Shiny: Visualizacion de datos con R

Diplomado en Data Science 22, MatPUC

Joshua Kunst Fuentes jbkunst@gmail.com

Version 2022

Johnson

Jue 21/7 - Sáb 23/7 - Jue 28/7 - Jue 4/8

Touring

Mie 27/7 - Mie 02/8 - Mie 10/8 - Mie 17/8

Programa

- Clase 1
  - Aplicación (web), ejemplos.
  - Introducción a shiny: Interfaz usuario y servidor
  - Reactividad
- Clase 2
  - Layouts
  - Integración HTMLWidgets
- Clase 3
  - Temas, templates y diseño
  - Compartir una app
- Clase 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:

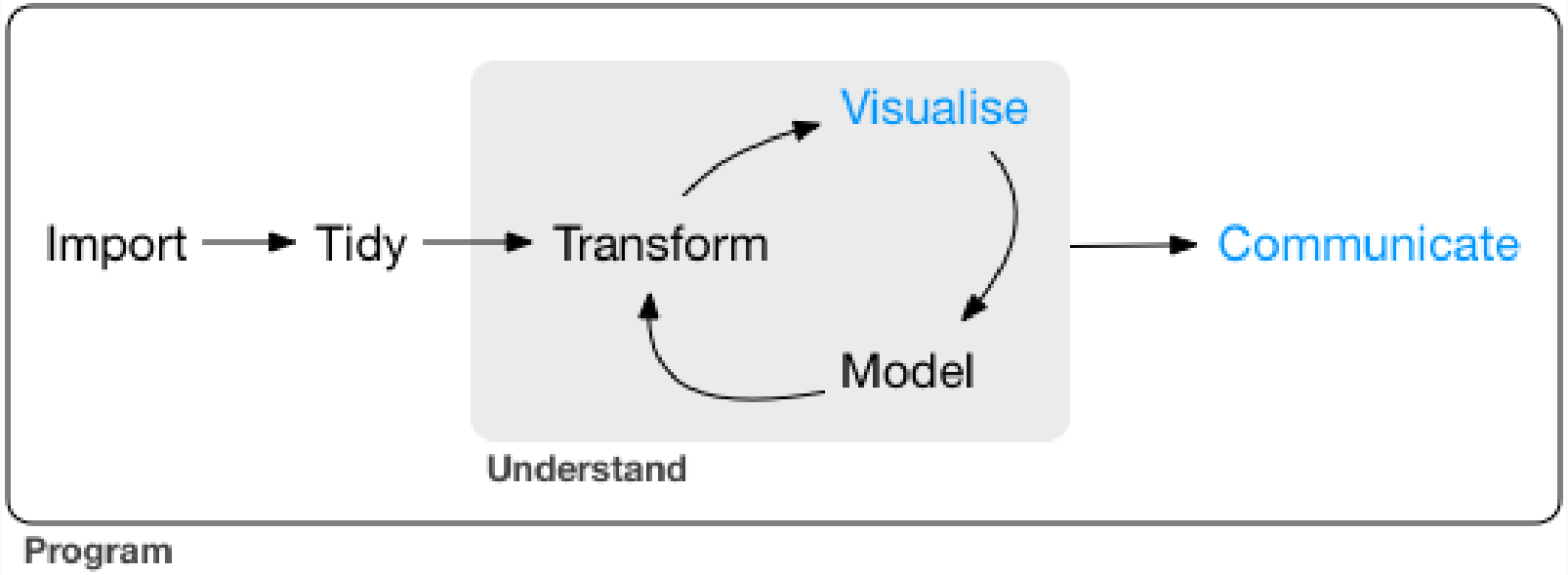
```
install.packages(
  c("tidyverse", "shiny", "shinythemes", "shinyWidgets",
    "shinydashboard", "DT", "leaflet", "plotly", "highcharter")
  )
```

Ayuda

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

- https://mastering-shiny.org/ (proceso de traducción al español https://github.com/cienciadedatos/mastering-shiny)
- https://shiny.rstudio.com/tutorial/
- https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/
- https://shiny.rstudio.com/images/shiny-cheatsheet.pdf
- https://raw.githubusercontent.com/rstudio/cheatsheets/main/translations/spanish/shiny_es.pdf

Por que shiny?

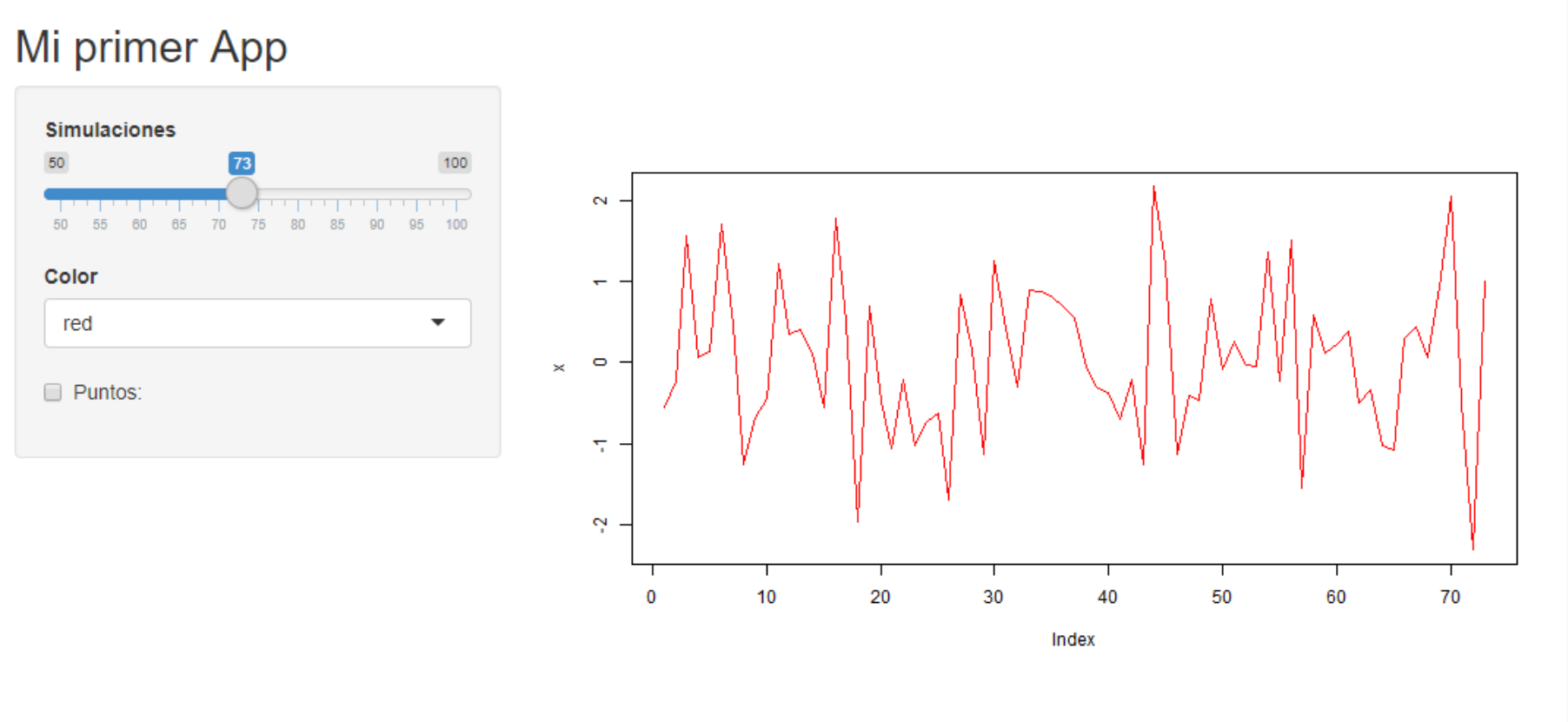


¿Qué es una **app**(licación) web?

Aplicación Web

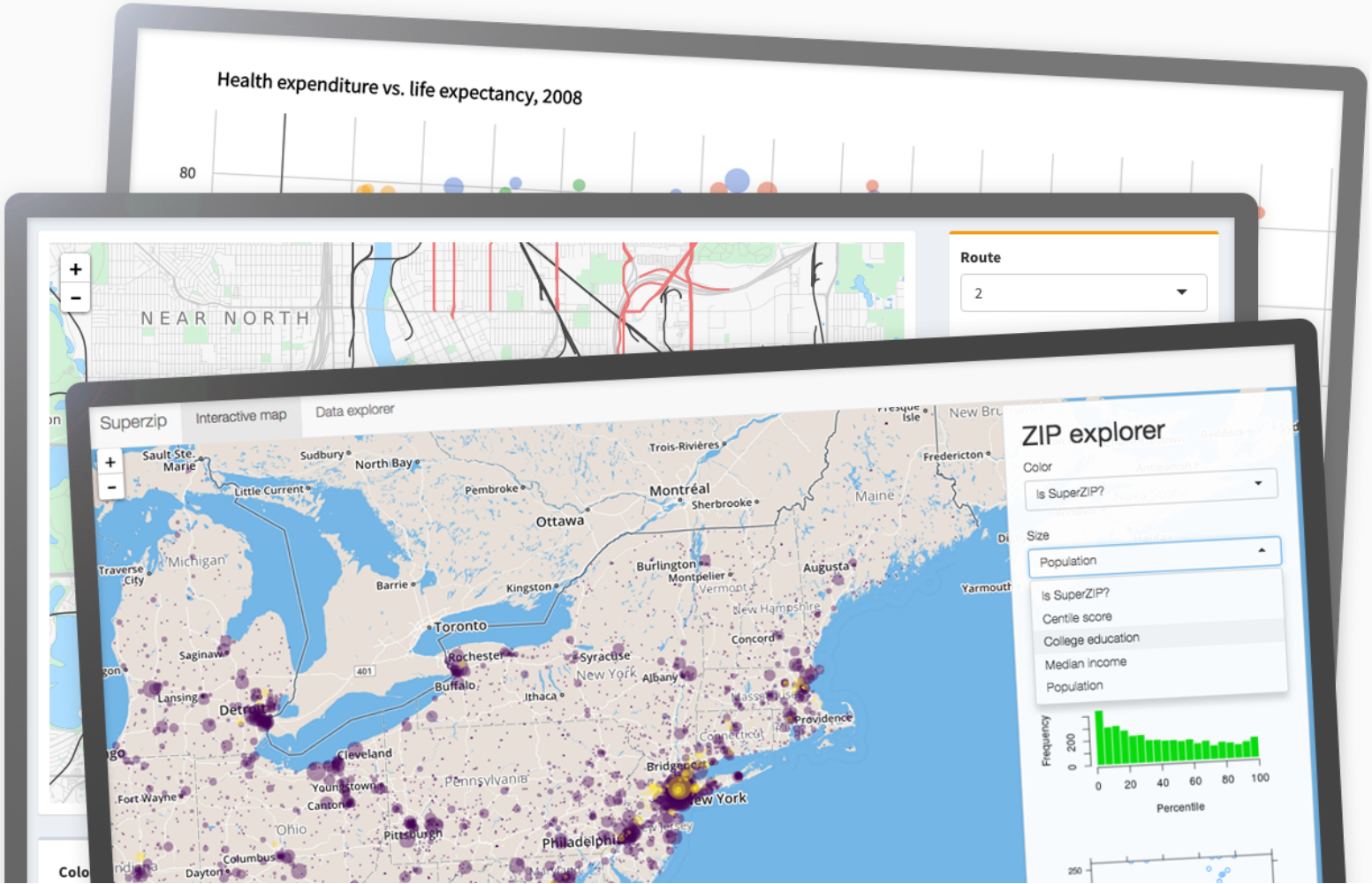
(Wikipedia:) Herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador.

Puede ser de lo más simple...



Aplicación Web

Hasta algo más complejo con más `input`s y `output`s



Ejemplos para motivarse!

Algunos simples.

- https://jjallaire.shinyapps.io/shiny-kmeans/
- https://jbkunst.shinyapps.io/05-arma/ (https://github.com/jbkunst/highcharter-shiny/tree/master/05-arma)

Otros con más detalle en la parte visual.

- https://jbkunst.shinyapps.io/trd-sttstcs/ (https://github.com/jbkunst/trd-sttstcs)
- https://jorgehcas1998.shinyapps.io/Dataton-app/ (https://github.com/socapal/dataton-tudinero)
- https://nz-stefan.shinyapps.io/commute-explorer-2

Fuente: Shiny contest https://blog.rstudio.com/2021/06/24/winners-of-the-3rd-annual-shiny-contest/ se le pide a los concursantes compartir el código, todos aprenden!

La estructura de una ShinyApp

```
library(shiny)

ui ← fluidPage()

server ← function(input, output) {}

runApp(list(ui = ui, server = 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)

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` y reflejaremos `outputs`: como tablas, gráficos.

La estructura de una ShinyApp

```
library(shiny)

ui ← fluidPage()

server ← function(input, output) {}

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

- `runApp` es la funcion 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)

ui ← fluidPage()

server ← function(input, output) {}

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

De formar general la aplicación será:

```
library(shiny)
# acá se cargarán paquetes y posiblemente también datos
# necesarios para ui (como definir opciones de inputs)

ui ← fluidPage(
  # código que da forma a nuestrá aplicación: títulos, secciones, textos, inputs
)

server ← function(input, output) {
  # toooda la lógica de como interactuan los outputs en función de los inputs
}
```

Ejercicio: Nuestra primer App andando

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

```
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)
   })
}

shinyApp(ui, server)
```

Funcionamiento de una app de Shiny

App

Contenedor

Otros contenedores

Inputs

Outputs

Interacción

Resultado

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)
   })
}
```

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: Por ejemplo uso de *tabs* de *menus*, o páginas con navegación. Más detalles http://shiny.rstudio.com/articles/layout-guide.html.

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)
   })
}
```

- `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 perminte que los valores de estos inputs sean usados como valores usuales en R en la parte del server (numericos, strings, booleanos, fechas).

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)
   })
}
```

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

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)
   })
}
```

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

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)
   })
}
```

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

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)
   })
}
```

- 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)

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)
   })
}
```

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

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)
   })
}
```

- Cada función `*Input` requiere un **id** para ser identificado en el server
- Cada `*Input` requiere argumentos especificos a cada tipo de input, valor por defecto, etiquetas, opciones, rangos, etc
- Acá, el valor númerico ingresado/modifcado por el usuario se puede acceder en el server bajo `input$nrand`

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)
   })
}
```

- `sliderInput` se usa para seleccionar un valor numérico entre un rango
- `selectInput` otorga la posibildad 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/

Ejercicio: Inputs y outputs vengan a mi!

Haga click en:

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

Hints importantes:

- No tema a escribir, ni preguntar!
- Está totalmente permitido equivocarse, de hecho se pondrán puntos extras.

Solucion

```
ui ← fluidPage(
    titlePanel("Applicación ejercicio 1"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("numerouno", "#1", min = 10, max = 500, value = 100),
            textInput("numerodos", "#2", value = 3)
        ),
        mainPanel(
           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
  })

}
```

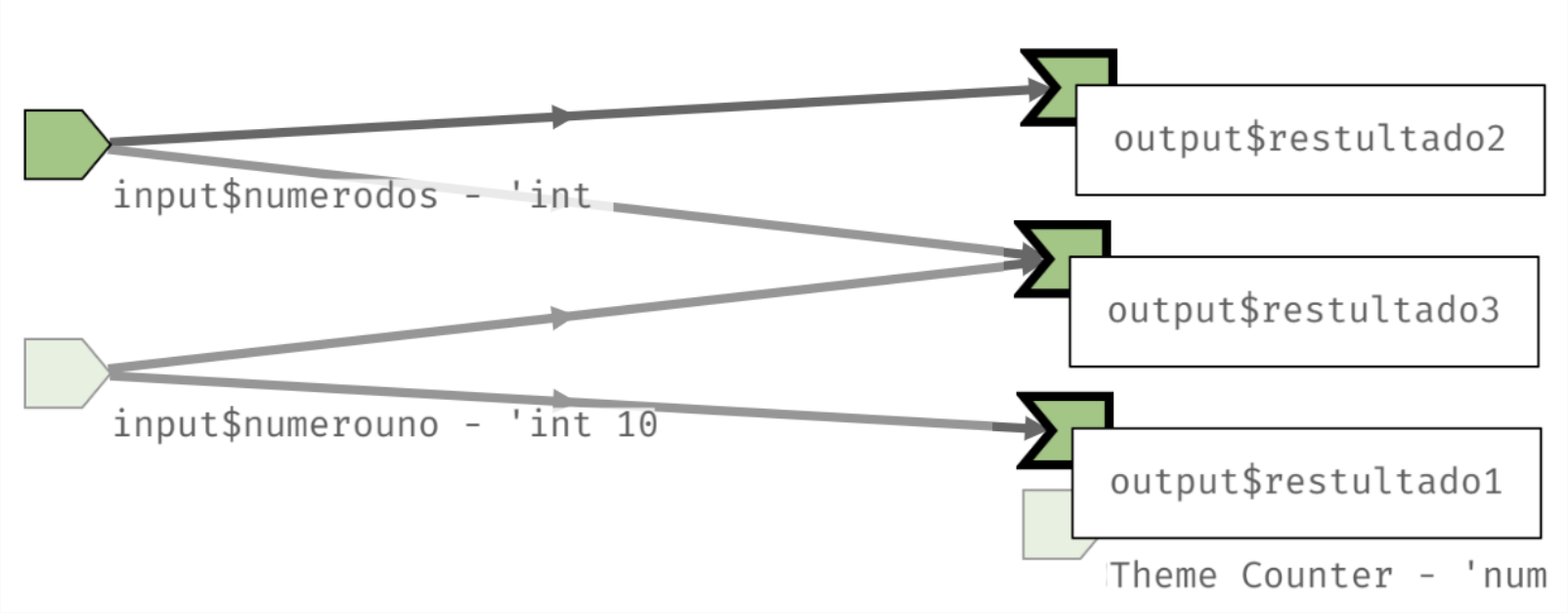
Solucion (ver. 2)

```
ui ← fluidPage(
    titlePanel("Applicación ejercicio 1"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("numerouno", "#1", min = 10, max = 500, value = 100),
            numericInput("numerodos", "#2", value = 3)
        ),
        mainPanel(
           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)
  })

}
```

Reactividad: Como funciona shiny

Consideremos la aplicación del ejemplo anterior.

Al cambiar un input -como lo es `input$numerouno` o `input$numerodos`- 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.