

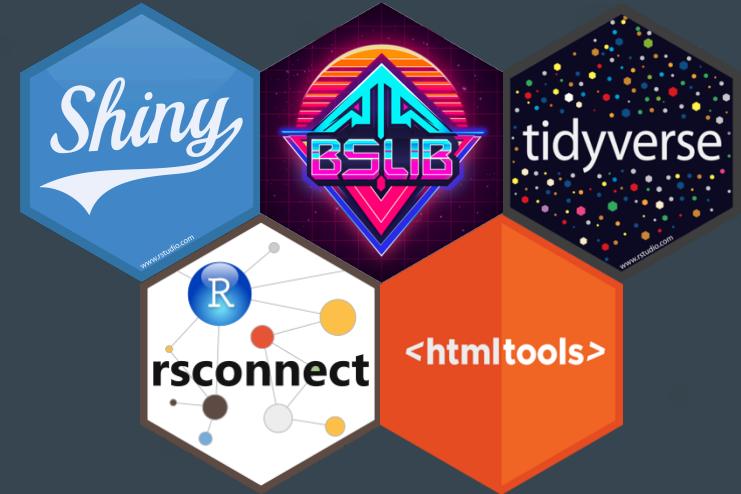
Shiny

Visualizacion de datos con R

Slides 1

Diplomado en Data Science, MatPUC

Joshua Kunst Fuentes jbkunst@gmail.com <https://jkunst.com/blog/>



Shiny: Visualizacion de datos con R

Version 2025

Clases

- Martes 18 de noviembre
- Jueves 20 de Noviembre

Programa

- Slides 1
 - Aplicación (web), ejemplos.
 - Introducción a shiny: Interfaz usuario y servidor
- Slides 2
 - Layouts
 - Integración HTMLWidgets
- Slides 3
 - Temas, templates y diseño
 - Compartir una app
 - Shiny en Python
- Slides 4
 - Expresiones reactivas
 - Orden de ejecución
 - Extensiones shiny

Antes de Partir

Antes de Partir

La prestación podrá acceder desde <https://jkunst.com/shiny-visualizacion-de-datos-con-R/clase-01.html> y el código fuente, apps, ejemplos en <https://github.com/jbkunst/shiny-visualizacion-de-datos-con-R>

Asumimos que tenemos conocimiento de como funciona R, paquetes, funciones, etc.

No es necesario en `shiny` pero usaremos los paquetes `dplyr` y `ggplot2` principalmente para hacer manipulación y visualización de los datos.

Necesitaremos algunos paquetes:

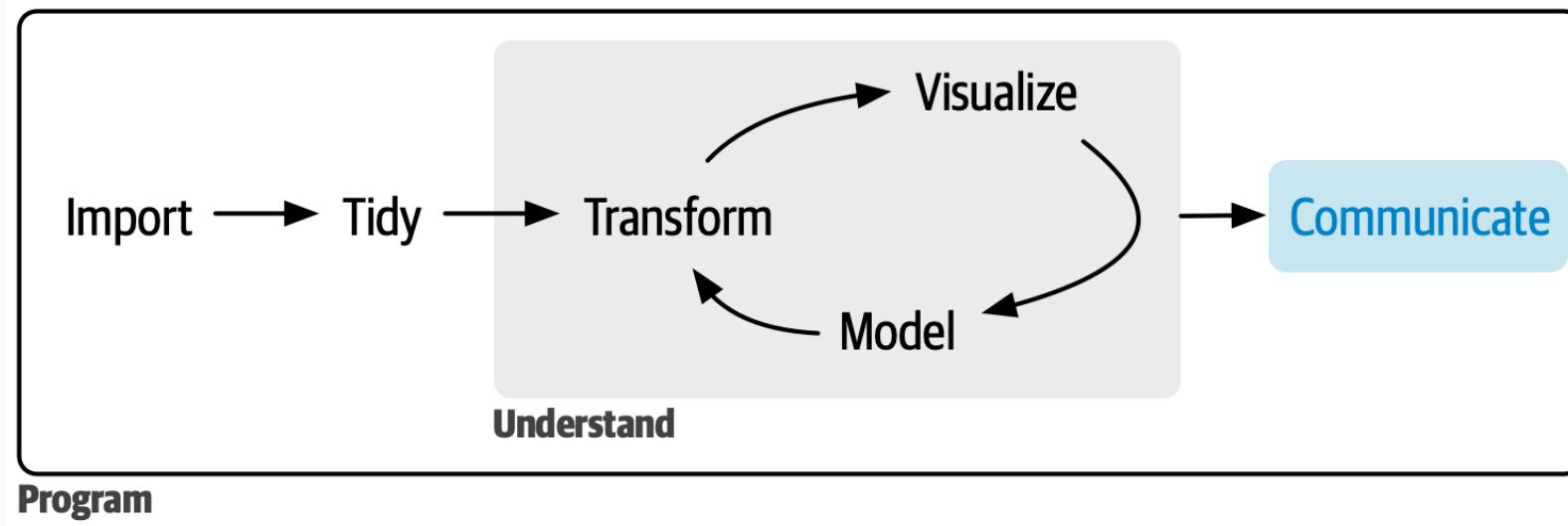
```
install.packages(  
  c("tidyverse", "shiny", "shinythemes", "shinywidgets", "bslib",  
    "shinydashboard", "DT", "leaflet", "plotly", "highcharter",  
    "forecast", "janitor")  
)
```

Ayuda

No olvidar que una buena forma de aprender es con la documentación oficial:

- <https://shiny.posit.co/>
- <https://mastering-shiny.org/>
- <https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/>

Por que shiny?



Por que shiny?

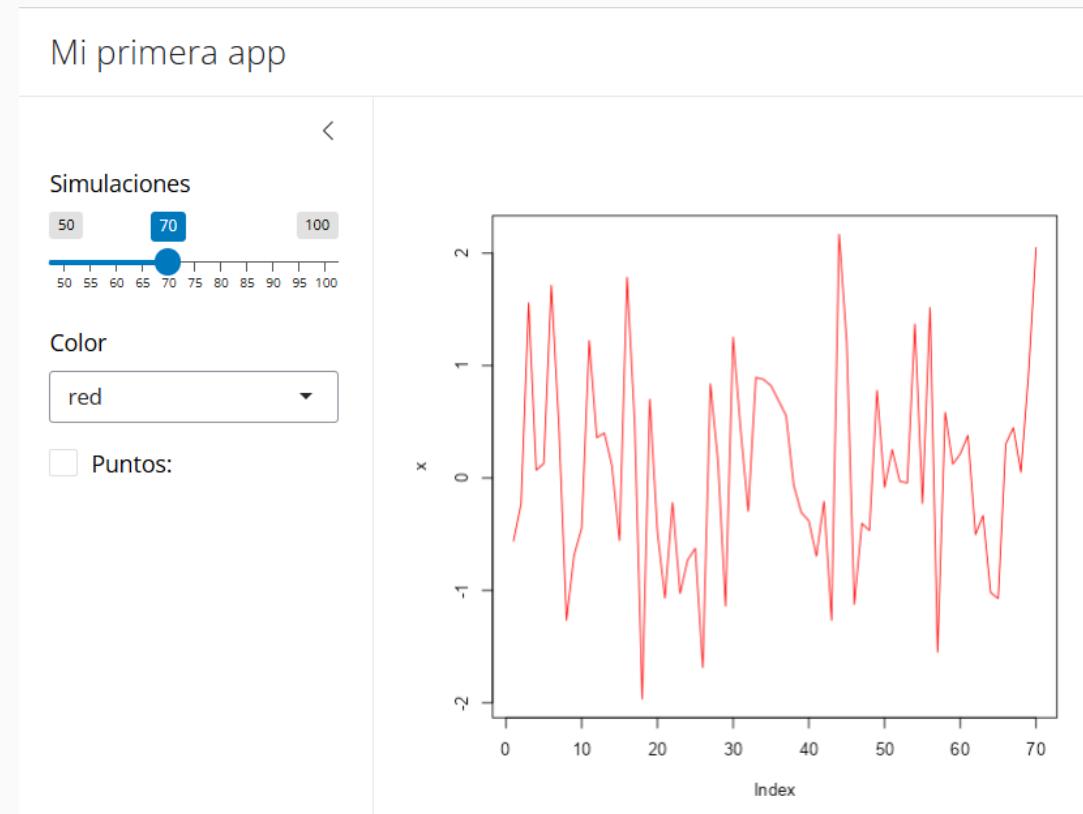
- **De script a app en minutos.** Lo que ya tienes en R/Python lo envuelves en una interfaz sin crear APIs ni reescribir lógica.
- **Sin cambiar de lenguaje ni stack.** Sigues usando R/Python – no necesitas aprender React, JS, Flask, Django, etc.
- **Interactividad analítica inmediata.** La reactividad está integrada: los outputs se actualizan solos al cambiar los inputs.

¿Qué es una **app**(licación) web?

Aplicación Web

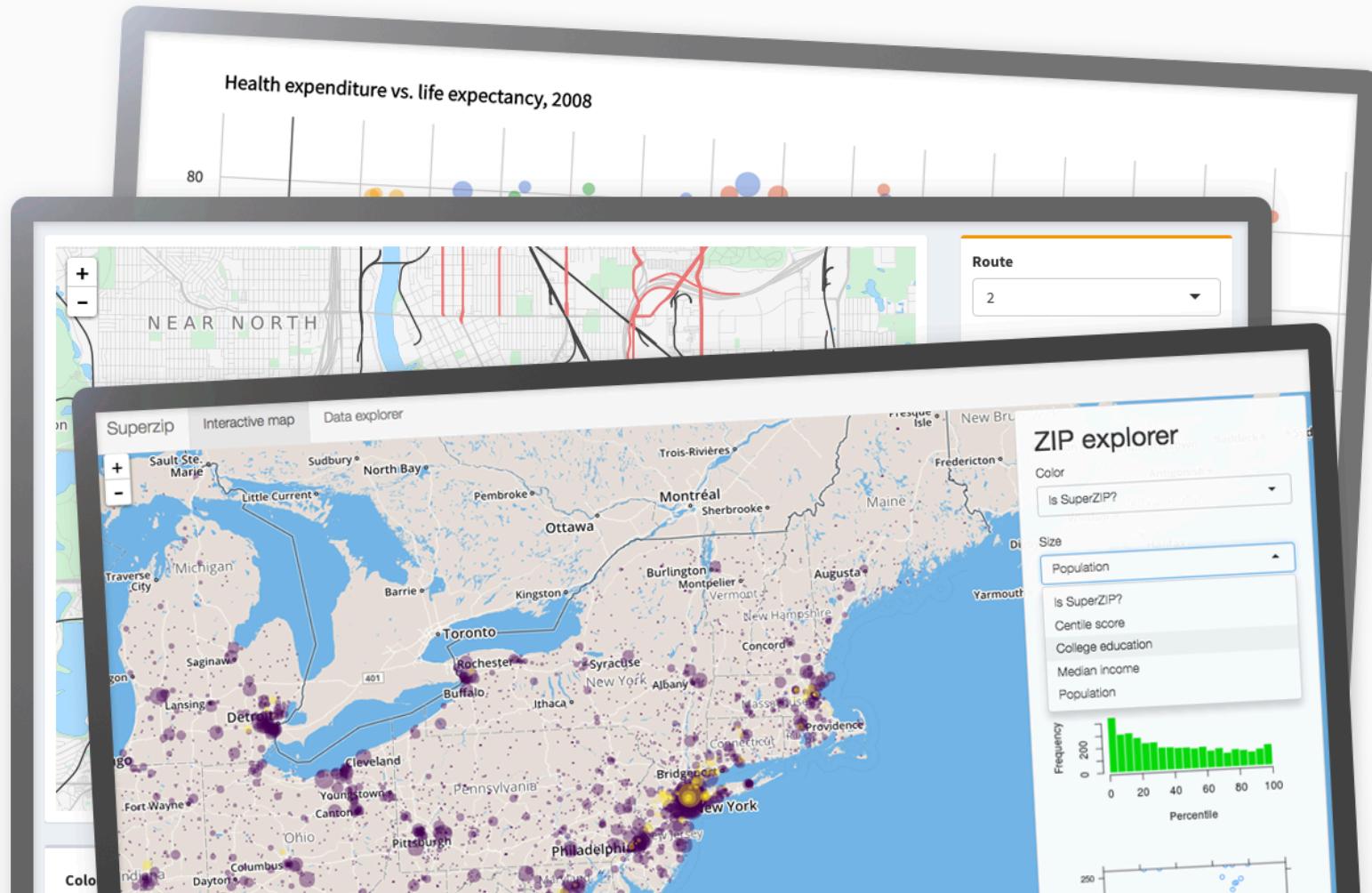
(Wikipedia:) Herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador.

Puede ser de lo más simple:



Aplicación Web

Hasta algo más complejo con más **inputs** y **outputs**:



Ejemplos para motivarse!

Algunos simples.

- <https://jjallaire.shinyapps.io/shiny-kmeans/>
- <https://jbkunst.shinyapps.io/bias-variance/> (<https://github.com/jbkunst/shiny-apps-educational>)

Otros con más detalle en la parte visual.

- <https://indice.nudos.cl/>
- <https://odes-chile.org/app/unidades/>
- <https://jbkunst.shinyapps.io/shiny-sidebot-demo-school/>
- <https://jbkunst.shinyapps.io/trd-sttstcs/> (<https://github.com/jbkunst/trd-sttstcs>)
- <https://jorgehc1998.shinyapps.io/Dataton-app/> (<https://github.com/socapal/dataton-tudinero>)
- <https://nz-stefan.shinyapps.io/commute-explorer-2>

La estructura de una ShinyApp

```
library(shiny)
library(bslib) # shiny incluye el paquete bslib, que usaremos para crear interfaces

ui <- page_fluid()

server <- function(input, output) {}

shinyApp(ui, server)
```

En `shiny`, una aplicación constará de 2 partes:

- La interfaz de usuario, `ui` (user interface), donde definiremos el look de nuestra aplicación, y lugar de `inputs` y `outputs`.
- El `server`, en donde especificaremos como interactuan los `outputs` en función de los `inputs`.

La estructura de una ShinyApp

```
library(shiny)
library(bslib)

ui <- page_fluid()

server <- function(input, output) {}

shinyApp(ui, server)
```

- Se define una interfaz de usuario (user interface). En adelante `ui`.
- En este caso es una página fluida vacía `page_fluid()`.
- 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)
library(bslib)

ui <- page_fluid()

server <- function(input, output) {}

shinyApp(ui, server)
```

- Se define el `server` en donde estará toda la lógica de nuestra aplicación.
- Principalmente serán instrucciones que dependerán de `inputs` y reflejaremos `outputs`: como tablas, gráficos.

La estructura de una ShinyApp

```
library(shiny)
library(bslib)

ui <- page_fluid()

server <- function(input, output) {}

shinyApp(ui, server)
```

- `shinyApp` 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.

La estructura de una ShinyApp

```
library(shiny)
library(bslib)

ui <- page_fluid()

server <- function(input, output) {}

shinyApp(ui, server)
```

De forma general la aplicación será:

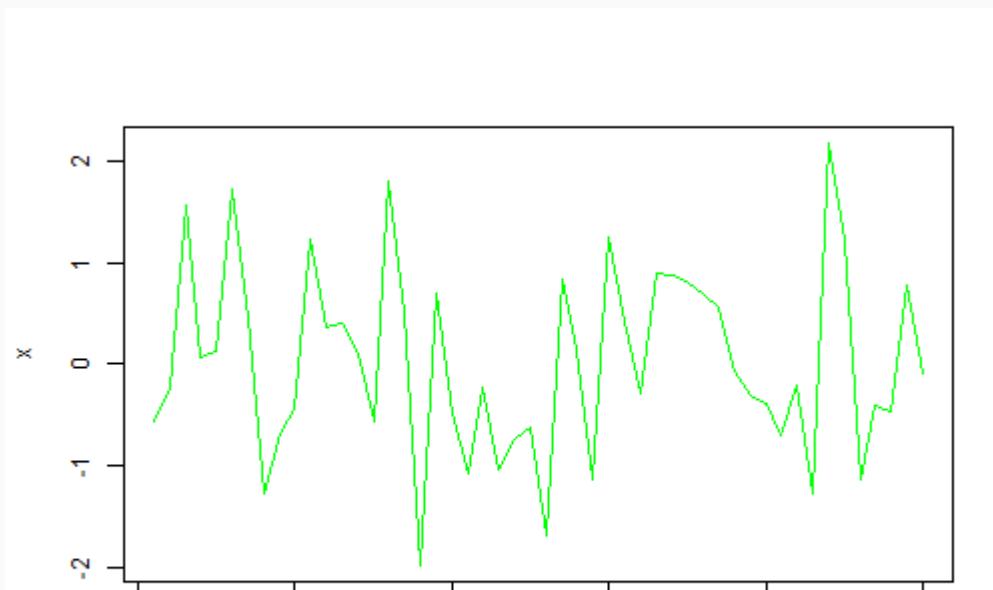
```
library(shiny)
library(bslib)
# acá se cargarán paquetes y también datos necesarios

ui <- page_fluid(
  # código que da forma a nuestra aplicación: títulos, secciones, textos, inputs
)

server <- function(input, output) {
  # toda la lógica de como interactúan los outputs en función de los inputs
}
```

Repasso R: Que hace el siguiente código?

```
set.seed(123)
x <- rnorm(50)
x
t <- ifelse(FALSE, "b", "l")
t
c <- "green"
c
plot(x, type = t, col = c) ## [1] "l"
## [1] "green"
```



Ejercicio: Nuestra primer App

Hacer funcionar el siguiente **código** en R Rstudio: (hint: sí, copy + paste + run).

Note que al *guardar* el script Rstudio lo reconoce como app y aparece un botón de RUN.

```
library(shiny)
library(bslib)

ui <- page_sidebar(
  title = "Mi primer app",
  sidebar = sidebar(
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),
    selectInput("col", "Color", c("red", "blue", "black")),
    checkboxInput("punto", "Puntos:", value = FALSE)
  ),
  plotOutput("grafico")
)

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

shinyApp(ui, server)
```

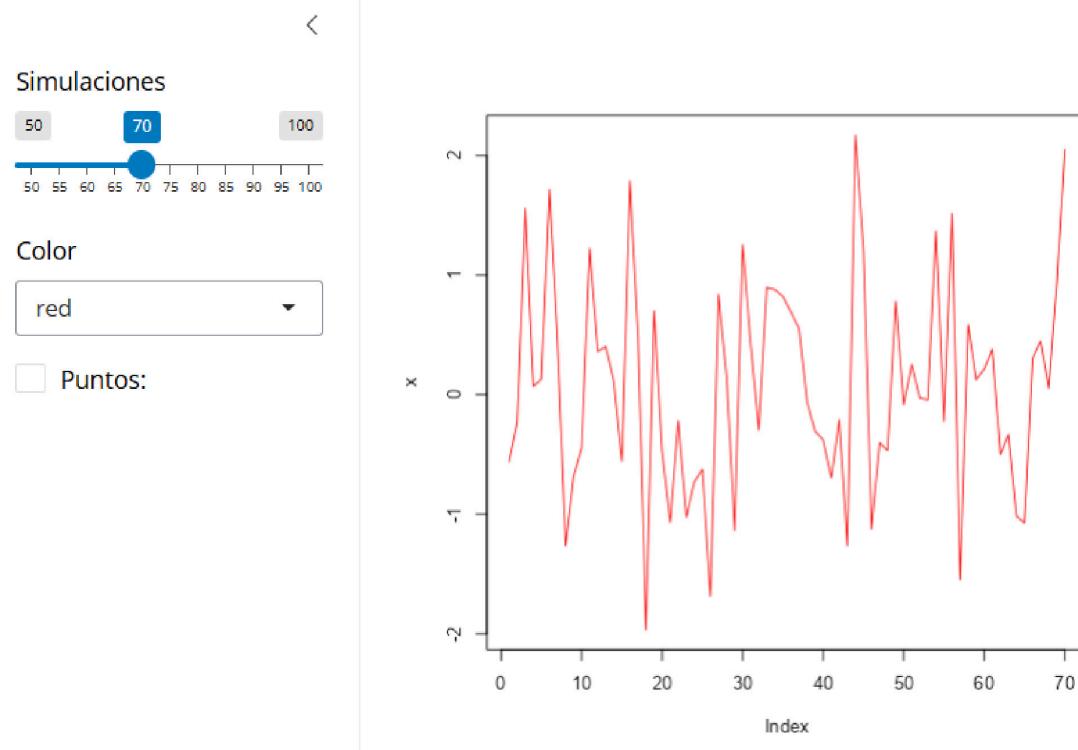
Si quieres más ejemplos

```
shiny::runExample("01_hello")      # un histograma
shiny::runExample("02_text")        # tablas y data frames
shiny::runExample("03_reactivity")  # una expresión reactiva
shiny::runExample("04_mpg")         # variables globales
shiny::runExample("05_sliders")     # controles deslizantes (sliders)
shiny::runExample("06_tabs")        # paneles con pestañas (tabs)
shiny::runExample("07_widgets")     # texto de ayuda y botones de envío
shiny::runExample("08_html")        # app Shiny construida desde HTML
shiny::runExample("09_upload")      # asistente para subir archivos
shiny::runExample("10_download")    # asistente para descargar archivos
shiny::runExample("11_timer")       # temporizador automático
```

Funcionamiento de una app de Shiny

App

Mi primera app



```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
,  
    plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$nrand)  
    t <- ifelse(input$punto, "b", "l")  
    plot(x, type = t, col = input$col)  
  })  
}
```

Página/layout

Mi primera app

<

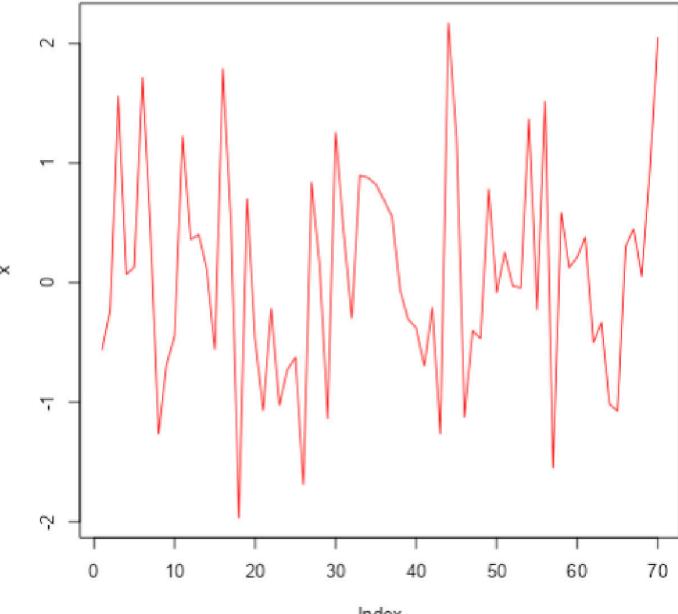
Simulaciones

50 70 100

Color

red

Puntos:



The screenshot shows a Shiny application interface. On the left, there's a sidebar with a slider for 'Simulaciones' set to 70, a dropdown menu for 'Color' set to 'red', and a checkbox for 'Puntos:' which is unchecked. The main area contains a plot with a red jagged line representing a simulated signal over an 'Index' from 0 to 70.

```
ui <- page_sidebar
title = "Mi primer app",
sidebar = sidebar(
  sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),
  selectInput("col", "Color", c("red", "blue", "black")),
  checkboxInput("punto", "Puntos:", value = FALSE)
),
plotOutput("grafico")
)

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

A red arrow points from the 'page_sidebar' line in the R code to the sidebar component in the application screenshot. Another red box highlights the 'page_sidebar' line in the code.

23 / 45

Elementos

Mi primera app

Simulaciones

50 70 100

Color

red

Puntos:

grafico

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar()  
  sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
  selectInput("col", "Color", c("red", "blue", "black")),  
  checkboxInput("punto", "Puntos:", value = FALSE)  
,  
  plotOutput("grafico")  
)  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$nrand)  
    t <- ifelse(input$punto, "b", "l")  
    plot(x, type = t, col = input$col)  
  })  
}
```

24 / 45

Inputs

Mi primera app



```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
,  
    plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$nrand)  
    t <- ifelse(input$punto, "b", "l")  
    plot(x, type = t, col = input$col)  
  })  
}
```

Output(s)

Mi primera app

Simulaciones

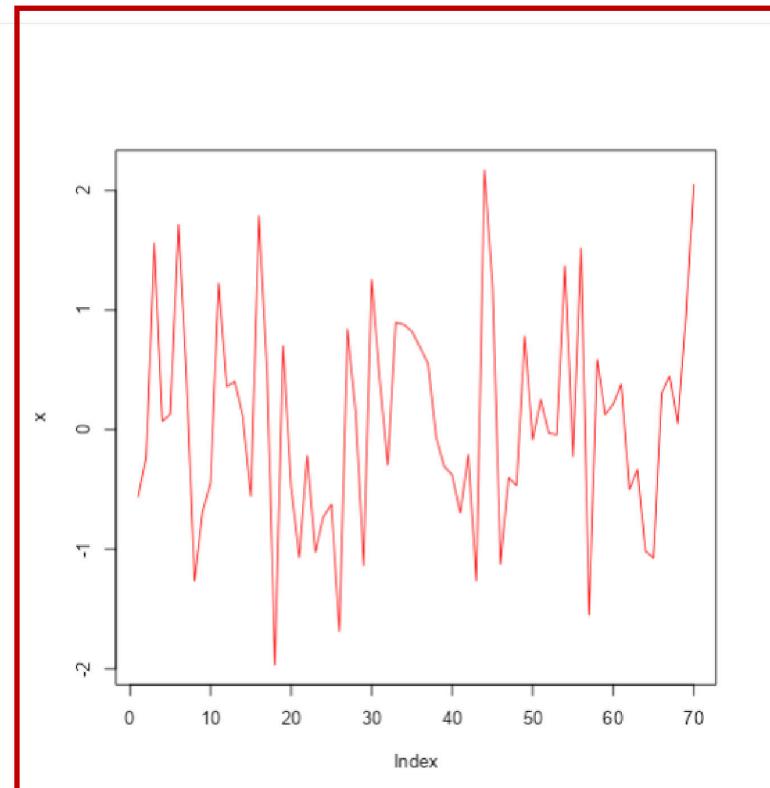
50 70 100

50 55 60 65 70 75 80 85 90 95 100

Color

red

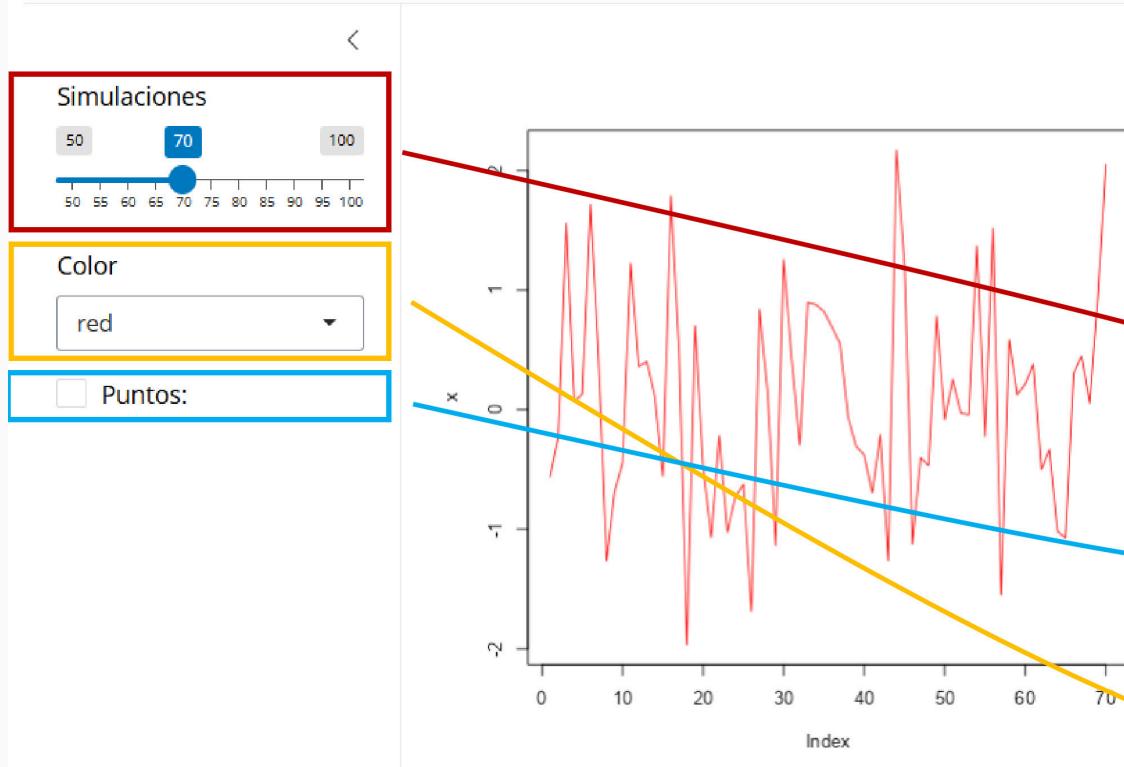
Puntos:



```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
)  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$nrand)  
    t <- ifelse(input$punto, "b", "l")  
    plot(x, type = t, col = input$col)  
  })  
}
```

Obtener valores en el ui del usuario el server

Mi primera app

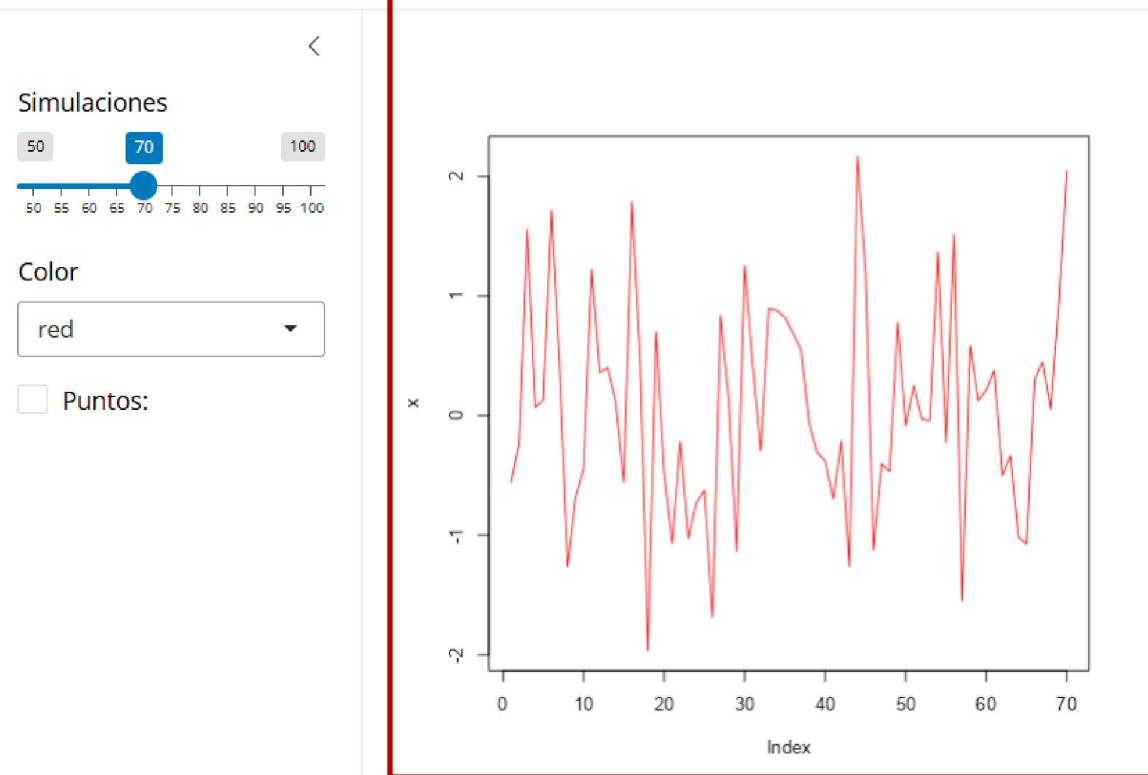


```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
)  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$nrand)  
    t <- ifelse(input$punto, "b", "l")  
    plot(x, type = t, col = input$col)  
  })  
}  
}
```

The code defines the user interface (ui) and the server logic. The ui consists of a sidebar with a slider for 'nrand' (value 70), a select input for 'col' (color red), and a checkbox for 'punto' (unchecked). The server logic generates a plot with a red noisy line and a blue trend line ('b' or 'l') based on the input parameters. Arrows point from the highlighted UI elements to the corresponding code lines in the server function.

Resultado de server a ui

Mi primera app



```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
)  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$nrand)  
    t <- ifelse(input$punto, "b", "l")  
    plot(x, type = t, col = input$col)  
  })  
}
```

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
  ),  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$nrand)  
    t <- ifelse(input$punto, "b", "l")  
    plot(x, type = t, col = input$col)  
  })  
}  
}
```

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
  ),  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$nrand)  
    t <- ifelse(input$punto, "b", "l")  
    plot(x, type = t, col = input$col)  
  })  
}  
}
```

- `page_sidebar`, `page_fluid`, `sidebar`, definen el diseño/*layout* de nuestra app.
- Existen muchas más formas de organizar una app: Por ejemplo uso de *tabs* de *menus*, o páginas con navegación. Más detalles <http://shiny.rstudio.com/articles/layout-guide.html>.

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
  ),  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$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 (<https://shiny.rstudio.com/gallery/widget-gallery.html>).
- 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).

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
,  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$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 ueden tener muchos outputs: tablas, texto, imágenes.

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
  ),  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$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`.

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
  ),  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$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.

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
  ),  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$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?)
- Hay muchas parejas como `renderText / textOutput` o `renderTable / tableOutput` entre otras (revisar la sección de outputs en el cheat sheet)

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
  ),  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$nrand)  
    t <- ifelse(input$punto, "b", "l")  
    plot(x, type = t, col = input$col)  
  })  
}  
}
```

- Cada `*output()` y `render*()` se asocian con un **id** definido por nosotros
- Este **id** debe ser único en la aplicación
- En el ejemplo `renderPlot` esta asociado con `plotOutput` vía el id `grafico`

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
  ),  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$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`

El funcionamiento de una ShinyApp

```
ui <- page_sidebar(  
  title = "Mi primer app",  
  sidebar = sidebar(  
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),  
    selectInput("col", "Color", c("red", "blue", "black")),  
    checkboxInput("punto", "Puntos:", value = FALSE)  
  ),  
  plotOutput("grafico")  
)  
  
server <- function(input, output) {  
  output$grafico <- renderPlot({  
    set.seed(123)  
    x <- rnorm(input$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 <http://shinyapps.dreamrs.fr/shinyWidgets/>

Agregando más outputs

```
library(shiny)
library(bslib)

datos <- rnorm(100)

ui <- page_sidebar(
  title = "Mi primer app",
  sidebar = sidebar(
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),
    selectInput("col", "Color", c("red", "blue", "black")),
    checkboxInput("punto", "Puntos:", value = FALSE)
  ),
  plotOutput("grafico")
)

server <- function(input, output) {
  output$grafico <- renderPlot({
    plot(head(datos, input$nrand), type = ifelse(input$punto, "b", "l"), col = input$col)
  })
}

shinyApp(ui, server)
```

- Asumamos que tenemos información en la variable `datos` y hemos reducido líneas para la parte del primer gráfico.
- Para agregar más outputs se deben incorporar el elemento `output` en el `ui` y luego el bloque de código asociado en el server como `output$id_del_output`.

Agregando más outputs

```
library(shiny)
library(bslib)

datos <- rnorm(100)

ui <- page_sidebar(
  title = "Mi primer app",
  sidebar = sidebar(
    sliderInput("nrand", "Simulaciones", min = 50, max = 100, value = 70),
    selectInput("col", "Color", c("red", "blue", "black")),
    checkboxInput("punto", "Puntos:", value = FALSE)
  ),
  plotOutput("grafico"),
  plotOutput("grafico2"),
  tableOutput("tabla")
)

server <- function(input, output) {
  output$grafico <- renderPlot({
    plot(head(datos, input$nrand), type = ifelse(input$punto, "b", "l"), col = input$col)
  })

  output$grafico2 <- renderPlot({
    hist(head(datos, input$nrand), col = input$col)
  })

  output$tabla <- renderTable({
    x <- head(datos, input$nrand)
    data.frame(mean(x), sd(x), length(x))
  })
}

shinyApp(ui, server)
```

Agregando más outputs

Es importante recordar:

- Los ids deben ser únicos.
- El tipo de cada output.
- No necesariamente debemos utilizar *todos* los inputs en cada output, por ejemplo la tabla no utiliza el color ni el si es punto/línea.
- Se pueden crear variables *intermedias* en cada bloque de output.
- Cada output retornará lo definido en la última línea de su bloque.

Ejercicio: Inputs y outputs vengan a mi!

Modifiquemos la última versión de la aplicación para que contenga:

- 2 inputs, un `sliderInput` y un `textInput`
- 3 outputs de tipo texto `textOutput` donde:
 1. El primero contenga simplemente el valor del primer input
 2. El segundo el valor del segundo input, y
 3. El tercero la suma de los dos inputs.

Solucion

```
library(shiny)
library(bslib)

ui <- page_sidebar(
  title = "Mi primer app",
  sidebar = sidebar(
    sliderInput("numerouno", "numero 1", min = 50, max = 100, value = 70),
    textInput("numerodos", "#2", value = 3)
  ),
  textOutput("resultado1"),
  textOutput("resultado2"),
  textOutput("resultado3")
)

server <- function(input, output) {

  output$resultado1 <- renderText({
    x <- input$numerouno
    x
  })

  output$resultado2 <- renderText({
    input$numerodos
  })

  output$resultado3 <- renderText({
    input$numerouno + input$numerodos
  })
}

shinyApp(ui, server)
```

Solucion (ver. 2)

```
library(shiny)
library(bslib)

ui <- page_sidebar(
  title = "Mi primer app",
  sidebar = sidebar(
    sliderInput("numerouno", "numero 1", min = 50, max = 100, value = 70),
    numericInput("numerodos", "#2", value = 3)
  ),
  textOutput("resultado1"),
  textOutput("resultado2"),
  textOutput("resultado3")
)

server <- function(input, output) {

  output$resultado1 <- renderText({
    x <- input$numerouno
    x
  })

  output$resultado2 <- renderText({
    input$numerodos
  })

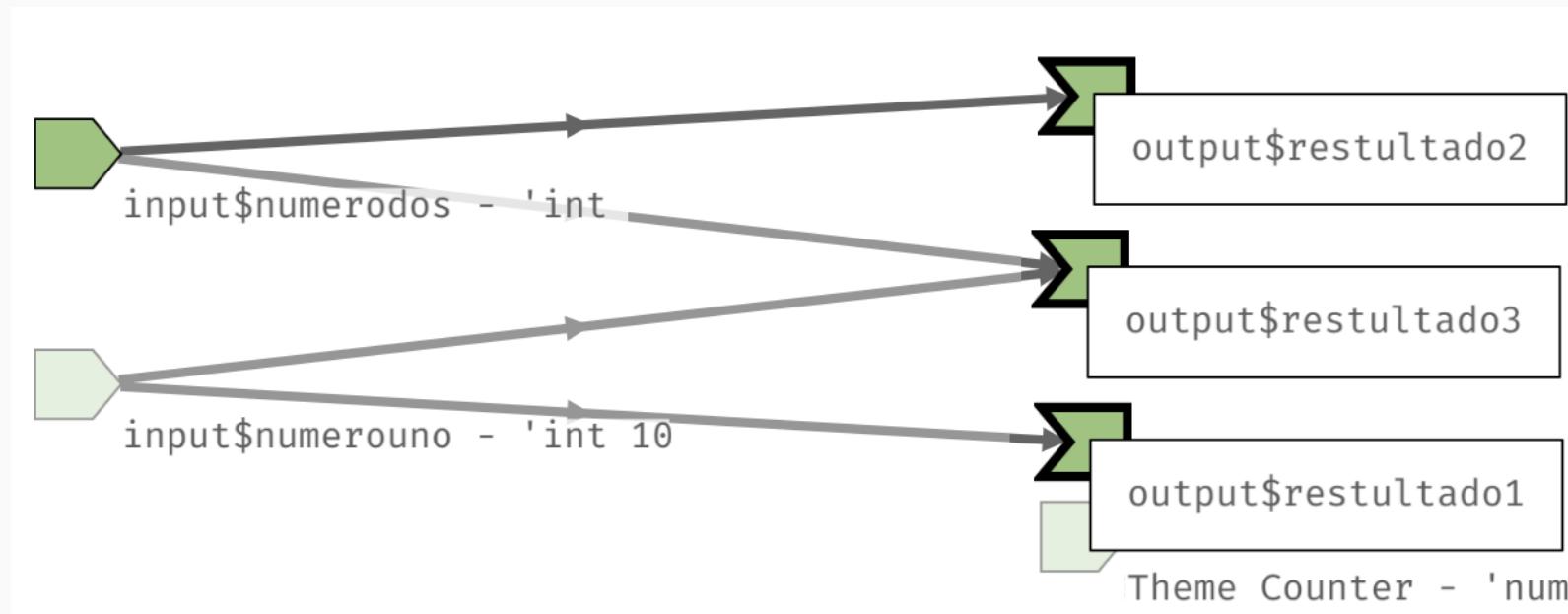
  output$resultado3 <- renderText({
    input$numerouno + as.numeric(input$numerodos)
  })
}

shinyApp(ui, server)
```

Reactividad: Como funciona shiny

Consideremos la aplicación del ejemplo anterior.

Al cambiar un input -como lo es `input$numerodos` o `input$numerouno`- shiny reconoce que expresiones (renders, como `renderText` en este caso) dependen dichos elementos y vuelve a calcularlos a penas suceda el cambio.



En este sentido, shiny funciona similar a excel.