDOR



OTROS CONTENEDORES



INPUTS



OUTPUTS

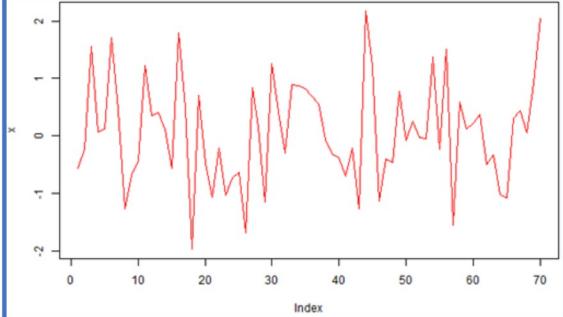
Simulaciones

50 55 60 65 70 75 80 85 90 95 100

Color

red

Puntos:

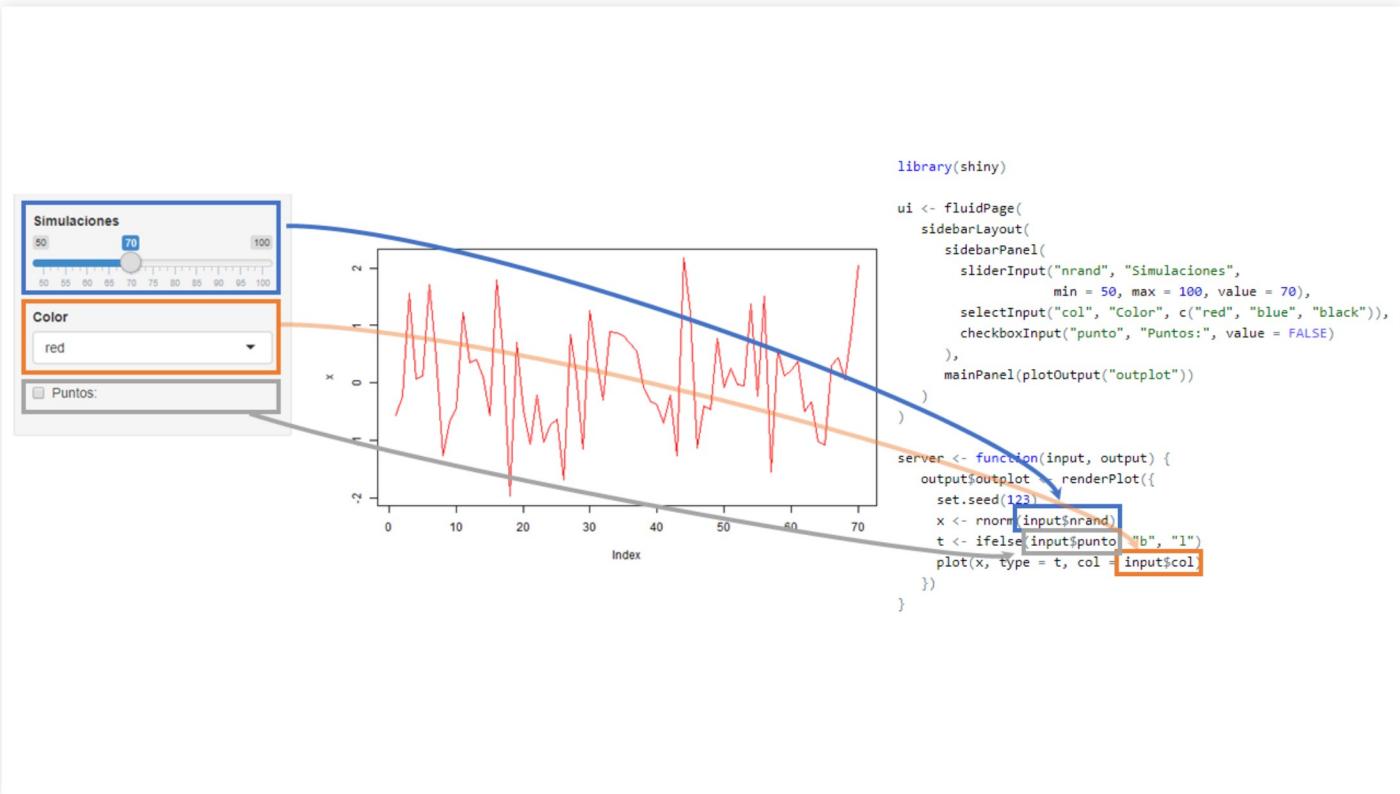


```
library(shiny)

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}
```

INTERACCIÓN



RESULTADO

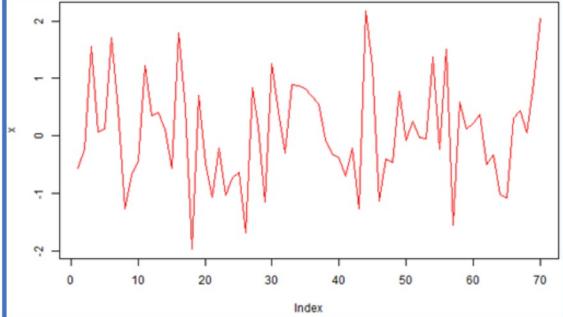
Simulaciones



Color

red

Puntos:



```
library(shiny)

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}
```

LA ESTRUCTURA DE UNA SHINYAPP 2

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}

```

- » `fluidPage`, `sidebarLayout`, `sidebarPanel`, `mainPanel` definen el diseño/*layout* de nuestra app
- » Existen muchas más formas de organizar una app. Más detalles en <http://shiny.rstudio.com/articles/layout-guide.html>

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}

```

- » `sliderInput`, `selectInput`, `checkboxInput` son los inputs de nuestra app, con esto el usuario puede interactuar con nuestra aplicación.
- » Estas funciones generan el input deseado en la app y shiny permite que los valores de estos inputs sean usados como valores usuales en R en la parte del server (numéricos, strings, booleanos, fechas).

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}

```

- » `plotOutput` define el lugar donde la salida estará
- » Como mencionamos, nuestras app pueden tener muchos outputs: tablas, texto, imágenes

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}

```

- » `renderPlot` define un tipo de salida gráfica
- » Existen otros tipos de salidas, como tablas `tableOutput` o tablas más interactivas como `DT::DTOutput`

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}

```

- » Este espacio determina la lógica de nuestra salida
- » Acá haremos uso de los inputs para entregar lo que deseamos

INTERACCIÓN ENTRE INPUTS Y OUTPUTS

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}

```

- » Las funciones `Output()` y `render()` trabajan juntas para agregar salidas de R a la interfaz de usuario
- » En este caso `renderPlot` esta asociado con `plotOutput` (¿cómo?)
- » Existen más parejas: `renderText/textOutput` o `renderTable/tableOutput`.

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}

```

- » Cada **Output()** y **render()** se asocian con un **id**
- » Este **id** debe ser único en la aplicación
- » En el ejemplo **renderPlot** esta asociado con **plotOutput** vía el **outplot**

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}

```

- » Cada función **Input** requiere un **id** para ser identificado en el server
- » Cada **Input** requiere argumentos específicos a cada tipo de input, valor por defecto, etiquetas, opciones, rangos, etc

- » Acá, el valor numérico ingresado/modificado por el usuario se puede acceder en el server bajo `input$nrand`

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      sliderInput("nrand", "Simulaciones",
                 min = 50, max = 100, value = 70),
      selectInput("col", "Color", c("red", "blue", "black")),
      checkboxInput("punto", "Puntos:", value = FALSE)
    ),
    mainPanel(plotOutput("outplot"))
  )
)

server <- function(input, output) {
  output$outplot <- renderPlot({
    set.seed(123)
    x <- rnorm(input$nrand)
    t <- ifelse(input$punto, "b", "l")
    plot(x, type = t, col = input$col)
  })
}

```

- » **sliderInput** se usa para seleccionar un valor numérico entre un rango
- » **selectInput** otorga la posibilidad que el usuario escoge entre un conjunto de valores
- » **checkboxInput** en el server es un valor lógico TRUE/FALSE

- » ¿Necesitas más? <https://gallery.shinyapps.io/065-update-input-demo/> y `shinyWidgets::shinyWidgetsGallery()`
- » Si deseas compartir tu ShinyApp puede ser útil
<https://shiny.rstudio.com/tutorial/written-tutorial/lesson7/>

**EJEMPLO: ANÁLICEMOS UN EJEMPLO EN
SHINY !**

EJERCICIO 1

- » Haga click en *File*, luego *New File* y *Shiny Web App*, seleccione el nombre
- » Ejecutela con *Run App* e interactúe
- » Luego modifique y cree una app que contenga:
- » 2 inputs, un `sliderInput` y un `textInput`
- » 3 output de tipo texto `textOutput` donde el primer contenga el valor del primer input, el segundo el valor del segundo input, y el tercero la suma de los dos inputs

**ACTIVIDAD PRÁCTICA II:
DESCARGUE EL ARCHIVO DESDE
LA PLATAFORMA !**