

Master en Ciencia de datos **Visualización Avanzada de Datos**

Visualización Interactiva con R, Shiny.

Miguel Lozano, Juan Gómez, Marcelino Martínez

Departamento de Ingeniería Electrónica, Escuela Técnica Superior de Ingeniería
Universidad de Valencia, Avda Universidad S/N
46100, Burjassot (Valencia)

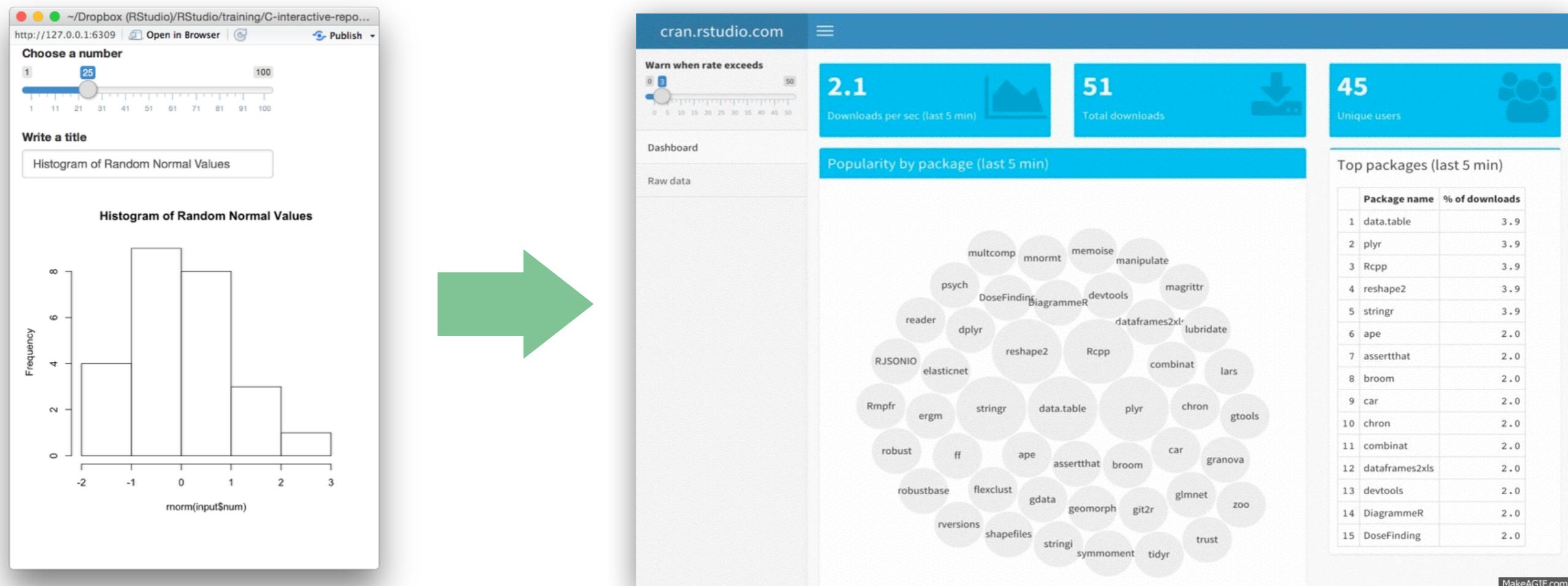
miguel.lozano@uv.es; juan.gomez-sanchis@uv.es; marcelno.martinez@uv.es

Creación de aplicaciones con Shiny

[imágenes obtenidas de bit.ly/shiny-quickstart-1]

Qué es Shiny?

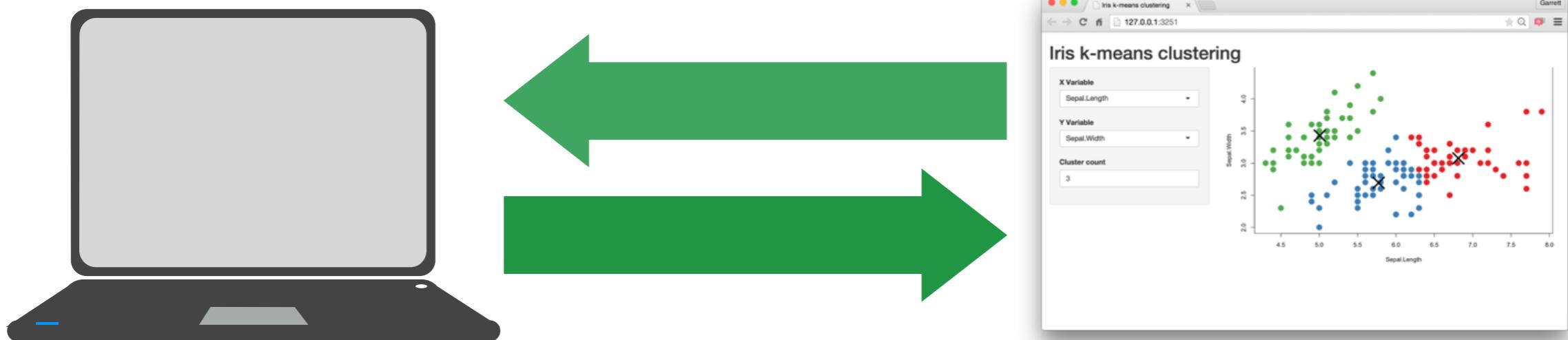
Shiny es una librería de R que permite construir aplicaciones web para visualizar gráficos interactivos en lenguaje R.



Arquitectura de una Aplicación Shiny

Arquitectura Shiny

Cada aplicación Shiny se mantiene en un ordenador ejecutando R



Arquitectura Shiny



Instrucciones del
servidor

Interfaz de
usuario (UI)

Plantilla de una aplicación

Código mínimo de una aplicación Shiny

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

Plantilla de una aplicación

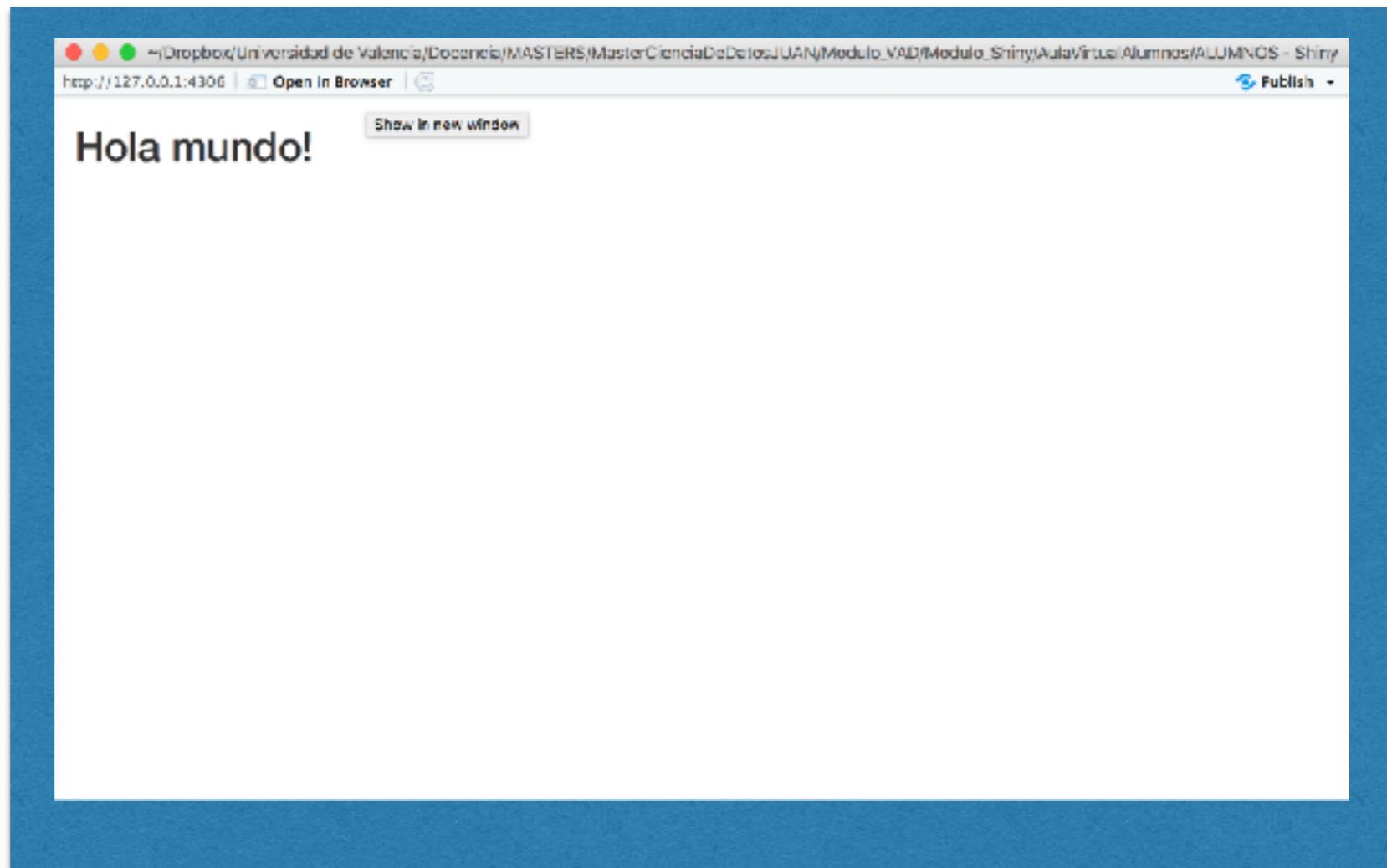
Añade elementos a la aplicación como argumentos
de fluidPage()

```
library(shiny)  
ui <- fluidPage("Hello World")  
  
server <- function(input, output) {}  
  
shinyApp(ui = ui, server = server)
```

© CC 2015 RStudio, Inc.

Plantilla de una aplicación

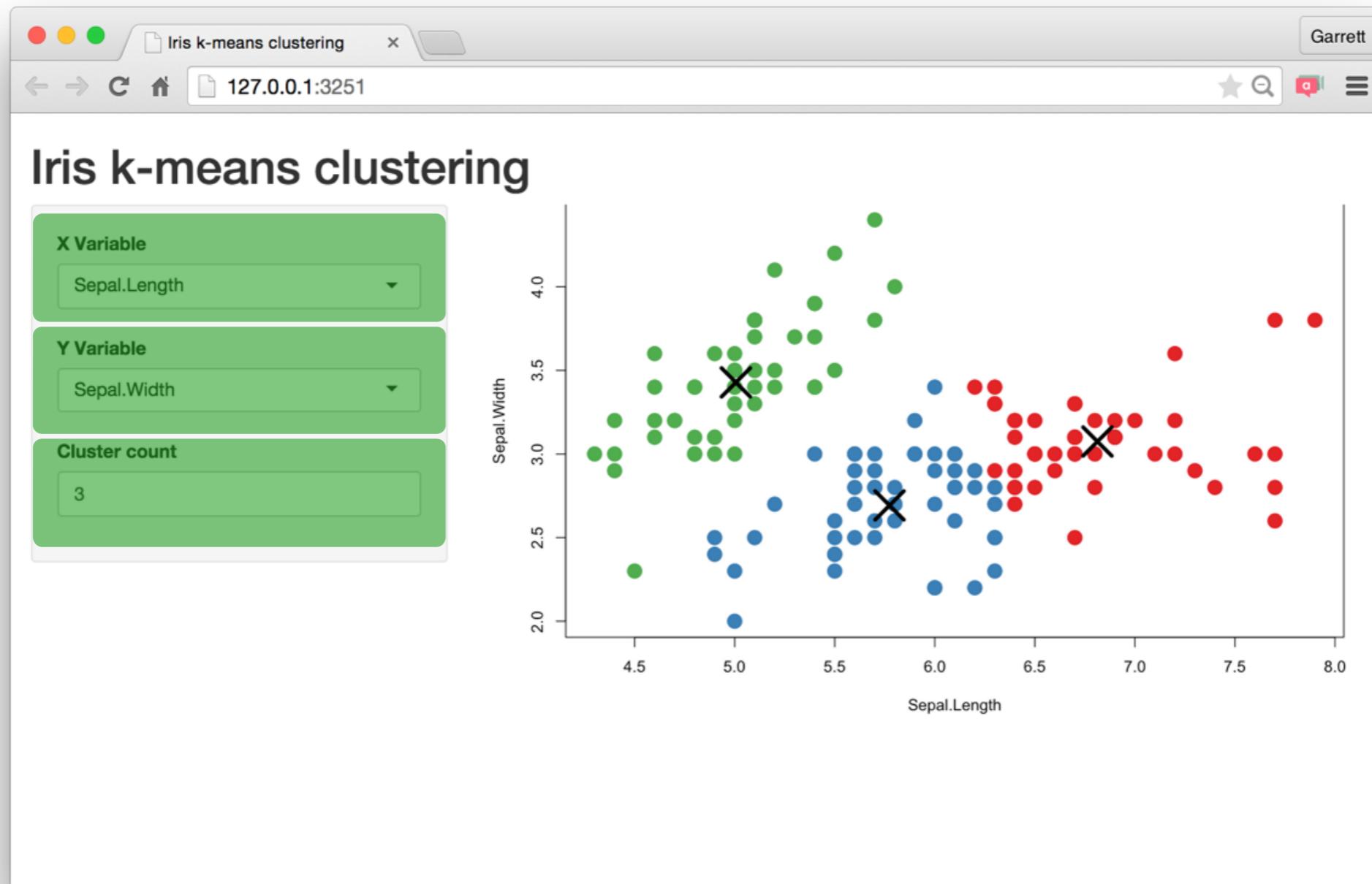
Añade elementos a la aplicación como argumentos de fluidPage()



Desarrollo de una aplicación Interfaz de usuario

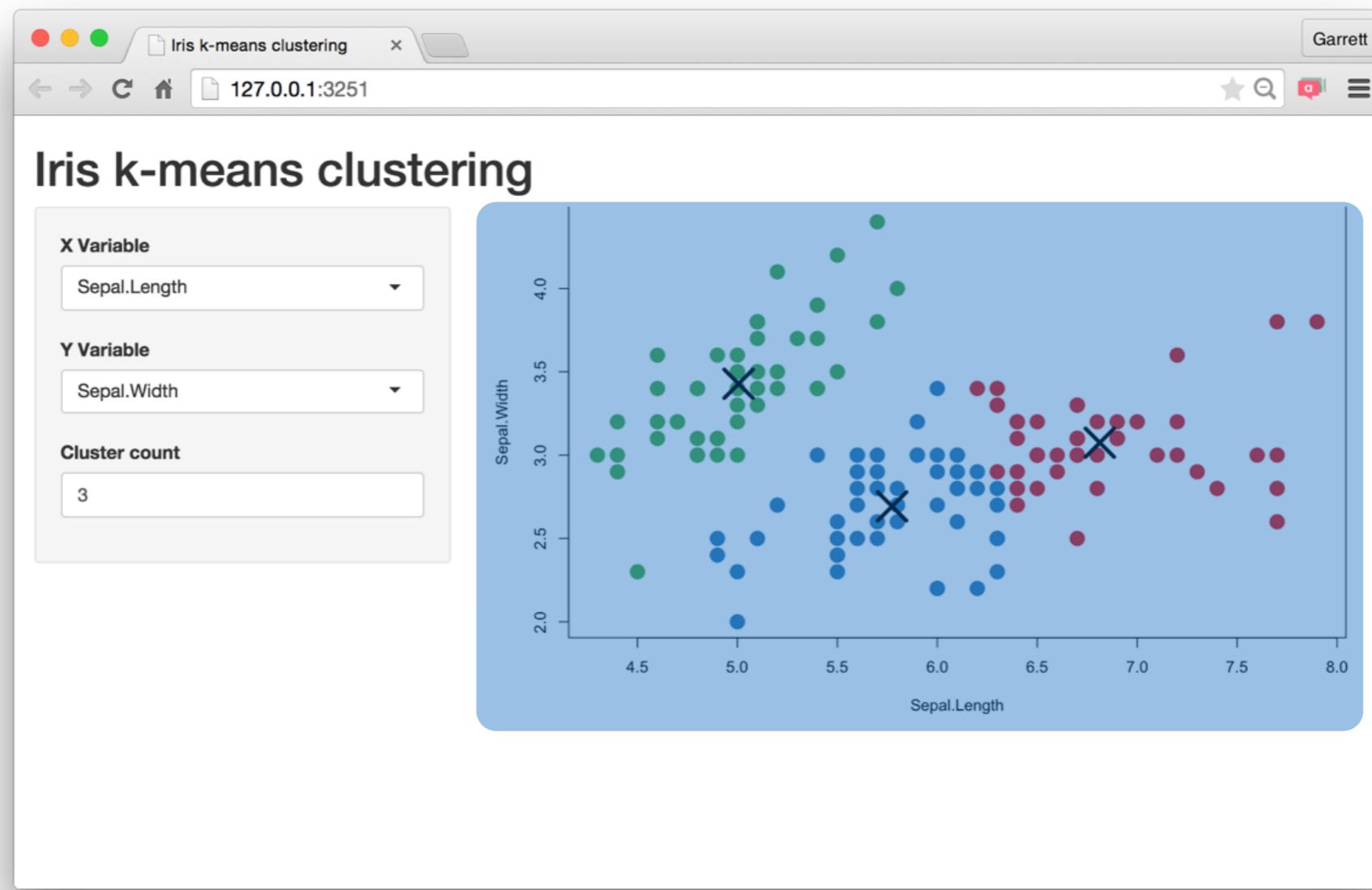
Interfaz de usuario

Toda aplicación consiste en **entradas** y **salidas**



Interfaz de usuario

Toda aplicación consiste en **entradas** y **salidas**



Interfaz de usuario

Añade elementos a la aplicación como argumentos de `fluidPage()`

```
ui <- fluidPage(  
  # *Input() functions,  
  # *Output() functions  
)
```

© CC 2015 RStudio, Inc.

Entradas

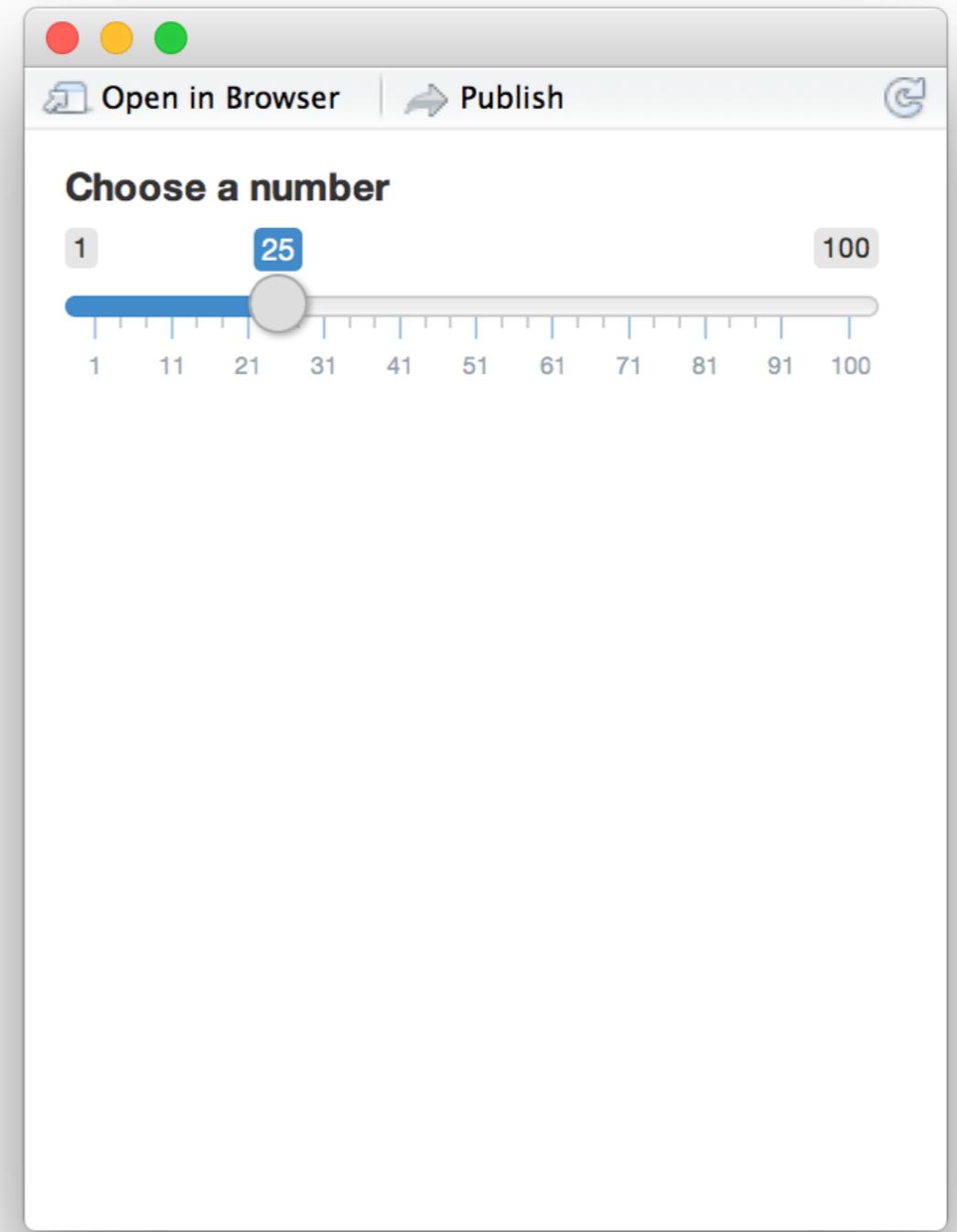
Entradas

Crea una entrada con una función *Input()

```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num",
  label = "Choose a number",
  value = 25, min = 1, max = 100)
)
```

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

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



Entradas

Buttons

Action

Submit

`actionButton()`
`submitButton()`

Single checkbox

Choice A

`checkboxInput()`

Checkbox group

- Choice 1
- Choice 2
- Choice 3

Date input

2014-01-01

`checkboxGroupInput()` `dateInput()`

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

No file chosen

`fileInput()`

Numeric input

1

`numericInput()`

Password Input

.....

`passwordInput()`

Radio buttons

- Choice 1
- Choice 2
- Choice 3

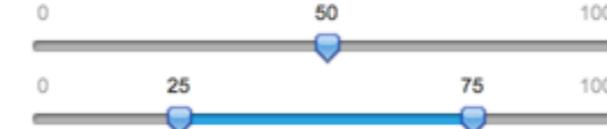
`radioButtons()`

Select box

Choice 1

`selectInput()`

Sliders



`sliderInput()`

Text input

Enter text...

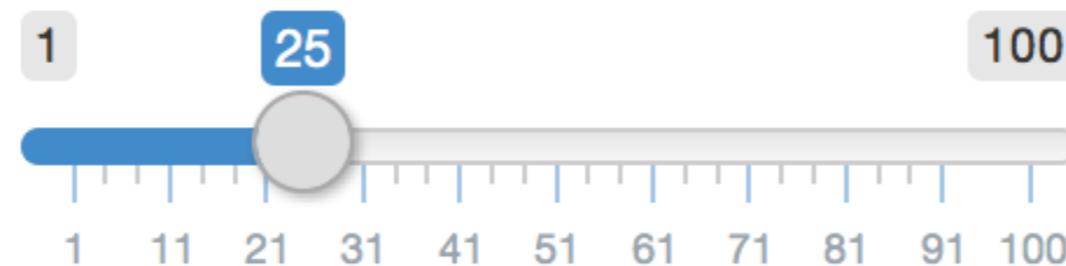
`textInput()`

© CC 2015 RStudio, Inc.



Sintaxis

Choose a number



```
sliderInput(inputId = "num", label = "Choose a number", ...)
```

input name
(for internal use)

Notice:
Id not ID

label to
display

input specific
arguments

?sliderInput

Salidas

Salidas

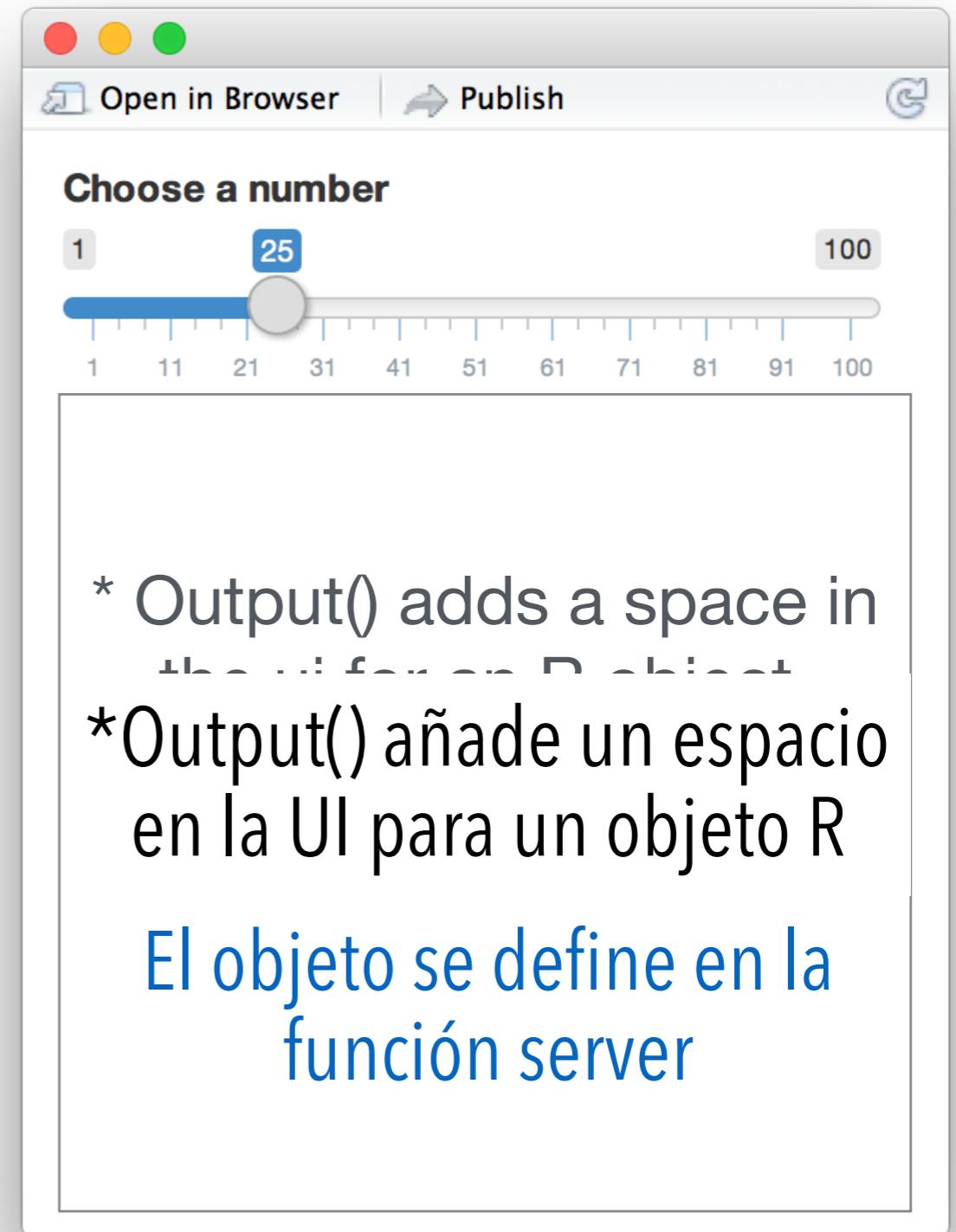
Crea una salida con una función *Output()

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)

server <- function(input, output) {}

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



Salidas

Función	Inserta
dataTableOutput()	tabla interactiva
htmlOutput()	HTML puro
imageOutput()	imagen
plotOutput()	gráfica
tableOutput()	tabla
textOutput()	texto
uiOutput()	elemento Shiny UI
verbatimTextOutput()	texto

Sintaxis

plotOutput("hist")

the type of output
to display

name to give to the
output object

© CC 2015 RStudio, Inc.

**Conecta en el
servidor
las entradas con
las salidas**

Función server

Se definen **3 reglas** para escribir la función server

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

© CC 2015 RStudio, Inc.

Función server

1

Guarda los objetos a mostrar en output\$

```
server <- function(input, output) {  
  output$hist <- # code  
}
```

Función server

1

Guarda los objetos a mostrar en output\$

output\$hist



plotOutput("hist")

Función server

2

Construye los objetos a mostrar con `render*()`

```
server <- function(input, output) {  
  output$hist <- renderPlot({  
    })  
}
```

Función server

Utiliza la función `render*()` para crear el tipo de salida adecuado

Función	Crea
<code>renderDataTable()</code>	tabla interactiva
<code>renderImage()</code>	imagen
<code>renderPlot()</code>	gráfica
<code>renderPrint()</code>	bloque de código de una salida print
<code>renderTable()</code>	tabla
<code>renderUI()</code>	elemento Shiny UI
<code>renderText()</code>	cadena de texto

render*()

Crea una salida reactiva para mostrar en la UI

```
renderPlot({ hist(rnorm(100)) })
```

type of object to build

code block that builds the object

Función server

2

Construye los objetos a mostrar con `render*()`

```
server <- function(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(100))  
  })  
}
```

Función server

3

Accede a los valores de **entrada** con `input$`

```
server <- function(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(input$num))  
  })  
}
```

Función server

3

Accede a los valores de **entrada** con `input$`

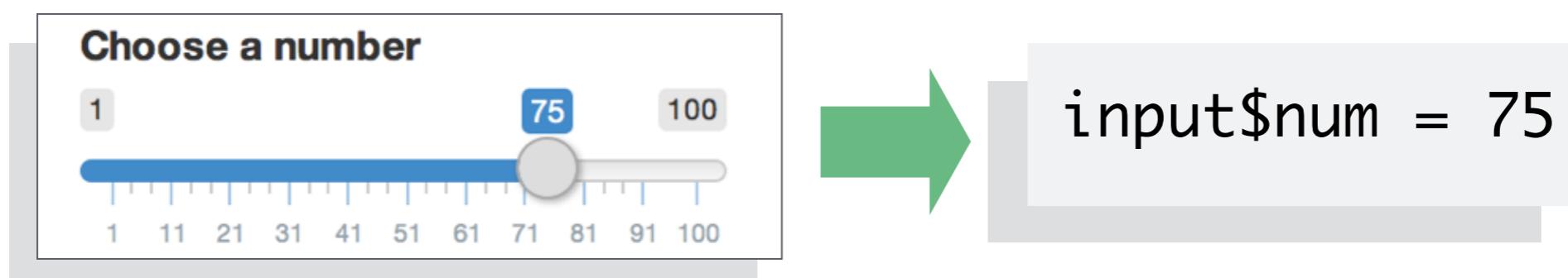
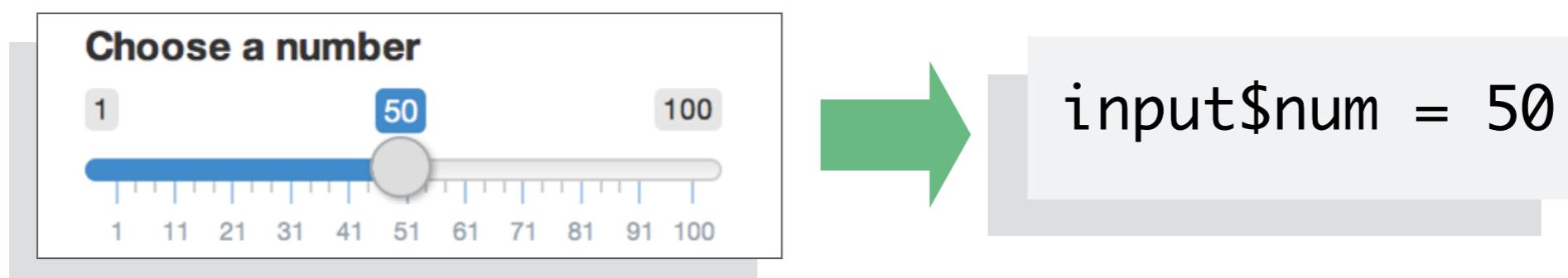
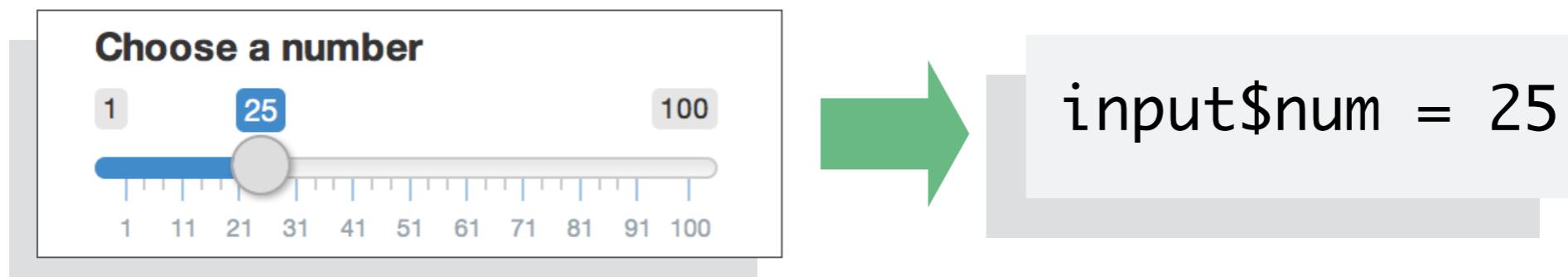
```
sliderInput(inputId = "num",...)
```



```
input$num
```

Valores de entrada

El valor de entrada cambia cada vez que un usuario cambia la entrada



Reactividad

Cada vez que cambia un valor de entrada se actualiza automáticamente (se vuelve a generar) el objeto de salida que lo contiene

```
function(input, output) {  
  output$hist <- renderPlot({  
    hist(rnorm(input$num))  
  })  
}  
})
```

Reactividad

Las funciones render*() construyen una salida reactiva en el UI de la aplicación

```
renderPlot( { hist(rnorm(input$num)) } )
```

cuando se notifica la modificación de una entrada, el objeto creado por la función render*() vuelve a ejecutar el bloque de código asociado

Reactividad

La función `reactive()` crea un objeto reactivo
(expresión reactiva)

```
data <- reactive( { rnorm(input$num) })
```

object will respond to *every reactive value in the code*

code used to build (and rebuild) object

cuando se notifica la modificación de una entrada, la expresión asociada a la función `reactive*()` se vuelve a ejecutar

Reactividad

La función `reactive()` crea un objeto reactivo
(expresión reactiva)

las expresiones reactivas se utilizan como una función

las expresiones reactivas mantienen su valor en caché

Reactividad

La función `reactive()` crea un objeto reactivo
(expresión reactiva)

```
# 02-two-outputs

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  verbatimTextOutput("stats")
)

server <- function(input, output) {

  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
  output$stats <- renderPrint({
    summary(rnorm(input$num))
  })
}

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

```
# 03-reactive

library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist"),
  verbatimTextOutput("stats")
)

server <- function(input, output) {
  data <- reactive({
    rnorm(input$num)
  })
  output$hist <- renderPlot({
    hist(data())
  })
  output$stats <- renderPrint({
    summary(data())
  })
}

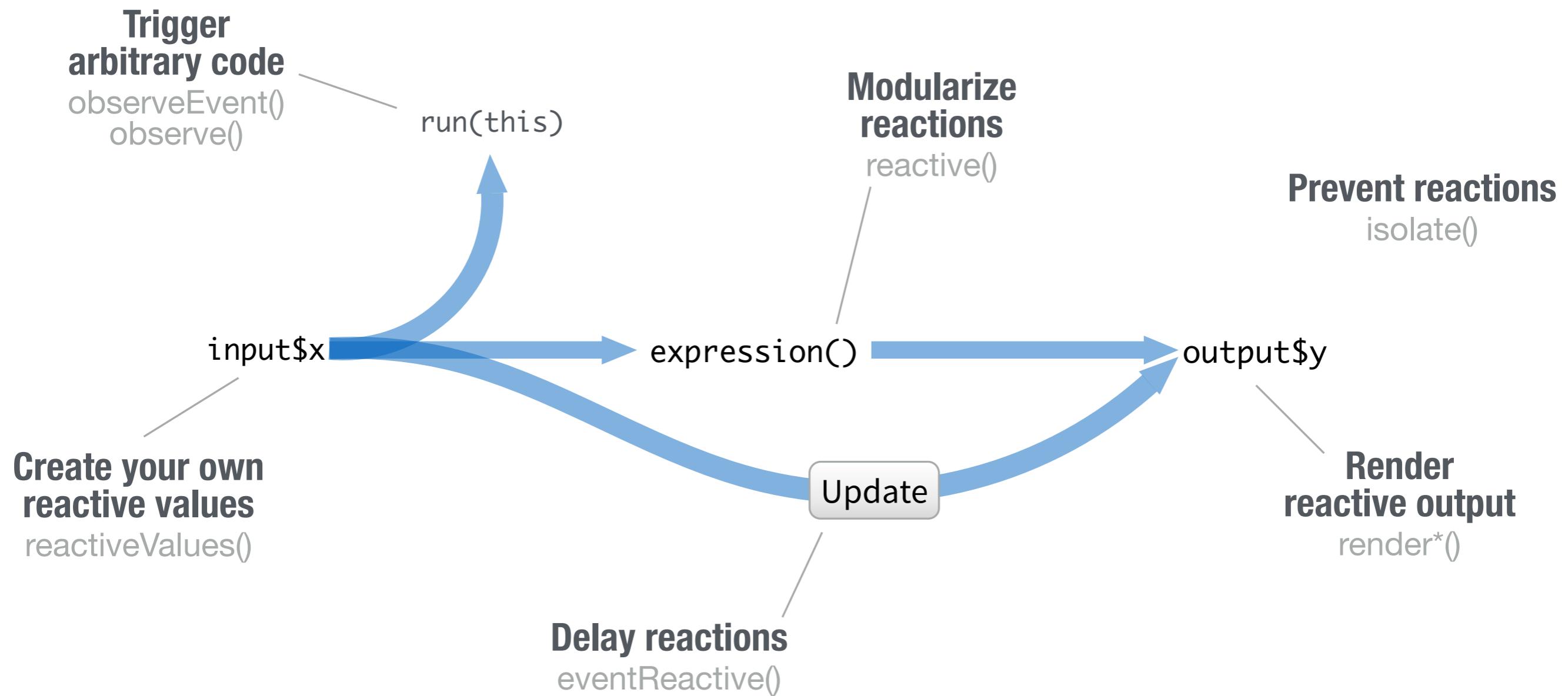
shinyApp(ui = ui, server = server)
```

Reactividad

Hay distintas funciones para el control de la reactividad:

Función	Uso
<code>isolate()</code>	evita una ejecución reactiva
<code>observeEvent()</code>	observa una entrada reactiva
<code>observe()</code>	observa todas las entradas reactivas
<code>eventReactive()</code>	retrasa la ejecución del código
<code>reactiveValues()</code>	controla programáticamente un valor

Reactividad



Actualizar la UI desde el server

Actualizar la UI

Utiliza las funciones `update*()` para cambiar el valor de una entrada de la UI desde la función `server()`

<code>updateActionButton</code>	Change the label or icon of an action button on the client
<code>updateCheckboxGroupInput</code>	Change the value of a checkbox group input on the client
<code>updateCheckboxInput</code>	Change the value of a checkbox input on the client
<code>updateDateInput</code>	Change the value of a date input on the client
<code>updateDateRangeInput</code>	Change the start and end values of a date range input on the client
<code>updateNumericInput</code>	Change the value of a number input on the client
<code>updateRadioButtons</code>	Change the value of a radio input on the client
<code>updateSelectInput</code> (<code>updateSelectizeInput</code>)	Change the value of a select input on the client
<code>updateSliderInput</code>	Change the value of a slider input on the client
<code>updateTabsetPanel</code> (<code>updateNavbarPage</code> , <code>updateNavlistPanel</code>)	Change the selected tab on the client
<code>updateTextInput</code>	Change the value of a text input on the client
<code>updateTextAreaInput</code>	Change the value of a textarea input on the client
<code>updateQueryString</code>	Update URL in browser's location bar

Actualizar la UI

```
shinyApp(  
  ui = fluidPage(  
    sidebarLayout(  
      sidebarPanel(  
        p("The first slider controls the second"),  
        sliderInput("control", "Controller:", min=0, max=20, value=10,  
                  step=1),  
        sliderInput("receive", "Receiver:", min=0, max=20, value=10,  
                  step=1)  
      ),  
      mainPanel()  
    ),  
  ),  
  server = function(input, output, session) {  
    observe({  
      val <- input$control  
      # Control the value, min, max, and step.  
      # Step size is 2 when input value is even; 1 when value is odd.  
      updateSliderInput(session, "receive", value = val,  
                        min = floor(val/2), max = val+4, step = (val+1)%%2 + 1)  
    })  
  }  
)
```

Añadir contenido estático

Añadir contenido estático

En HTML se añade contenido con **tags**:

<h1></h1>

<a>

Añadir contenido estático

En shiny hay funciones para recrear los tags HTML:

`tags$h1()`  `<h1></h1>`

`tags$a()`  `<a>`

tags

Cada elemento de la lista de tags es una función que crea un tag html

names(tags)

```
## [1] "a"          "abbr"       "address"    "area"  
## [5] "article"    "aside"      "audio"      "b"  
## [9] "base"       "bdi"        "bdo"        "blockquote"  
## [13] "body"       "br"         "button"     "canvas"  
## [17] "caption"    "cite"       "code"       "col"  
## [21] "colgroup"   "command"    "data"       "datalist"  
## [25] "dd"         "del"        "details"    "dfn"  
## [29] "div"        "dl"         "dt"         "em"  
## [33] "embed"      "eventsource" "fieldset"   "figcaption"  
## [37] "figure"     "footer"     "form"       "h1"  
## [41] "h2"         "h3"        "h4"        "h5"
```

Sintaxis

```
tags$a(href = "www.rstudio.com", "RStudio")
```

the list
named tags

the function/tag name
(followed by parentheses)

named arguments
appear as tag attributes
(set boolean attributes to NA)

unnamed arguments
appear inside the tags
(call tags\$...() to create nested tags)

```
<a href="www.rstudio.com">RStudio</a>
```

tags

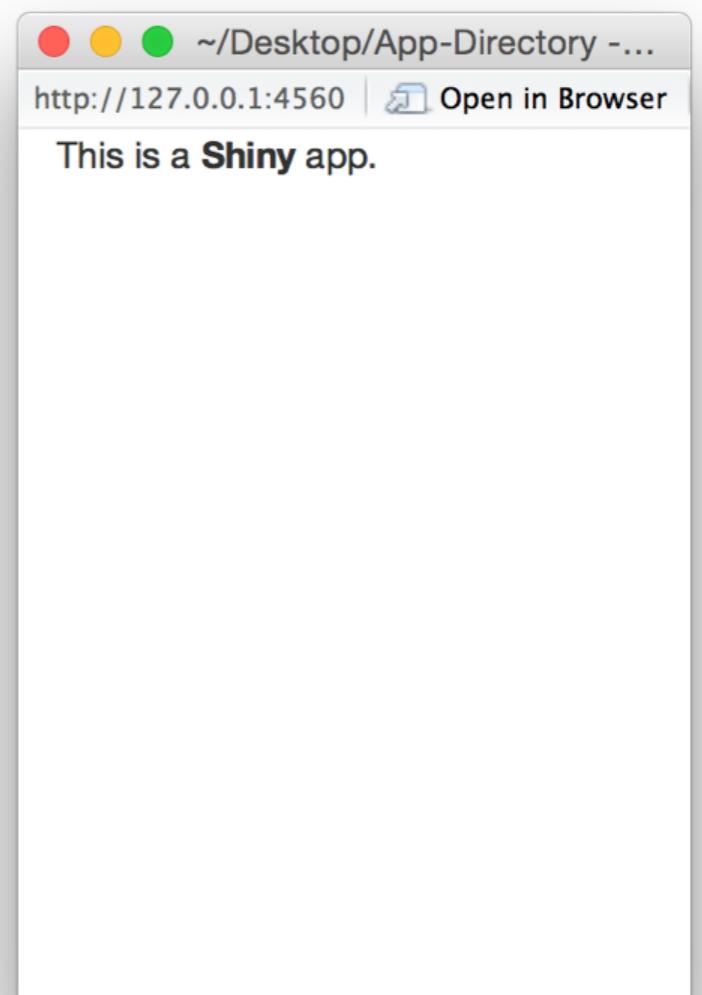
Los tags más usuales tienen una función específica:

Función	Inserta
h1()-h6()	encabezado h1 a h6
a()	enlace
texto	texto plano (sin tag)
p()	párrafo
em()	texto en cursiva
strong()	texto en negrita
code()	texto monoespaciado
br()	salto de línea
hr()	línea horizontal
img()	imagen

anidamiento

Se pueden anidar funciones dentro de otras

```
fluidPage(  
  tags$p("This is a",  
  tags$strong("Shiny"),  
  "app.")  
)
```

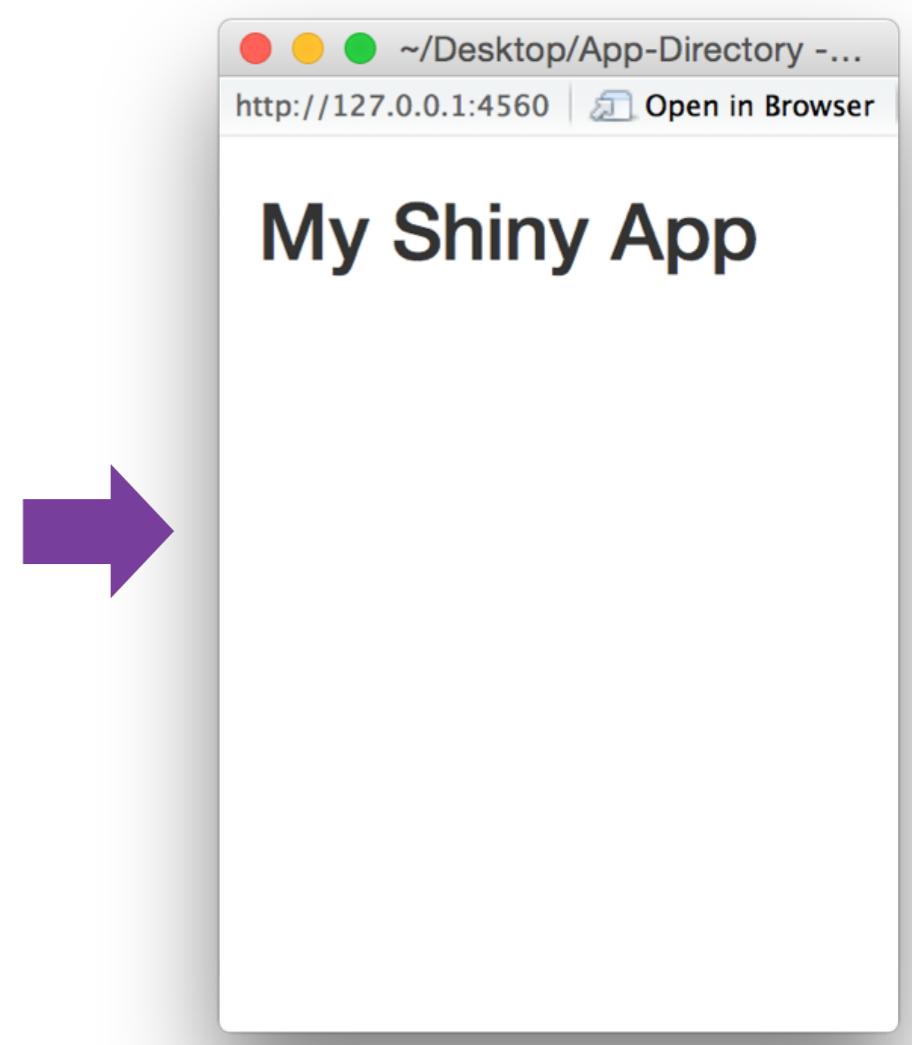


© 2015 RStudio, Inc. All rights reserved.

HTML puro

La función `HTML()` pasa una cadena de texto como HTML a la UI

```
fluidPage(  
  HTML("<h1>My Shiny App</h1>")  
)
```

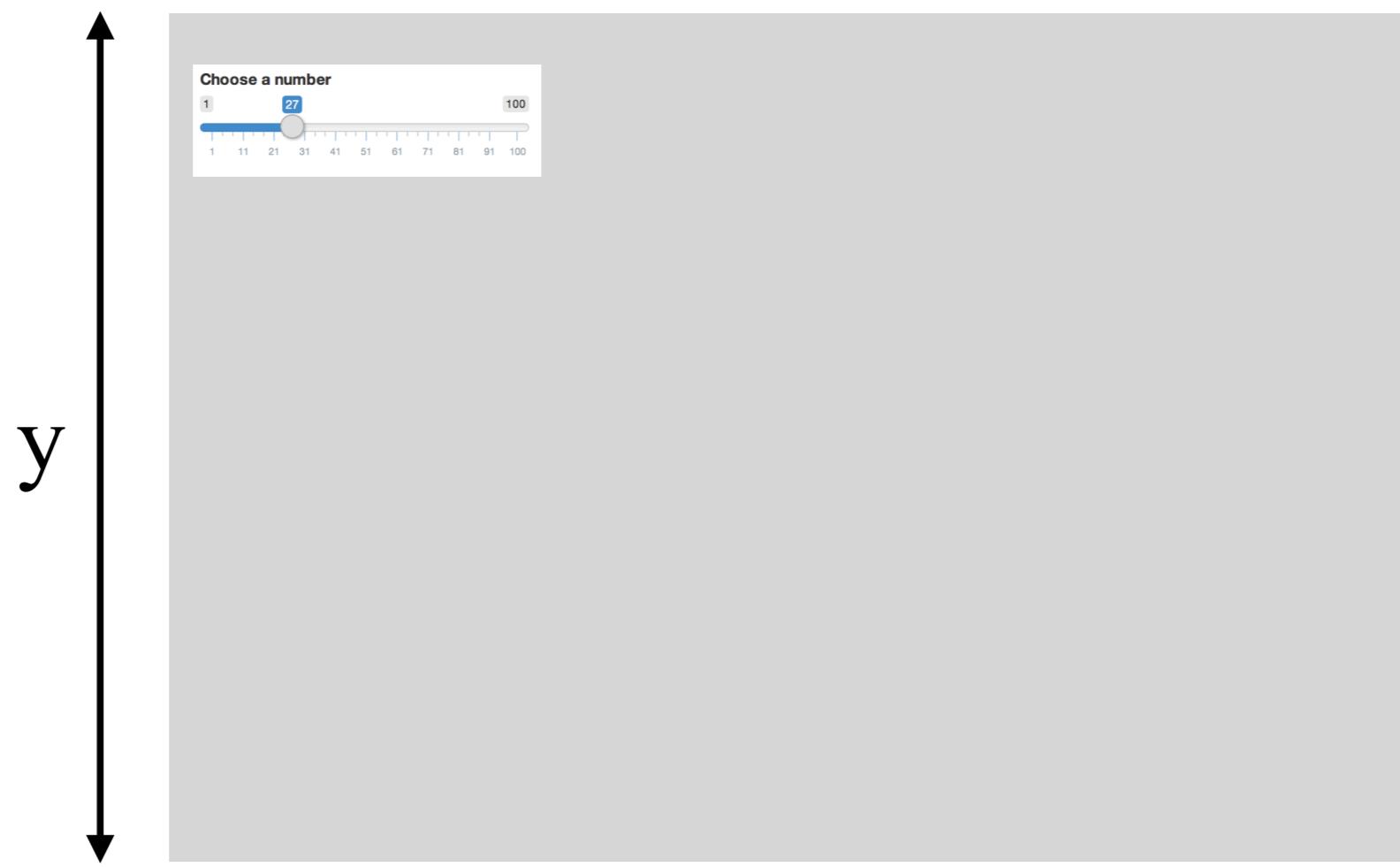


© 2015 RStudio, Inc. All rights reserved.

Ajustar la disposición

Funciones de layout

Las funciones de disposición (*layout*) permiten posicionar los elementos en la aplicación



Funciones de layout

Estas funciones crean el HTML para dividir la UI en una rejilla

`fluidRow()`

```
<div class="row"></div>
```

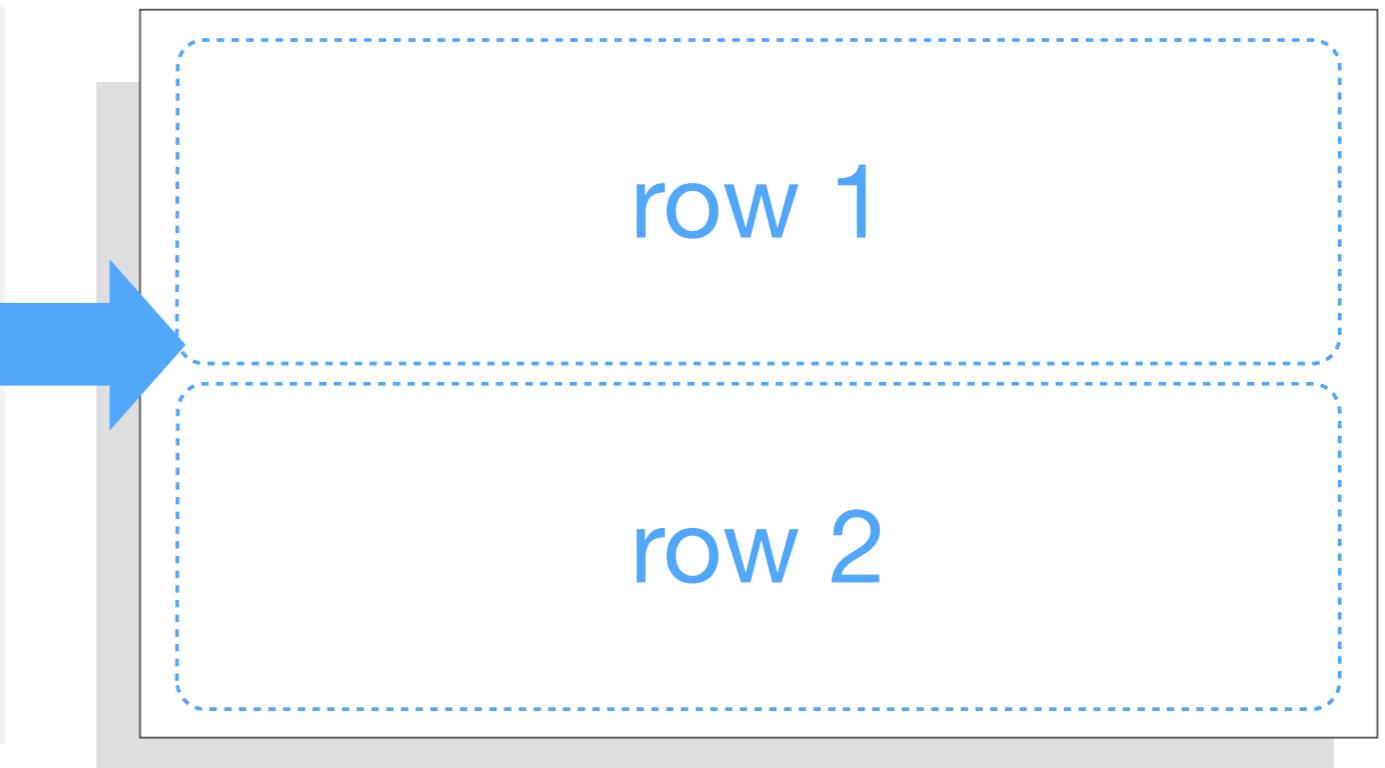
`column(width = 2)`

```
<div class="col-sm-2"></div>
```

fluidRow()

fluidRow() añade filas a la rejilla. Cada fila va debajo de las previas.

```
ui <- fluidPage(  
  fluidRow(),  
  fluidRow()  
)
```

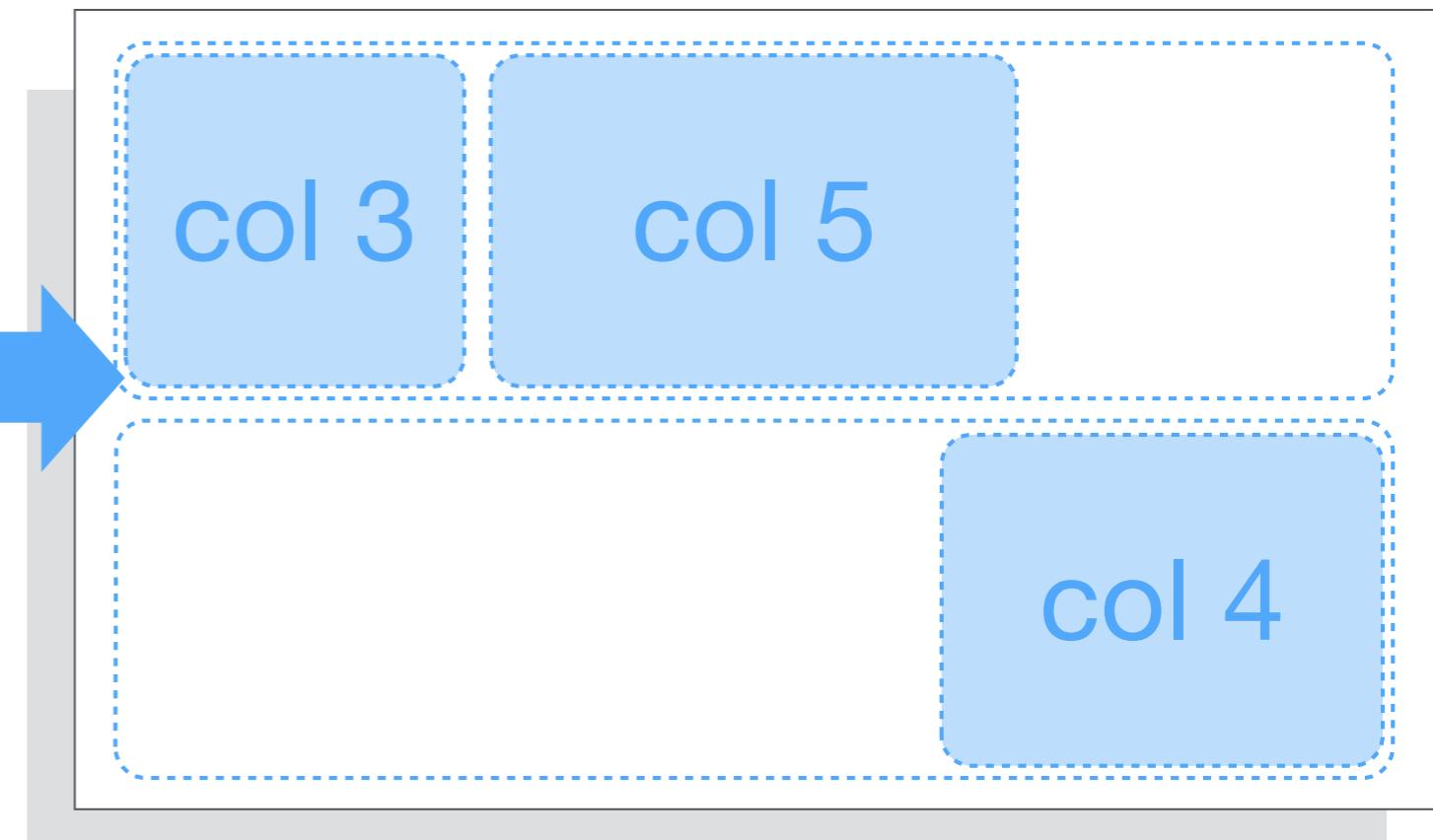


column()

column() añade columna dentro de una fila. Cada columna va a la derecha de la columna previa.

Se puede especificar el ancho y offset de cada columna para un total de 12 columnas por fila.

```
ui <- fluidPage(  
  fluidRow(  
    column(3),  
    column(5)),  
  fluidRow(  
    column(4, offset = 8))
```



Para poner un elemento en la rejilla se llama como argumento a la función de layout.

```
fluidRow("In the row")
```

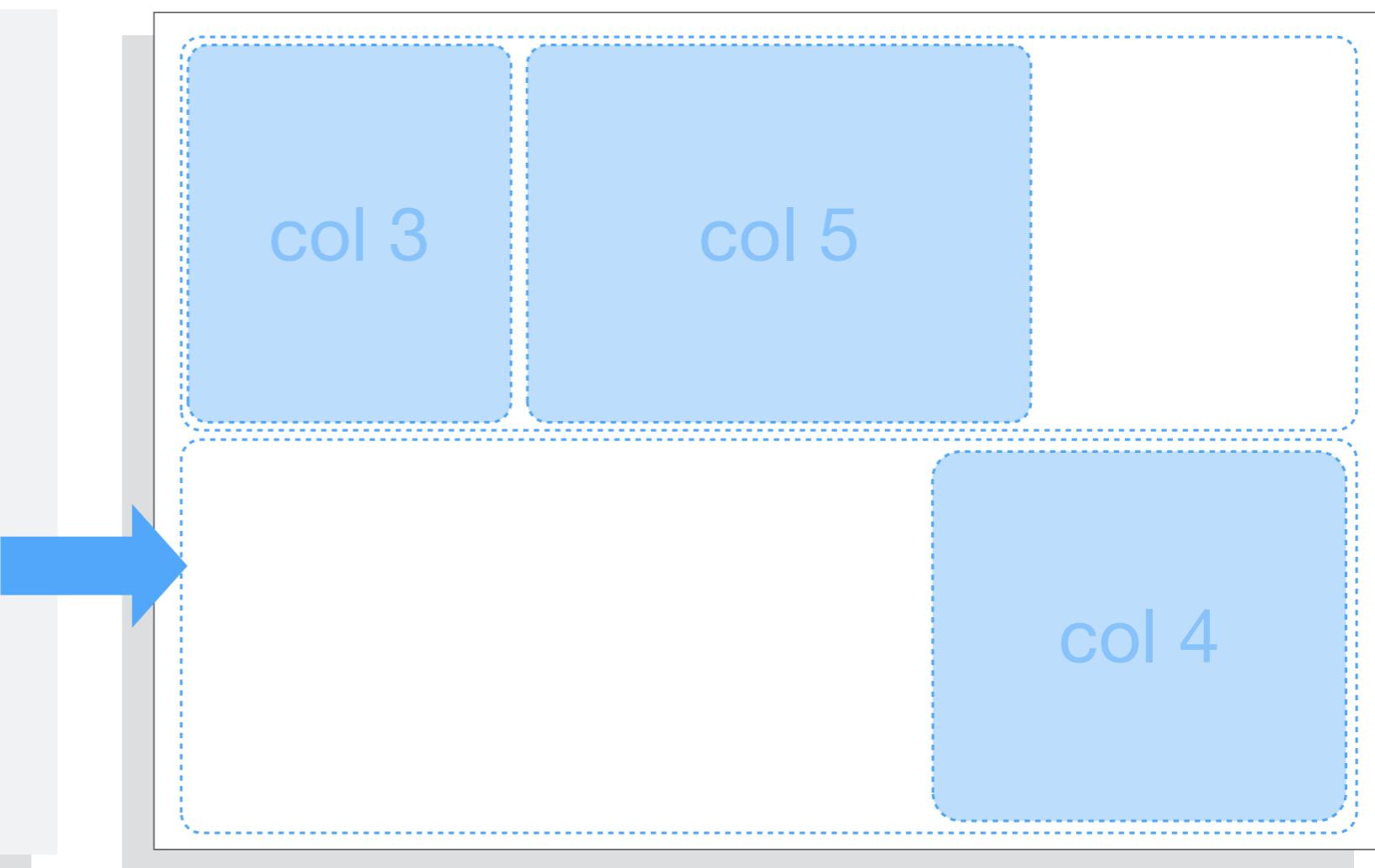
```
<div class="row">In the row</div>
```

```
column(2, plotOutput("hist"))
```

```
<div class="col-sm-2">
  <div id="hist" class="shiny-plot-output"
    style="width: 100% ; height: 400px"></div>
</div>
```

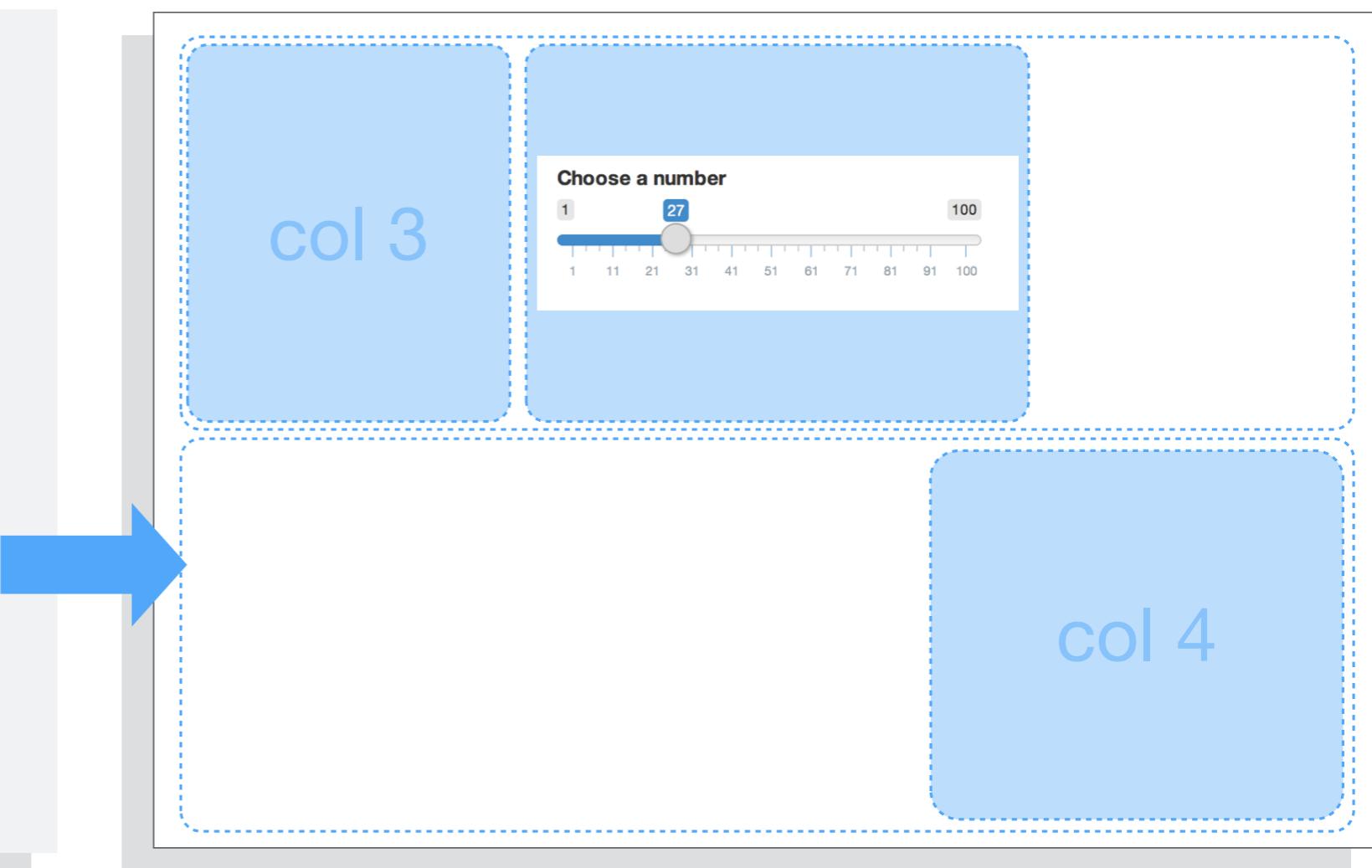
Para poner un elemento en la rejilla se llama como argumento a la función de layout.

```
fluidPage(  
  fluidRow(  
    column(3),  
    column(5)  
>,  
  fluidRow(  
    column(4, offset = 8,)  
>  
>)
```



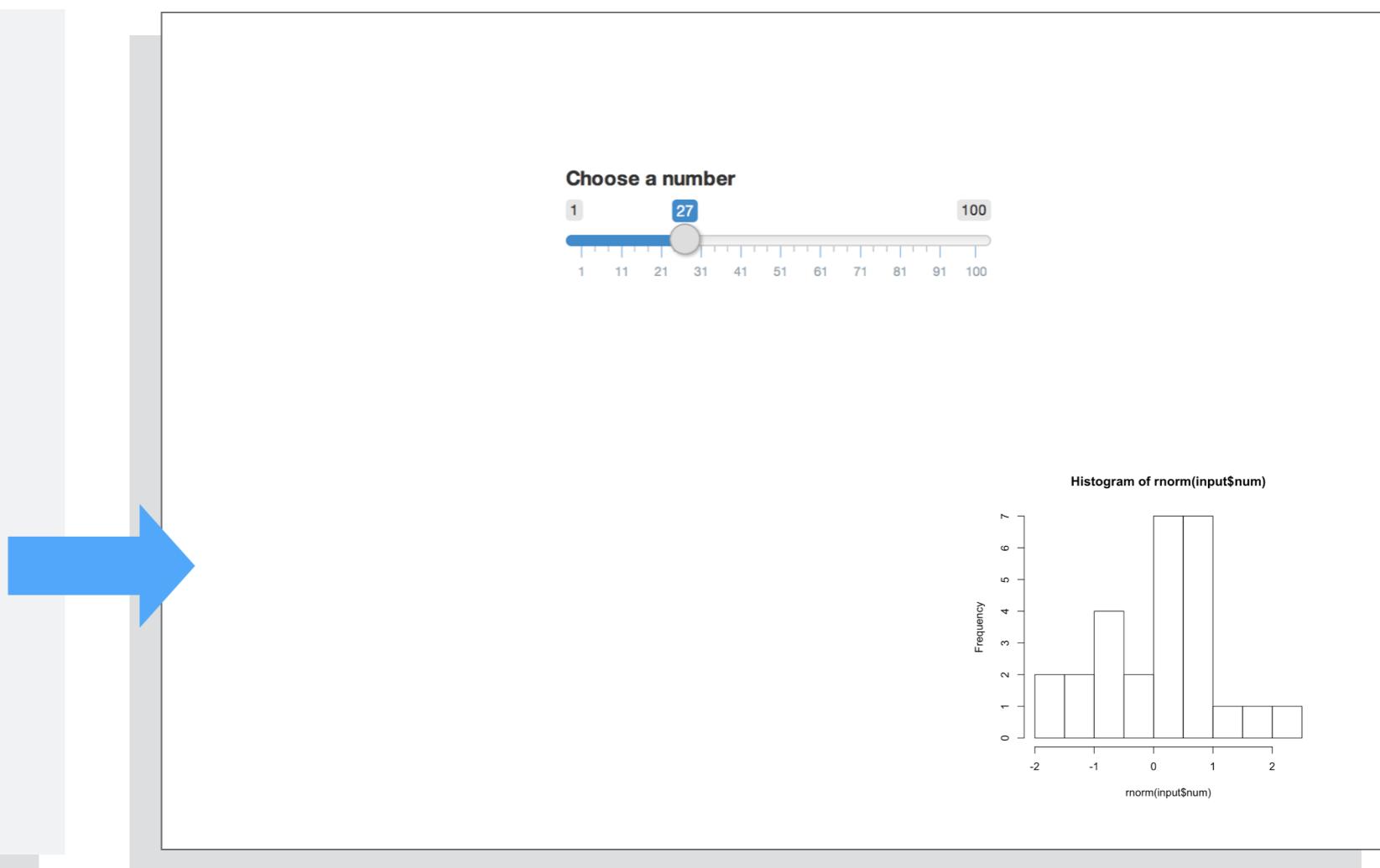
Para poner un elemento en la rejilla se llama como argumento a la función de layout.

```
fluidPage(  
  fluidRow(  
    column(3),  
    column(5, sliderInput(...))  
  ),  
  fluidRow(  
    column(4, offset = 8,  
      plotOutput("hist"))  
  )  
)
```



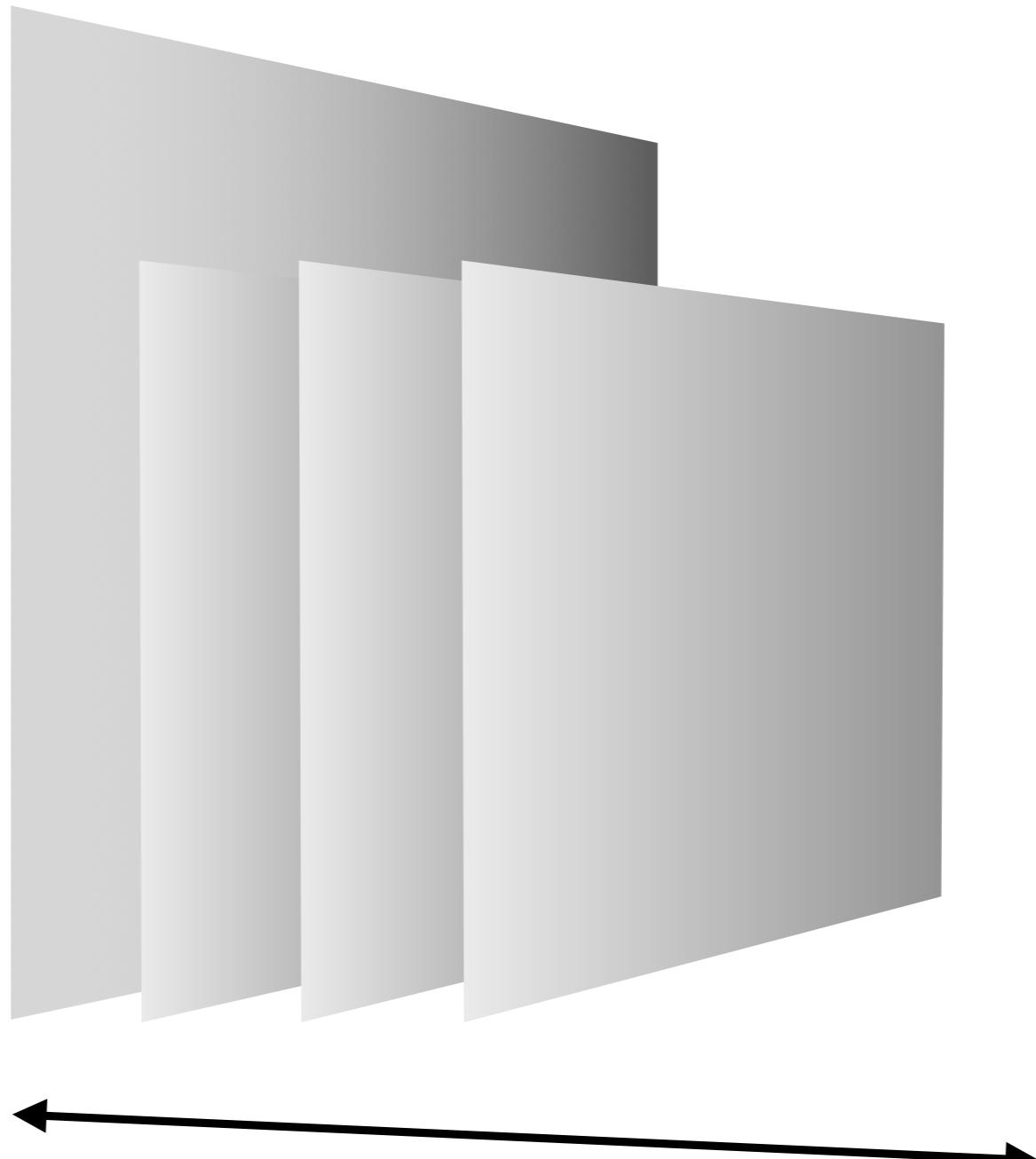
Para poner un elemento en la rejilla se llama como argumento a la función de layout.

```
fluidPage(  
  fluidRow(  
    column(3),  
    column(5, sliderInput(...))  
,  
  fluidRow(  
    column(4, offset = 8,  
      plotOutput("hist"))  
)  
)
```



Ajustar la disposición mediante paneles y capas

Las funciones de *layout* permiten posicionar los elementos en la aplicación



Paneles

Los paneles agrupan múltiples elementos en una única unidad con propiedades propias.

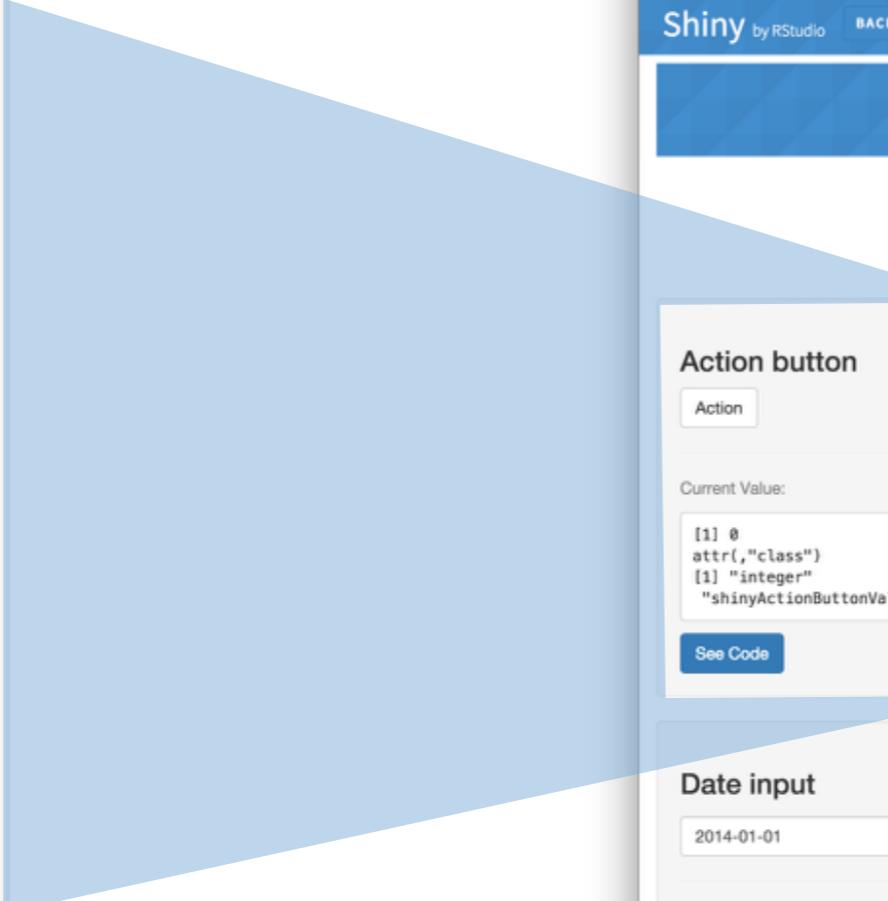
Action button

Action

Current Value:

```
[1] 0  
attr(),"class")  
[1] "integer"  
"shinyActionButtonValue"
```

See Code



Shiny - Widget Gallery

Shiny by RStudio BACK TO GALLERY GET CODE SHARE Search

Shiny Widgets Gallery

For each widget below, the Current Value(s) window displays the value that the widget provides to shinyServer. Notice that the values change as you interact with the widgets.

Action button

Single checkbox

Checkbox group

Date input

Date range

File input

<http://shiny.rstudio.com/gallery/widget-gallery.html>

© 2014 RStudio, Inc. All rights reserved.

Paneles

Los paneles agrupan múltiples elementos en una única unidad con propiedades propias.

absolutePanel()

Panel position set rigidly (absolutely), not fluidly

conditionalPanel()

A JavaScript expression determines whether panel is visible or not.

fixedPanel()

Panel is fixed to browser window and does not scroll with the page

headerPanel()

Panel for the app's title, used with pageWithSidebar()

inputPanel()

Panel with grey background, suitable for grouping inputs

mainPanel()

Panel for displaying output, used with pageWithSidebar()

navlistPanel()

Panel for displaying multiple stacked tabPanels(). Uses sidebar navigation

sidebarPanel()

Panel for displaying a sidebar of inputs, used with pageWithSidebar()

tabPanel()

Stackable panel. Used with navlistPanel() and tabsetPanel()

tabsetPanel()

Panel for displaying multiple stacked tabPanels(). Uses tab navigation

titlePanel()

Panel for the app's title, used with pageWithSidebar()

wellPanel()

Panel with grey background.

tabPanel()

`tabPanel()` crea un conjunto aplicable de elementos en pestañas (*tabs*), cada una con su propia interfaz de usuario.

`tabPanel("Tab 1", ...)`

A title
(for navigation)

elements to
appear in the tab

Pestañas

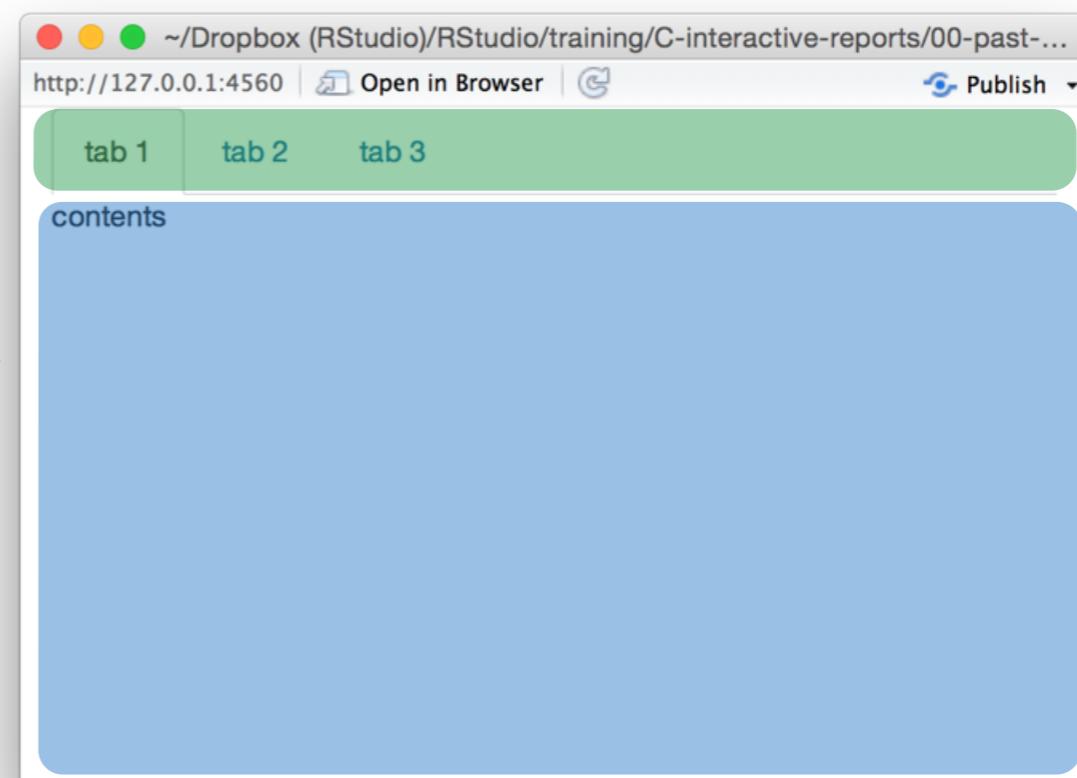
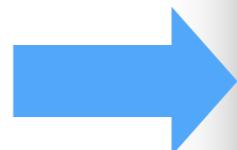
`tabPanel()` se combina con las siguientes funciones:

- `tabsetPanel()`
- `navlistPanel()`
- `navbarPage()`

tabsetPanel()

`tabsetPanel()` combina las pestañas en un único panel separado por *tabs*.

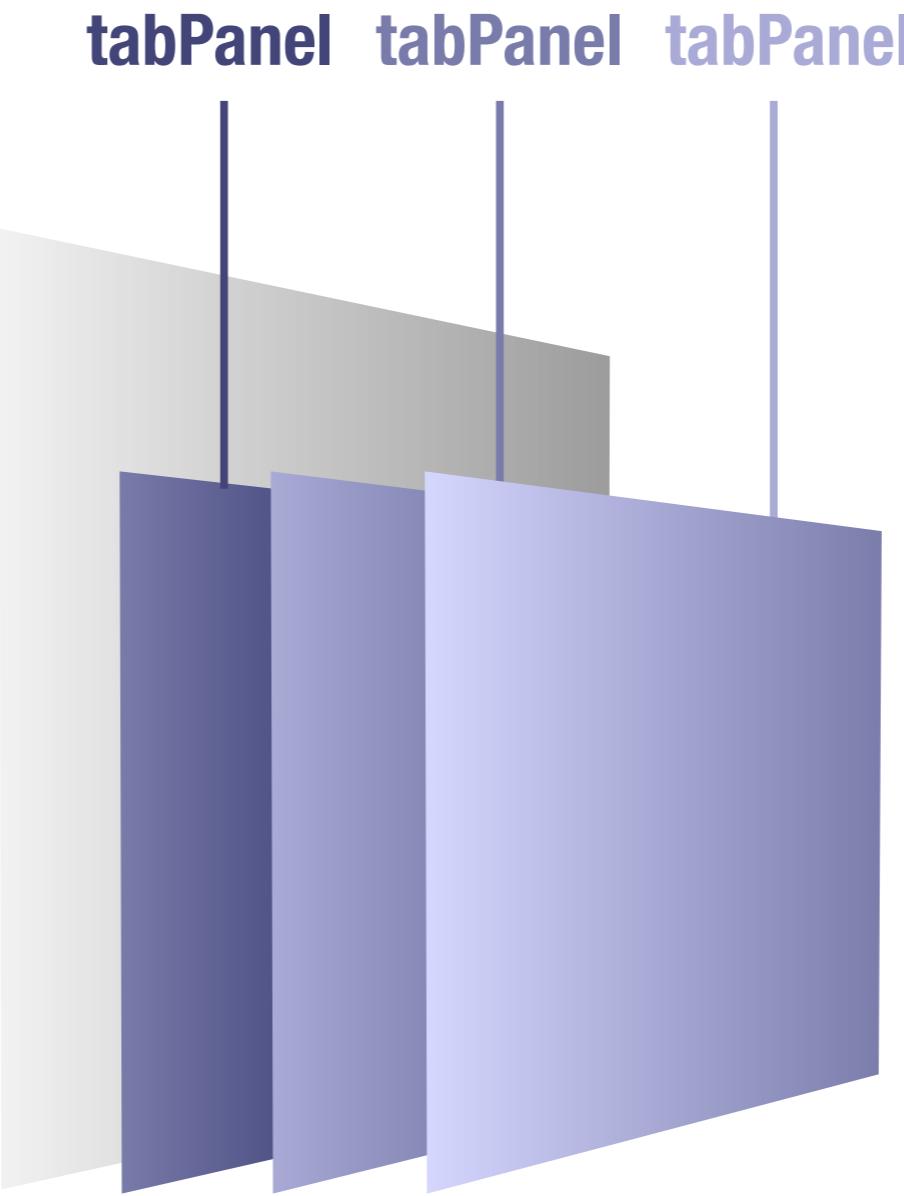
```
fluidPage(  
  tabsetPanel(  
    tabPanel("tab 1", "contents"),  
    tabPanel("tab 2", "contents"),  
    tabPanel("tab 3", "contents")  
  )  
)
```



Navigation

Content

tabsetPanel()

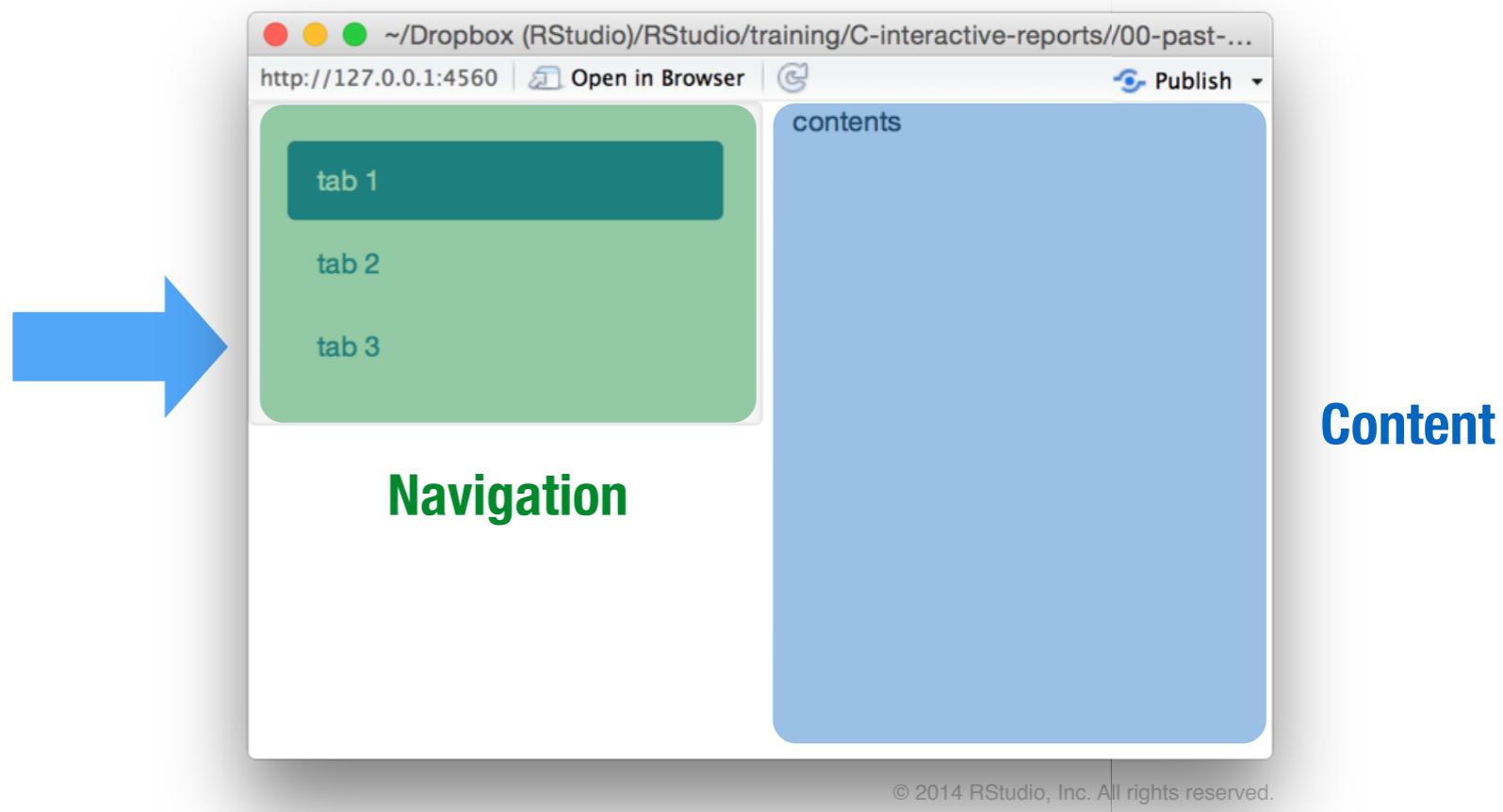


tabsetPanel

navlistPanel()

`navlistPanel()` combina las pestañas en un único panel separado por enlaces.

```
fluidPage(  
  navlistPanel(  
    tabPanel("tab 1", "contents"),  
    tabPanel("tab 2", "contents"),  
    tabPanel("tab 3", "contents")  
  )  
)
```

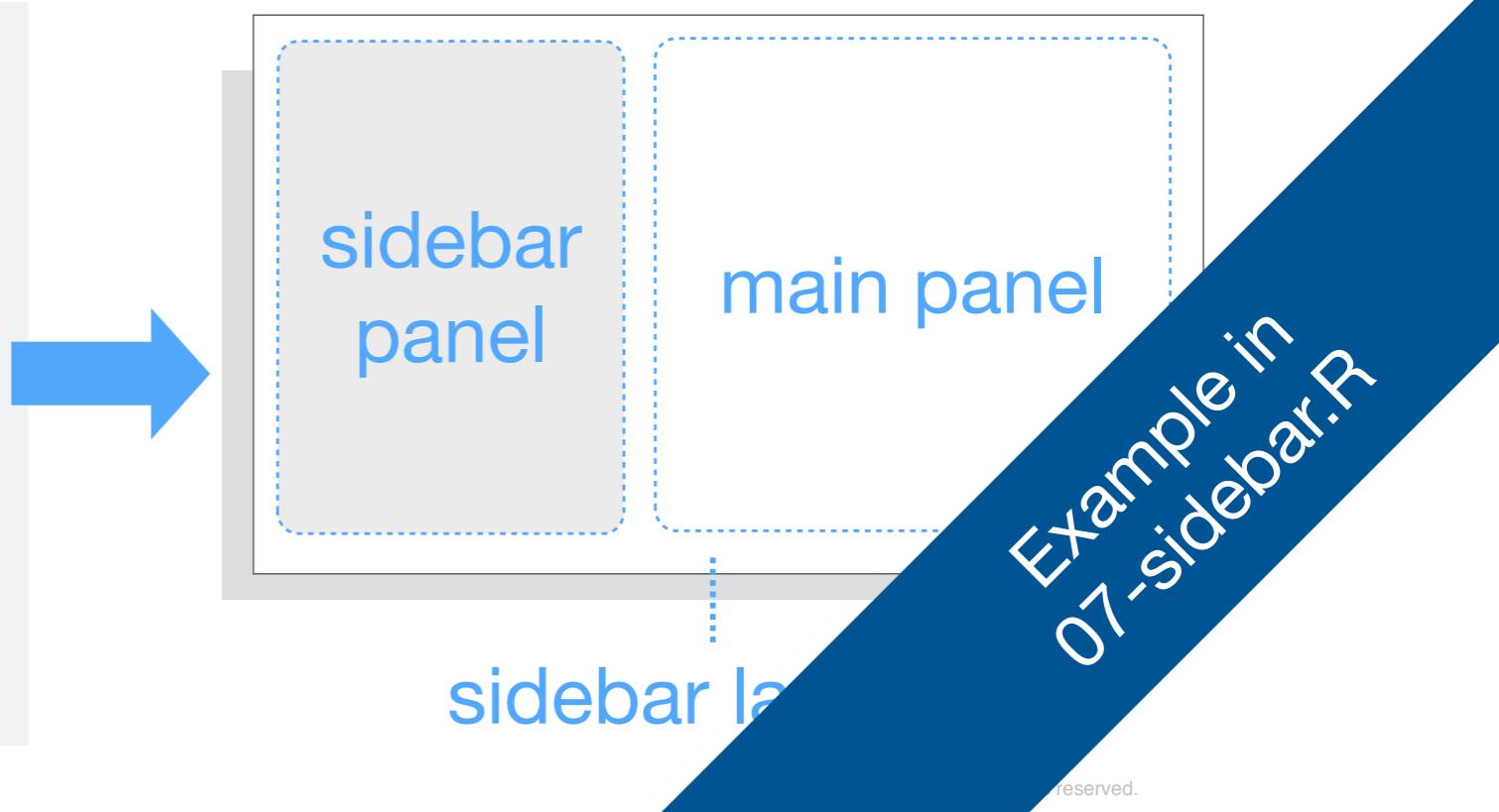


Uso de disposiciones predefinidas

sidebarLayout()

Divide la aplicación en dos secciones con
sidebarPanel() y mainPanel()

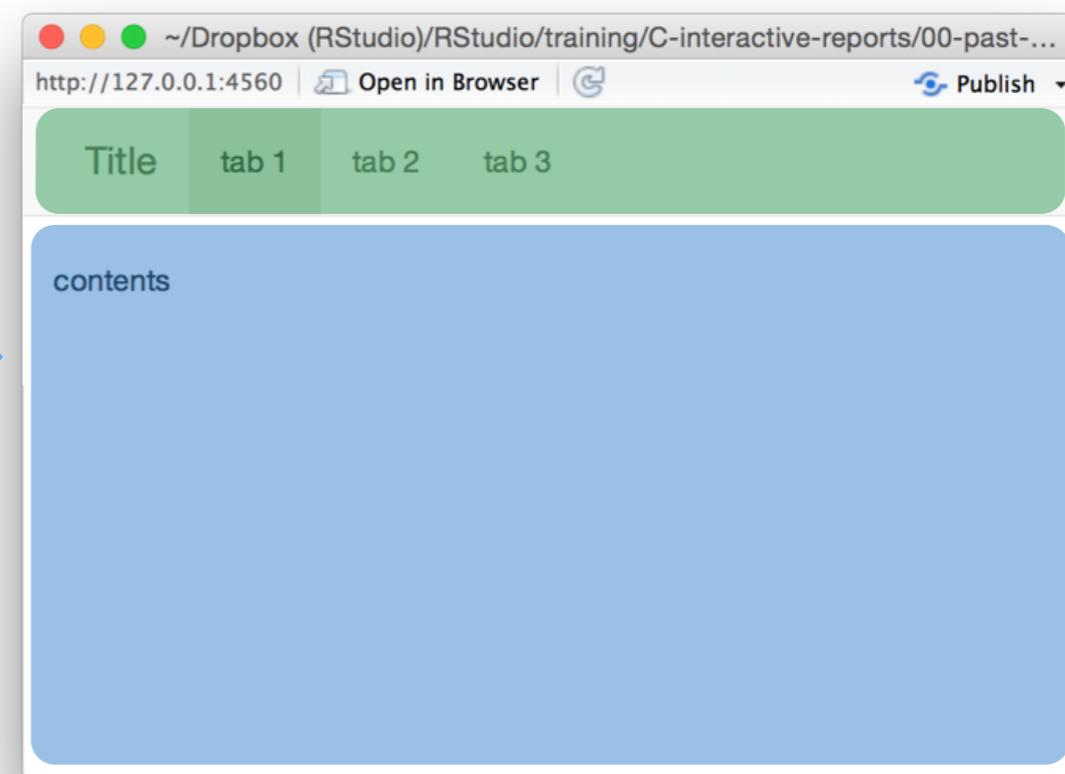
```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(),  
    mainPanel()  
)  
)
```



navbarPage()

navbarPage() combina las pestañas en una única página, reemplazando a fluidPage(). Necesita un título.

```
navbarPage(title = "Title",  
          tabPanel("tab 1", "contents"),  
          tabPanel("tab 2", "contents"),  
          tabPanel("tab 3", "contents"))
```

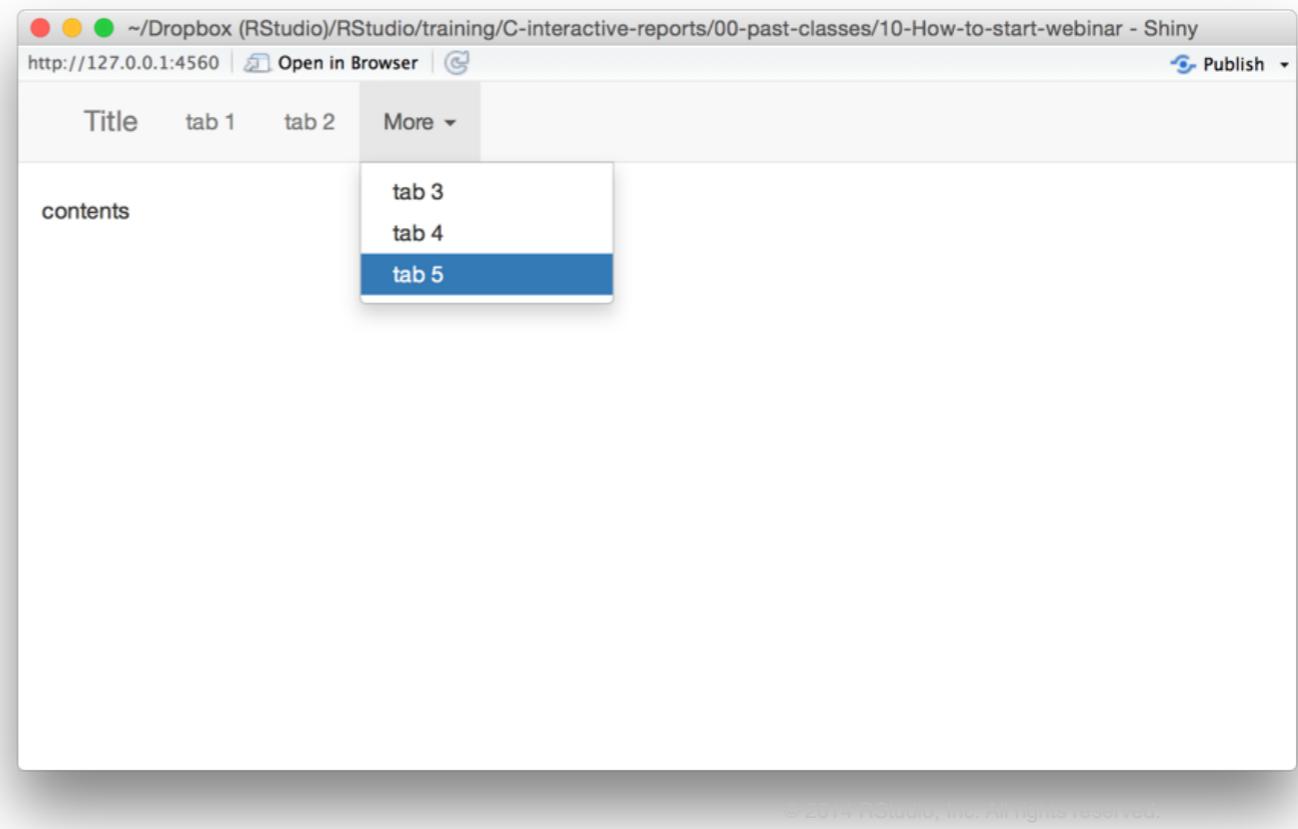


© 2014 RStudio, Inc. All rights reserved.

navbarMenu()

navbarMenu() combina los enlaces a los tags en un menú desplegable de navbarPage()

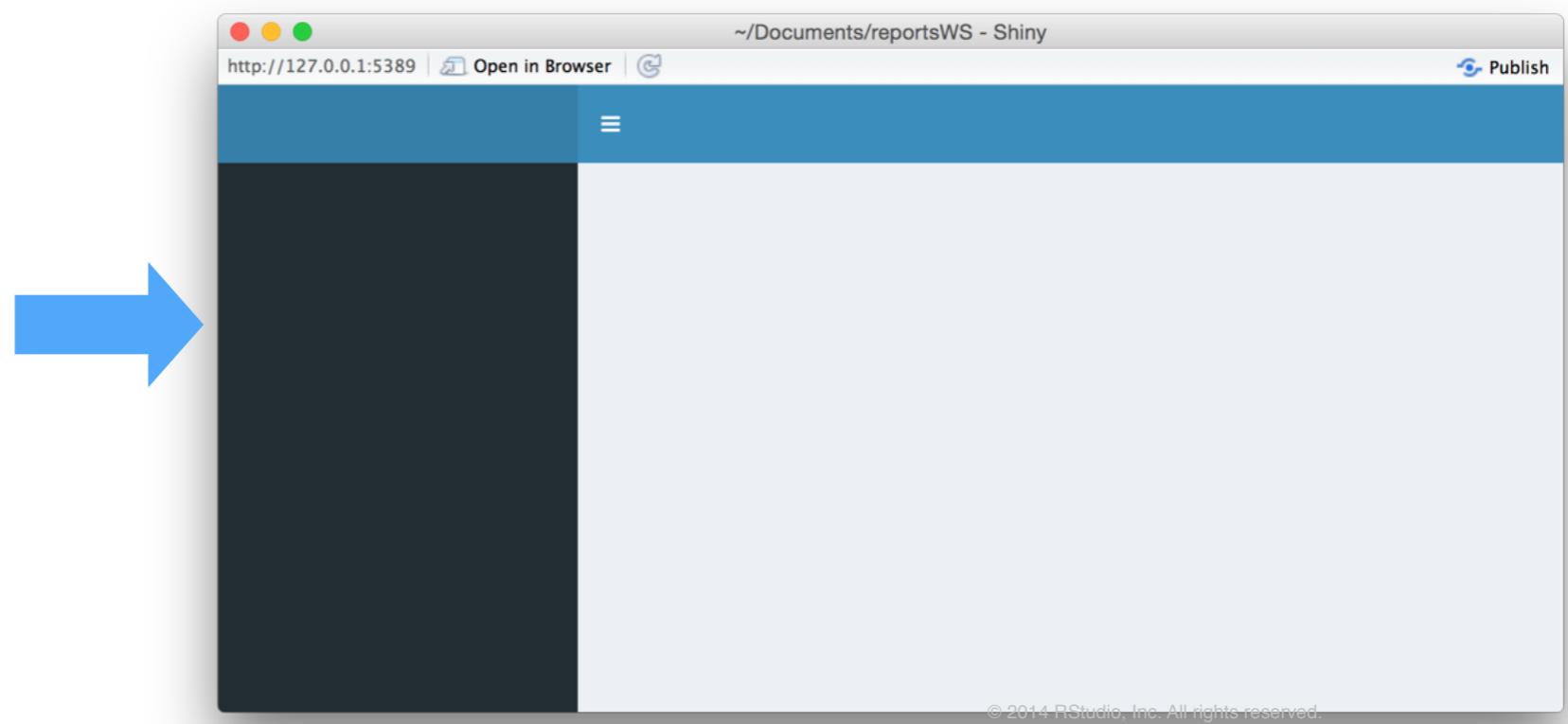
```
navbarPage(title = "Title",
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  navbarMenu(title = "More",
    tabPanel("tab 3", "contents"),
    tabPanel("tab 4", "contents"),
    tabPanel("tab 5", "contents"))
)
```



dashboardPage()

dashboardPage() es un *layout* con tres componentes, que viene en el paquete shinydashboard

```
library(shinydashboard)  
ui <- dashboardPage(  
  dashboardHeader(),  
  dashboardSidebar(),  
  dashboardBody()  
)
```



Utiliza hojas de estilo

CSS

Hojas de estilo

Shiny utiliza el framework CSS Bootstrap 3.

Sobre este CSS podemos personalizar el estilo de una aplicación de tres maneras:

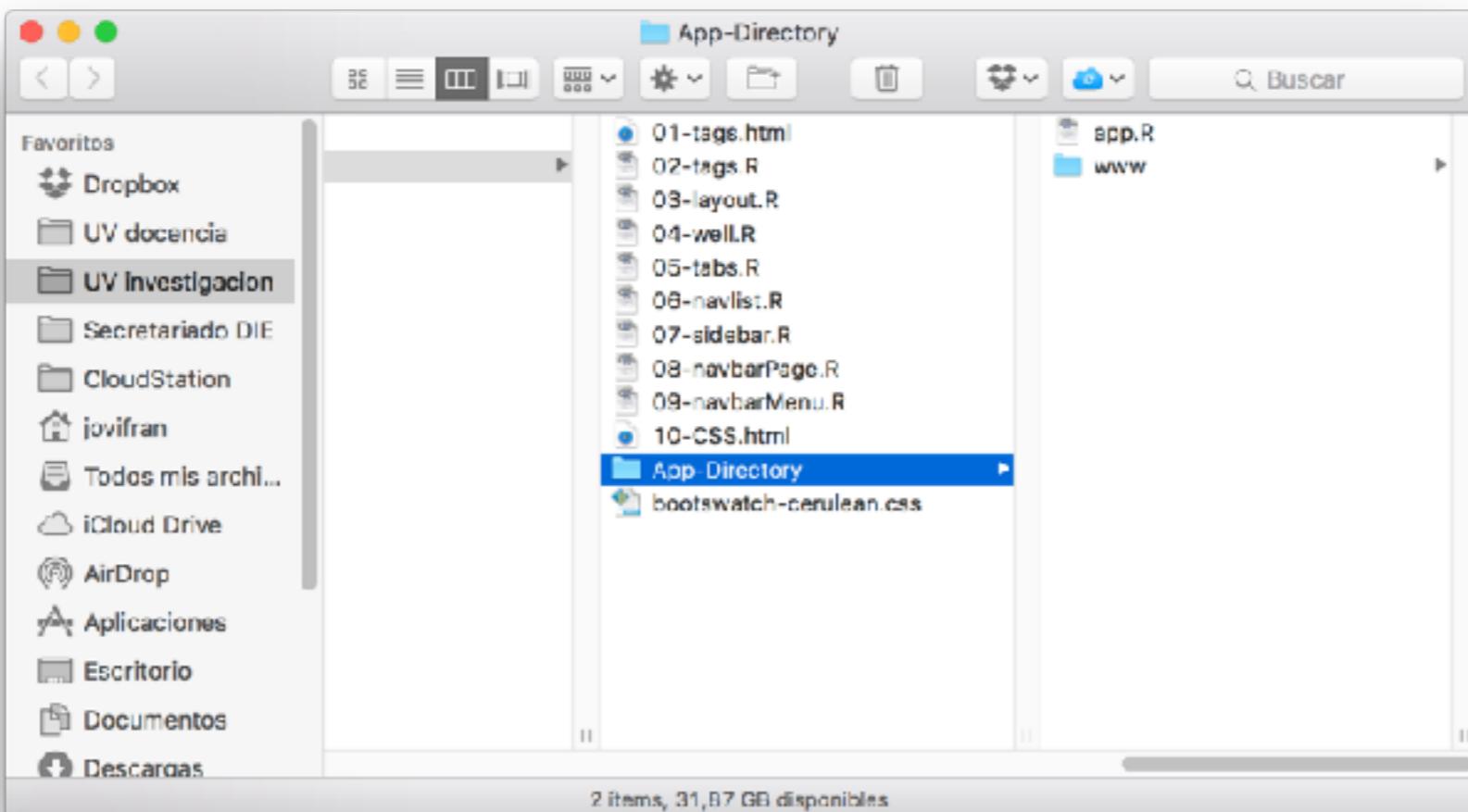
1. Enlace a un CSS externo
2. Uso de un CSS global en la cabecera (*header*)
3. Uso de CSS individuales en el atributo *style* de cada tag

Distribuir una aplicación Shiny

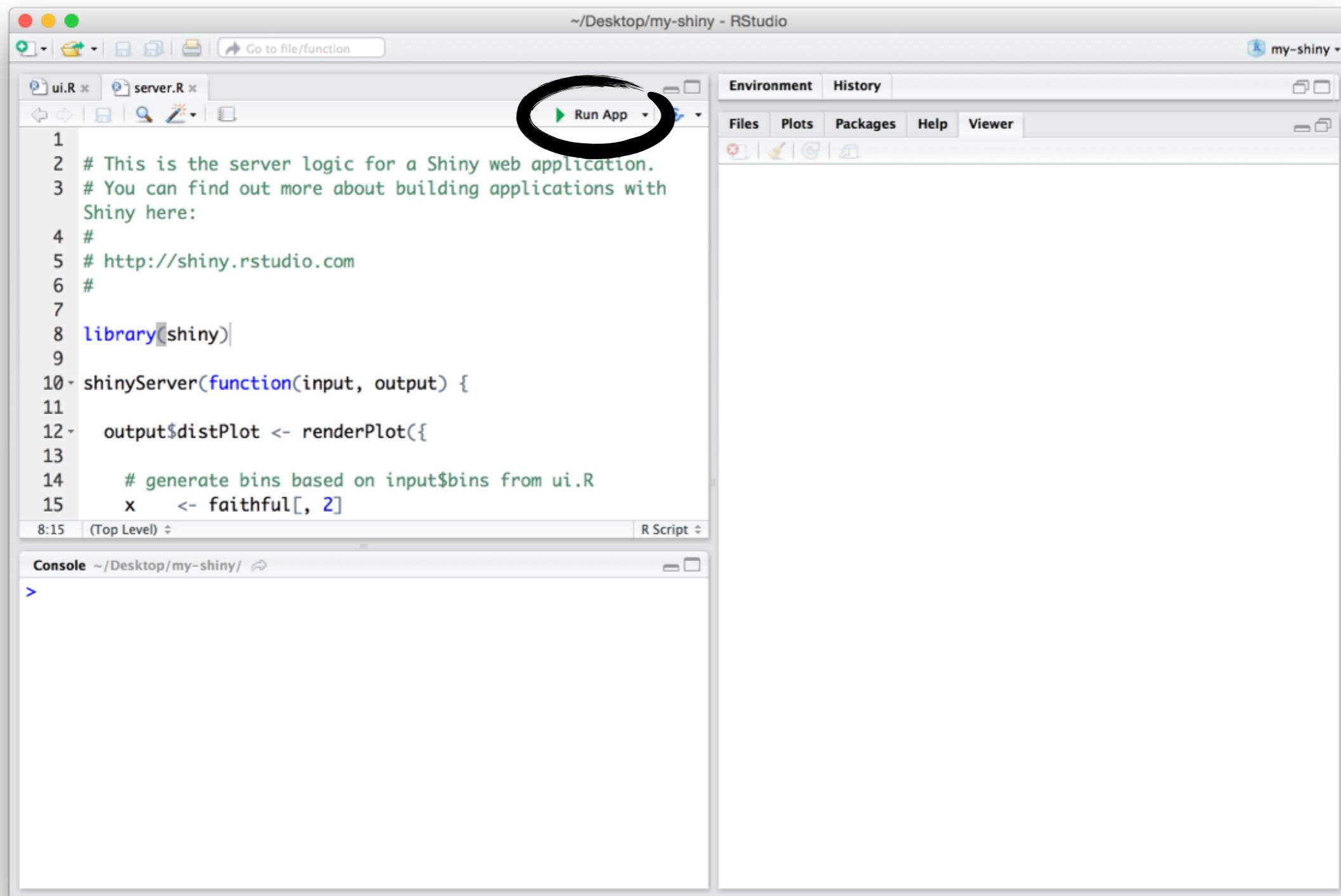
Guardar una aplicación

Todos los archivos de la aplicación deben contenerse en un único directorio:

- **app.R** (*script de la app terminado en una llamada a shinyApp()*)
- **datasets, imágenes, css, scripts de soporte, etc.**



Ejecutar en local



© CC 2015 RStudio, Inc.

Ejecutar en remoto

- Alojar la aplicación en el servidor [shinyapps.io](https://www.shinyapps.io) de RStudio:

<https://www.shinyapps.io>

- Instalar un Shiny Server en un servidor Linux:

<https://www.rstudio.com/products/shiny/shiny-server/>