

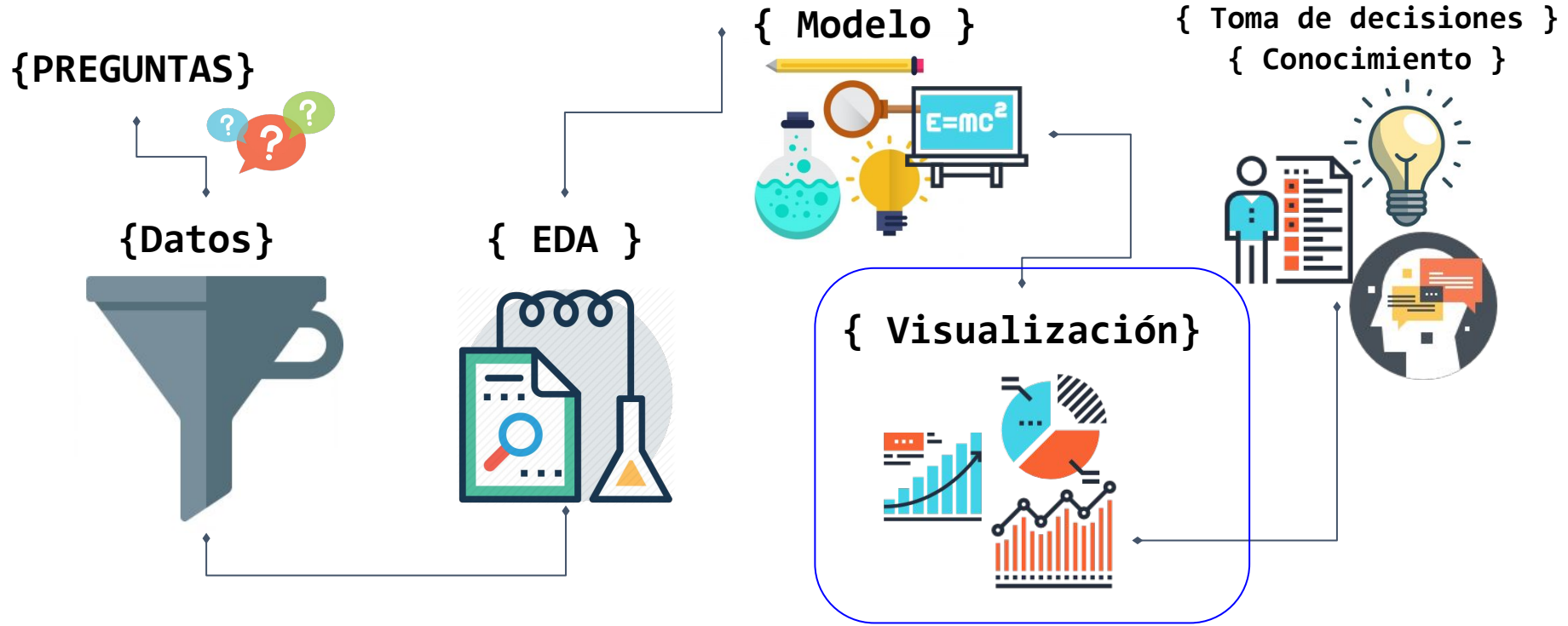
{ SHINY: Parte I }

Que vamos a ver hoy?

- ShinyApp library
- ShinyApp arquitectura
- User interface
- Server
- Inputs
- Outputs



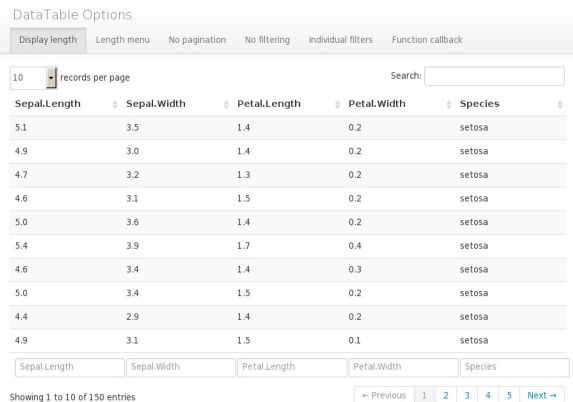
PROCESO DE ANÁLISIS



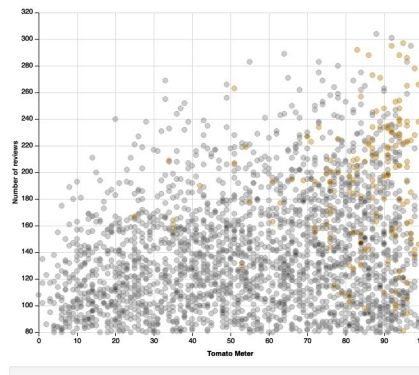
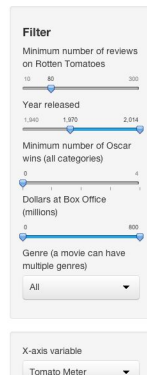
{ Shiny App }

Shiny es un paquete R que facilita la creación de aplicaciones web interactivas directamente desde R.

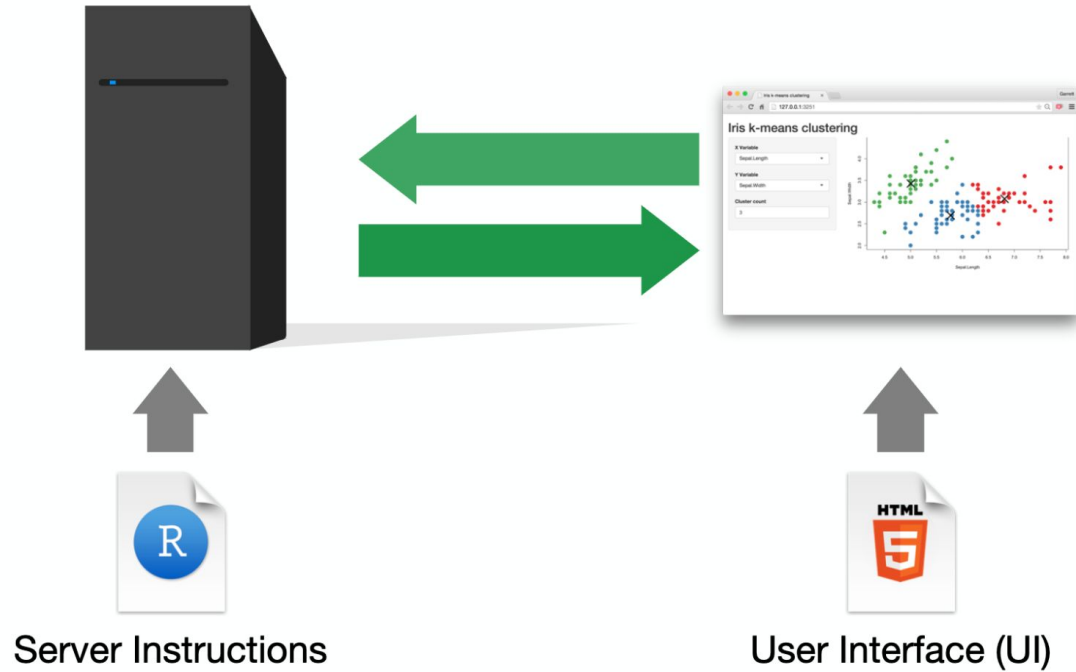
También puede ampliar sus aplicaciones Shiny con temas CSS , widgets html y acciones de JavaScript .



Movie explorer



{ Arquitectura }



{ Mi primer Shiny app }

```
library(shiny)
```

1

Importar shiny

```
ui<- fluidPage()
```

2

Generar user interface

```
server<- function(input, output)
```

3

Generar el Server

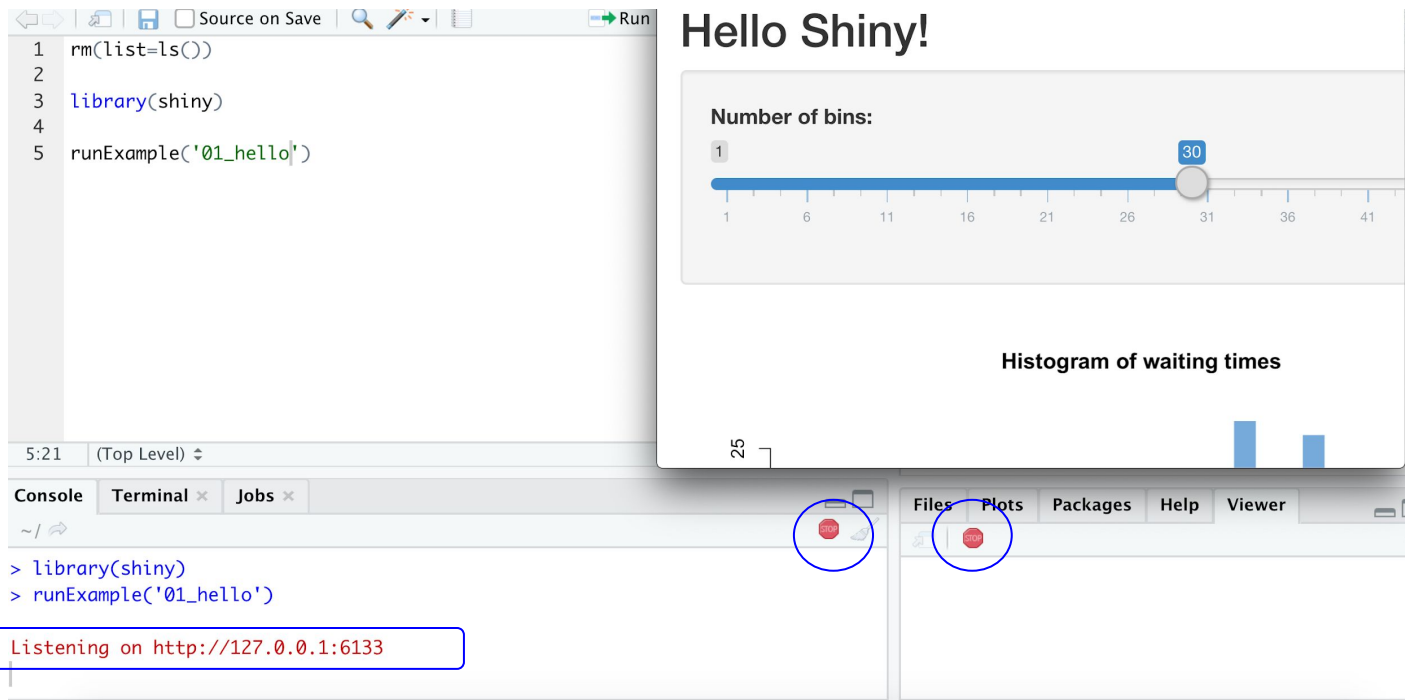
```
shinyApp(ui=ui, server=server)
```

4

Generar la Shiny App



{ Cerrar una aplicación }



The screenshot displays the RStudio interface with a Shiny application running. The source editor on the left contains the following R code:

```
1 rm(list=ls())  
2  
3 library(shiny)  
4  
5 runExample('01_hello')
```

The console at the bottom shows the execution of the code:

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

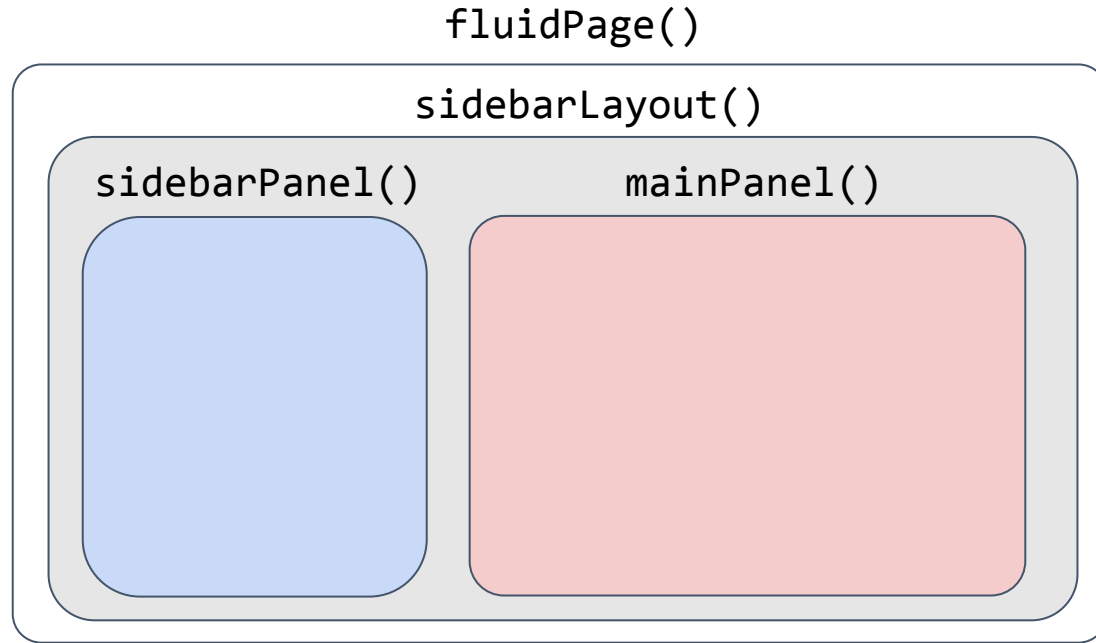
A message box at the bottom of the console indicates: "Listening on http://127.0.0.1:6133".

The Shiny application window, titled "Hello Shiny!", features a slider control for "Number of bins" with a range from 1 to 41 and a current value of 30. Below the slider is a histogram titled "Histogram of waiting times".

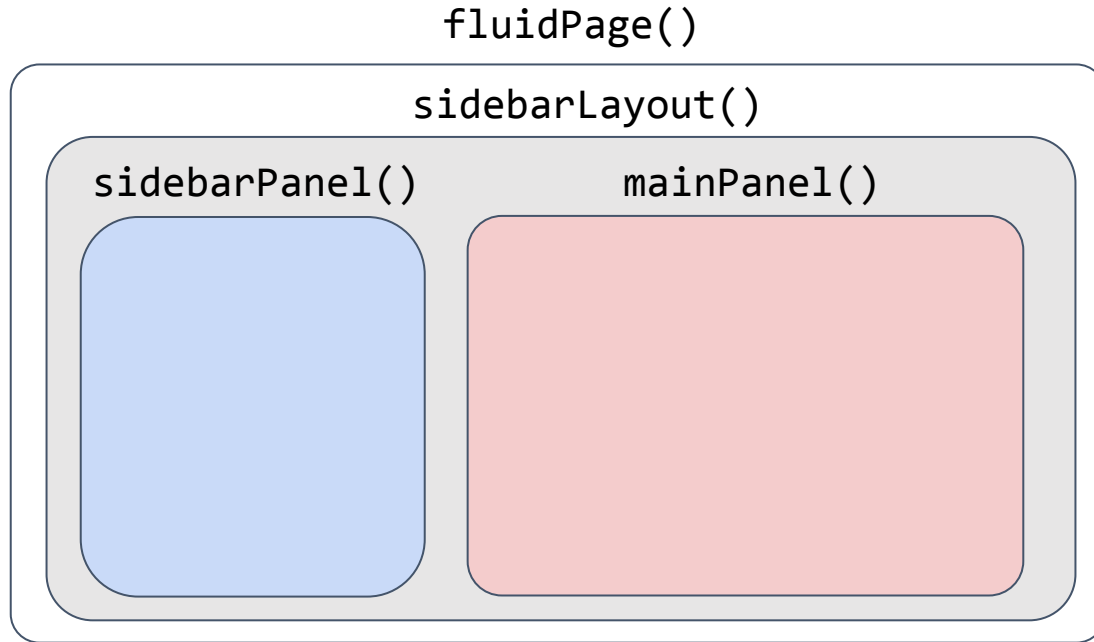
Two red "stop" buttons are circled in blue. One is located in the bottom-left corner of the RStudio window, and the other is located in the bottom-right corner of the Shiny application window. These buttons are used to stop the application.



{ UI - Layout }



{ UI - Layout }



```
ui=fluidPage(  
  sidebarLayout(  
    sidebarPanel(...),  
    mainPanel(...)  
  )  
)
```



{ Inputs }

Buttons

Action

Submit

`actionButton()`
`submitButton()`

Single checkbox

☒ Choice A

`checkboxInput()`

Checkbox group

☒ Choice 1
☐ Choice 2
☐ Choice 3

`checkboxGroupInput()` `dateInput()`

Date input

2014-01-01

`dateInput()`

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

Choose File No file chosen

`fileInput()`

Numeric input

1

`numericInput()`

Password Input

.....

`passwordInput()`

Radio buttons

☒ Choice 1
☐ Choice 2
☐ Choice 3

`radioButtons()`

Select box

Choice 1

`selectInput()`

Sliders

0 50 100
0 25 75 100

`sliderInput()`

Text input

Enter text...

`textInput()`



{ Inputs }

function	widget
actionButton	Botón para acciones
checkboxGroupInput	Un grupo de checkboxes
checkboxInput	Un simple checkbox
dateInput	Selección de una fecha
dateRangeInput	Selección de un par de fechas
fileInput	Subir un archivo
helpText	Texto de ayuda para otro control
numericInput	Campo para insertar un dato numérico
radioButtons	Un grupo de radio buttons
selectInput	Clásica caja con opciones para seleccionar
sliderInput	Control para seleccionar un dato numérico
submitButton	Un botón de submit
textInput	Campo para ingresar un dato de texto



{ Output }

Outputs - render*() and *Output() functions work together to add R output to the UI



DT::renderDataTable(expr,
options, callback, escape,
env, quoted)

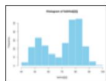


dataTableOutput(outputId, icon, ...)



renderImage(expr, env, quoted, deleteFile)

imageOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)



renderPlot(expr, width, height, res, ..., env,
quoted, func)

plotOutput(outputId, width, height, click,
dblclick, hover, hoverDelay, hoverDelayType,
brush, clickId, hoverId, inline)

data.frame(" ", 3 obs, of 2 variables:
\$ Input.Length: num 3.2 3.2 3.2
\$ Input.Length: num 3.2 3.2 3.2

renderPrint(expr, env, quoted, func,
width)

verbatimTextOutput(outputId)

	Input.Length	Input.Length	Input.Length	Input.Length
1	3.2	3.2	3.2	3.2
2	3.2	3.2	3.2	3.2
3	3.2	3.2	3.2	3.2

renderTable(expr, ..., env, quoted, func)

tableOutput(outputId)

foo

renderText(expr, env, quoted, func)

textOutput(outputId, container, inline)

renderUI(expr, env, quoted, func)

uiOutput(outputId, inline, container, ...)
& **htmlOutput**(outputId, inline, container, ...)

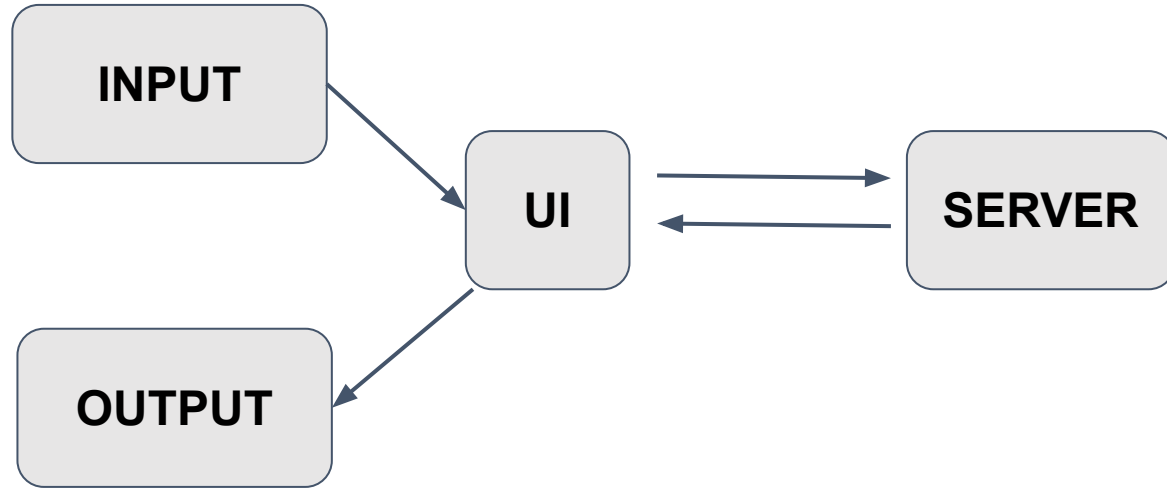


{ Output }

Output	function render	function creates
htmlOutput/uiOutput	renderUI	Salida para HTML
imageOutput	renderImage	Imágenes
plotOutput	renderPlot	gráficos
tableOutput	renderTable	Para imprimir una tabla (data frame, matrix, etc)
textOutput	renderText	salidas de texto
verbatimTextOutput	renderPrint	imprimir texto (ej: summary(cars))



{ Input - Output }



{ Cheat Sheet }

<https://rstudio.com/wp-content/uploads/2015/03/shiny-spanish.pdf>

Shiny Hoja de referencia
lee más en shiny.rstudio.com
Shiny 0.10.0 Actualizado: 6/14

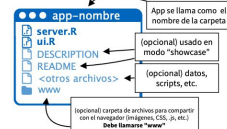


2. server.R Instrucciones que constituyen los componentes R de tu app. Para escribir server.R:

- Provee server.R con el mínimo de código necesario, `shinyServer(function(input, output){})`
- Define los componentes en R para tu app entre las llaves `{}` después de `function(input, output)`.
- Guarda cada componente R destinados para tu interfaz (UI) como `output$nombre componente`.
- Crea cada componente de salida con una función `render*`.
- Dale a cada función `render*` el código R que el servidor necesita para construir el componente. El servidor notará valores reactivos que aparecen en el código y reconstruirá el componente cada vez que estos valores cambian.
- Has referencia a valores en "widgets" con `input$nombre del widget`.

RStudio® and Shiny® son marcas registradas de RStudio, Inc.
© 2014 RStudio. <https://shiny.rstudio.com>
800-448-1212 <https://shiny.rstudio.com>
Traducido por Francisco Javier Rodríguez

1. Estructura Cada app es una carpeta que contiene un archivo `server.R` y comúnmente un archivo `ui.R` (opcionalmente contiene archivos extra)

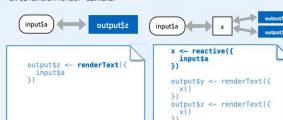


```
# carga paquetes, scripts, datos
shinyServer(function(input, output) {
  # crea variables específicas para usuario
  input$title <- renderText({
    "Hola mundo"
  })
  output$grafica <- renderPlot({
    y <- mtcars[, "mpg"]
    plot(x, y, pch = 16)
  })
})
```

4. Reactividad Cuando una entrada (input) cambia, el servidor reconstruye cada salida (output) que depende de ella (también si la dependencia es indirecta). Puedes controlar este comportamiento a través de la cadena de dependencias.

render* - Una salida se actualiza automáticamente cuando una entrada en su función `render*` cambia.

reactive - Usa `reactive` para crear objetos que se usan en múltiples salidas.



funciones render*

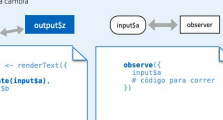
función	espera	crea
<code>renderDataTable</code>	objetos como tablas	tabla DataFrames
<code>renderImage</code>	links a imágenes	imagen HTML
<code>renderPlot</code>	gráficos	gráfica
<code>renderPrint</code>	salida impresa	texto
<code>renderTable</code>	objetos como tablas	tabla simple
<code>renderText</code>	cadena de caracteres	texto
<code>renderUI</code>	objetos "tag" o HTML	elemento UI (HTML)

valores de entrada (`input`) son reactivos.
Deben estar rodeados por una de:
render* - crea un componente shiny UI (interfaz)
reactive - crea una expresión reactiva
observe - crea un observador reactivo
isolate - crea una copia no reactiva de un objeto reactivo

3. Ejecución Coloca código en el lugar donde correrá la menor cantidad de veces

- Corre una vez - código puesto fuera de `shinyServer` solo corre una vez cuando inicias tu app. Úsalo para instrucciones generales. Crea una sola copia en memoria.
- Corre una vez por usuario - código puesto dentro de `shinyServer` corre una vez por cada usuario que visita tu app (lo refresca su navegador). Úsalo para instrucciones que necesites dar por cada usuario de tu app. Crea una copia por cada usuario.
- Corre a menudo - código puesto dentro de una función `render*`, `reactive` o `observe` correrá muchas veces. Úsalo solo para código que el servidor necesita para reconstruir un componente UI después de que un widget cambia.

observe - Usa `observe` para crear código que corre cuando una entrada cambia, pero que no crea un objeto de salida.



ui.R

```
shinyUI(fluidPage(
  titlePanel("datos mtcars"),
  sidebarLayout(
    sidebarPanel(
      selectInput("titulo", "título gráfica",
        value = "x y"),
      selectInput("m", "Escala una var x:",
        choices = names(mtcars),
        selected = "disp"),
      selectInput("y", "Escala una var y:",
        choices = names(mtcars),
        selected = "mpg")
    ),
    mainPanel(
      h3(textOutput("texto")),
      plotOutput("plot")
    )
  )
)
```

Componentes R - Los objetos de salida que has definido en `server.R`. Para colocar un componente:

- Selecciona la función `Output` que construye el tipo de objeto que quieres colocar en la UI.
- Passa la función `Output` a una cadena de caracteres correspondiente al nombre del objeto en `server.R`.

`output$grafica <- renderPlot(.)` → `plotOutput("gráfica")`

funciones "Output"

<code>dataTableOutput</code>	<code>tableOutput</code>
<code>htmlOutput</code>	<code>textOutput</code>
<code>imageOutput</code>	<code>uiOutput</code>
<code>plotOutput</code>	<code>verbatimTextOutput</code>

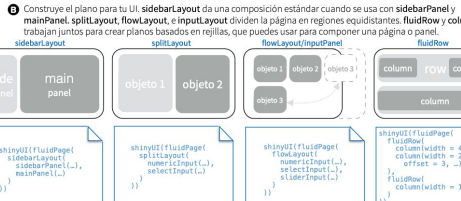
6. Corre tu app

runApp - corre archivos locales
runWeb - corre archivos alojados en www.github.com
runGit - corre archivos guardados como git (git.github.com)
runURL - corre archivos guardados en algún URL

RStudio® and Shiny® son marcas registradas de RStudio, Inc.
© 2014 RStudio. <https://shiny.rstudio.com>
800-448-1212 <https://shiny.rstudio.com>
Traducido por Francisco Javier Rodríguez

5. ui.R Una descripción de la interfaz (UI) de tu app, la página web que muestra tu app. Para escribir ui.R:

- Incluye el mínimo de código necesario para ui.R, `shinyUI(fluidPage())`
Nota: Usa `navbarPage` en vez de `fluidPage` si quieres que tu app tenga múltiples páginas conectadas con un navbar
- Construye el plano para tu UI. `sidebarLayout` da una composición estándar cuando se usa con `sidebarPanel` y `mainPanel`. `splitLayout`, `flowLayout`, `columnLayout` dividen la página en regiones equidistantes. `fluidRow` y `column` trabajan juntos para crear planos basados en rejillas, que puedes usar para componer una página o panel.



Widgets - El primer argumento de cada función de widget es el nombre del widget. Puedes acceder a su valor actual en `server.R` con `input$nombre`.

widget	función	argumentos comunes
botón de acción	<code>actionButton</code>	<code>input, label</code>
caja	<code>checkboxGroup</code>	<code>input, label, value</code>
grupo de casillas	<code>checkboxGroup</code>	<code>input, label, choices, selected</code>
selector de fecha	<code>dateRangeInput</code>	<code>input, label, value, min, max, format</code>
selección rango fecha	<code>dateRangeInput</code>	<code>input, label, value, min, max, format</code>
lista de archivos	<code>fileInput</code>	<code>input, label, multiple</code>
campo numérico	<code>numericInput</code>	<code>input, label, value, min, max, step</code>
botón de selección	<code>radioButtons</code>	<code>input, label, choices, selected</code>
cadena de selección	<code>selectInput</code>	<code>input, label, choices, selected, multiple</code>
botón de error	<code>sliderInput</code>	<code>input, label, min, max, value, step</code>
campo de texto	<code>textInput</code>	<code>input, label, value</code>

7. Comparte tu app

Presenta tu app como una página web accesible en línea

ShinyApps.io
Aloja tus apps en el servidor de RStudio. Opciones gratis y pagas.
www.shinyapps.io

Shiny Server
Construye tu propio servidor linux para alojar apps. Gratis y de código abierto.
shiny.rstudio.com/deploy

Shiny Server Pro
Construye un servidor comercial con autenticación, gestión de recursos y más.
shiny.rstudio.com/deploy