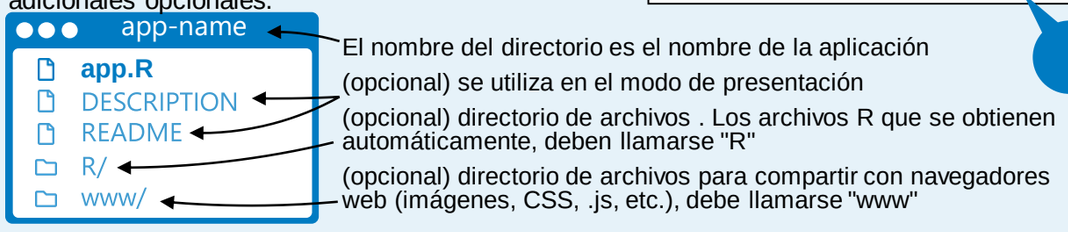
Shiny para R : : GUÍA RÁPIDA

Creando una aplicación

Una aplicación Shiny es una página web (ui) conectada a un equipo que ejecuta una sesión de R en vivo (servidor).

Los usuarios pueden manipular la interfaz de usuario, lo que hará que el servidor actualice las pantallas de la interfaz de usuario (mediante la ejecución de código R).

Guarde su plantilla como app.R. Mantenga su aplicación en un directorio junto con archivos adicionales opcionales.



Launch apps stored in a directory with **runApp(**<path to directory>**)**.

Para generar la plantilla, escriba shinyapp y presione Tab en el IDE de RStudio o vaya a File > New Project > New Directory > Shiny Application

En ui se anidan funciones R para crear una interfaz HTML

Indicar al servidor cómo representar las salidas y responder a las entradas con R

```
# app.R
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server <- function(input, output, session) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}

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

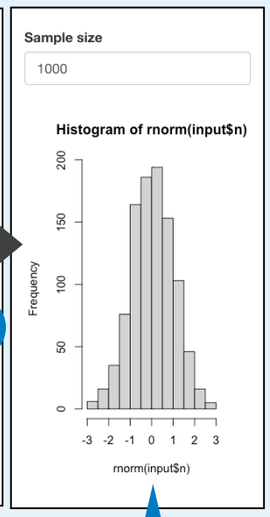
Personalizar la interfaz de usuario con funciones de diseño

Agregue entradas con funciones *Input()

Agregue salidas con funciones *Output()

Envolver el código en las funciones render*() antes de guardarlo en la salida

Consulte las entradas de la interfaz de usuario con input$<id> y las salidas con output$<id>



¡Llama a shinyApp() para combinar la interfaz de usuario y el servidor en una aplicación interactiva!

Para ver ejemplos anotados de aplicaciones Shiny, ejecute **runExample(**<example name>**)**. Ejecute **runExample()** sin argumentos para una lista de nombres de ejemplo.

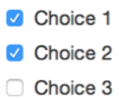
Inputs

Recopile valores del usuario.

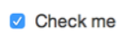
Acceda al valor actual de un objeto de entrada con **input$<inputId>**. Los valores de entrada son **reactive**.

actionButton(inputId, label, icon, width, …**)**

actionLink(inputId, label, icon, …**)**



checkboxGroupInput(inputId, label, choices, selected, inline, width, choiceNames, choiceValues**)**



checkboxInput(inputId, label, value, width**)**

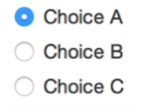
dateInput(inputId, label, value, min, max, format, startview, weekstart, language, width, autoclose, datesdisabled, daysofweekdisabled**)**

dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator, width, autoclose**)**

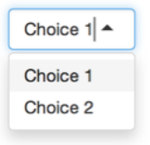
fileInput(inputId, label, multiple, accept, width, buttonLabel, placeholder**)**

numericInput(inputId, label, value, min, max, step, width**)**

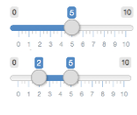
passwordInput(inputId, label, value, width, placeholder**)**



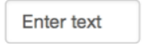
radioButtons(inputId, label, choices, selected, inline, width, choiceNames, choiceValues**)**



selectInput(inputId, label, choices, selected, multiple, selectize, width, size**)** También **selectizeInput()**



sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post, timeFormat, timezone, dragRange**)**



textInput(inputId, label, value, width, placeholder**)** También **textAreaInput()**

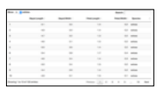
Compartir

Comparte tu aplicación de tres maneras:

1. **Alójala en shinyapps.io**, sistema basado en la nube, servicio de Posit. Para implementar Shiny:
   - Crea una cuenta gratis profesional en **shinyapps.io**
   - Haga clic en el icono Publicar en el IDE de RStudio o ejecute: **rsconnect::deployApp(**"<path to directory>"**)**
2. **Compre Posit Connect**, una plataforma de publicación para R y Python. posit.co/products/enterprise/connect/
3. **Cree su propio Shiny Server** **posit.co/products/open-source/shinyserver/**

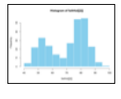
Salidas

render*() y *Output() estas funciones trabajan juntas para añadir salidas de R a laUI.



DT::**renderDataTable(**expr, options, searchDelay, callback, escape, env, quoted, outputArgs**)** — **dataTableOutput(**outputId**)**

renderImage(expr, env, quoted, deleteFile, outputArgs**)** — **imageOutput(**outputId, width, height, click, dblclick, hover, brush, inline**)**



renderPlot(expr, width, height, res, …, alt, env, quoted, execOnResize, outputArgs**)** — **plotOutput(**outputId, width, height, click, dblclick, hover, brush, inline**)**

renderPrint(expr, env, quoted, width, outputArgs**)** — **verbatimTextOutput(**outputId, placeholder**)**

renderTable(expr, striped, hover, bordered, spacing, width, align, rownames, colnames, digits, na, …, env, quoted, outputArgs**)** — **tableOutput(**outputId**)**

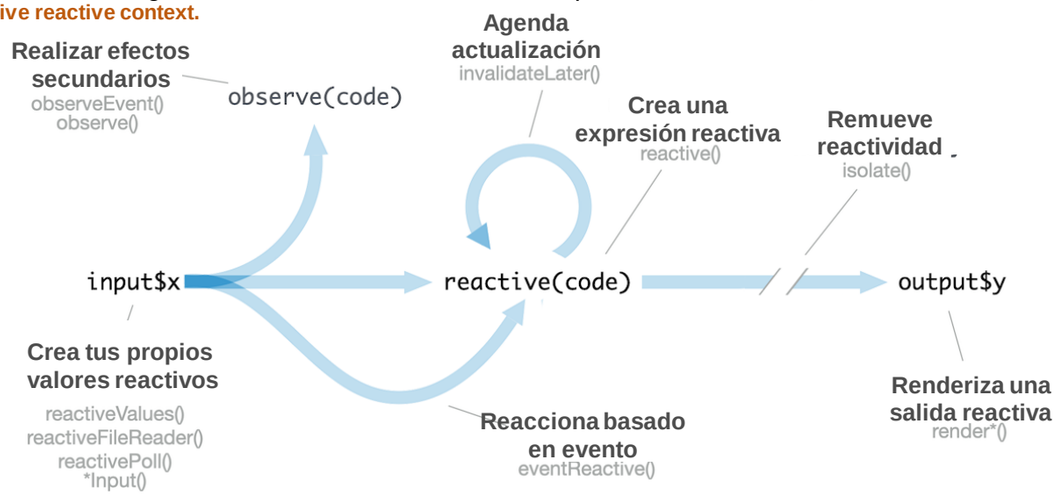
renderText(expr, env, quoted, outputArgs, sep**)** — **textOutput(**outputId, container, inline**)**

renderUI(expr, env, quoted, outputArgs**)** — **uiOutput(**outputId, inline, container, …**)** **htmlOutput(**outputId, inline, container, …**)**

Estos son los tipos de salida principales. Consulte **htmlwidgets.org** para más opciones.

Reactividad

Los valores reactivos funcionan junto con las funciones reactivas. Llame a un valor reactivo desde dentro de los argumentos de una de estas funciones para evitar el error **Operation not allowed without an active reactive context.**



CREA TUS PROPIOS VALORES REACTIVOS

```
# *Input() example
ui <- fluidPage(
 textInput("a","","A")
)
```

```
#reactiveVal example
server <- function(input,output){
 rv <- reactiveVal()
 rv$number <- 5
}
```

***Input() functions**

Cada función de entrada crea un valor reactivo almacenado como **input$<inputId>**.

reactiveVal(…)

Crea un único objeto de valores reactivos.

reactiveValues(…)

Crea una lista de nombres, valores reactivos.

CREACIÓN DE EXPRESIONES REACTIVAS

```
ui <- fluidPage(
 textInput("a","","A"),
 textInput("z","","Z"),
 textOutput("b"))

server <- function(input,output){
 re <- reactive({
  paste(input$a,input$z)
 })
 output$b <- renderText({
  re()
 })
}
shinyApp(ui, server)
```

reactive(x, env, quoted, label, domain**)**

Expresiones reactivas:

- **cache** su valor para reducir el cómputo
- se puede llamar a otro lugar
- notifican dependencias cuando se invalidan

Llame a la expresión con sintaxis de función, p. ej. re().

REACCIONAR EN FUNCIÓN DEL EVENTO

```
ui <- fluidPage(
 textInput("a","","A"),
 actionButton("go","Go"),
 textOutput("b")
)

server <- function(input,output){
 re <- eventReactive(
  input$go,{input$a}
 )
 output$b <- renderText({
  re()
 })
}

shinyApp(ui, server)
```

eventReactive(eventExpr, valueExpr, event.env, event.quoted, value.env, value.quoted, ..., label, domain, ignoreNULL, ignoreInit**)**

Crea una expresión reactiva con código en el 2º argumento que solo se invalida cuando cambian los valores reactivos en el 1º argumento.

RENDERIZAR SALIDA REACTIVA

```
ui <- fluidPage(
 textInput("a","","A"),
 textOutput("b")
)

server <- function(input,output){
 output$b <-
  renderText({
   input$a
  })
}

shinyApp(ui, server)
```

render*() functions

Crea un objeto para mostrarlo. Volverá a ejecutar el código en el cuerpo para reconstruir el objeto cada vez que cambie un valor reactivo en el código.

Guarde los resultados en **output$<outputId>**.

REALIZAR EFECTOS SECUNDARIOS

```
ui <- fluidPage(
 textInput("a","","A"),
 actionButton("go","Go")
)

server <- function(input,output){
 observeEvent(
  input$go,{
  print(input$a)
  }
 )
}

shinyApp(ui, server)
```

observe(x, env**)**

Crea un observador a partir de la expresión dada.

observeEvent(eventExpr, handlerExpr, event.env, event.quoted, handler.env, handler.quoted, ..., label, suspended, priority, domain, autoDestroy, ignoreNULL, ignoreInit, once**)**

Ejecuta código en el 2º argumento cuando cambian los valores reactivos en el 1º argumento.

ELIMINAR LA REACTIVIDAD

```
ui <- fluidPage(
 textInput("a","","A"),
 textOutput("b")
)

server <- function(input,output){
 output$b <-
  renderText({
   isolate({input$a})
  })
}

shinyApp(ui, server)
```

isolate(expr**)**

Ejecuta un bloque de código. Devuelve una copia no reactiva de los resultados.

UI - La interfaz de usuario de una aplicación es un documento HTML.

Usa las funciones de Shiny para ensamblar este HTML con R.

```
fluidPage(
 textInput("a","")
)
```

Returns HTML

```
## <div class="container-fluid">
##  <div class="form-group shiny-input-container">
##    <label for="a"></label>
##    <input id="a" type="text"
##      class="form-control" value=""/>
##  </div>
## </div>
```

Agregue elementos HTML estáticos con etiquetas, una lista de funciones paralelas a las etiquetas HTML comunes, por ejemplo, tags$a(). Los argumentos sin nombre se pasarán a la etiqueta; Los argumentos con nombre se convertirán en atributos de etiqueta.

Ejecute **names(tags)** para obtener una lista completa.

```
tags$h1("Header") -> <h1>Header</h1>
```

Las etiquetas más comunes tienen funciones contenedoras. No es necesario anteponer el prefijo a sus nombres **tags$**

```
ui <- fluidPage(
 h1("Header 1"),
 hr(),
 br(),
 p(strong("bold")),
 p(em("italic")),
 p(code("code")),
 a(href="", "link"),
 HTML("<p>Raw html</p>")
)
```

Para incluir un archivo CSS, use **includeCSS()**, o

1. Coloque el archivo en el subdirectorio **www**
2. Enlace a él con:

```
tags$head(tags$link(rel = "stylesheet",
 type = "text/css", href = "<file name>"))
```

Para incluir JavaScript, use **includeScript()** o

1. Coloque el archivo en el subdirectorio **www**
2. Enlace a él con:

```
tags$head(tags$script(src = "<file name>"))
```

IMÁGENES Para incluir una imagen:

1. Coloquela en el subdirectorio **www**
2. Enlacelal con img(src="<file name>")

Diseños

Use el paquete bslib para diseñar la aplicación y sus componentes.

DISEÑOS DE PÁGINA

Diseños de panel de control

page_sidebar() Una página de la barra lateral

page_navbar() Aplicación de varias páginas con una barra de navegación superior

page_fillable() Un diseño de página que rellena la pantalla

Diseños básicos

page() **page_fluid()** **page_fixed()**

DISEÑOS DE INTERFAZ DE USUARIO

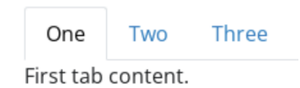
Varias columnas

layout_columns() Organice los elementos de la interfaz de usuario en la cuadrícula CSS de 12 columnas de Bootstrap

layout_column_wrap() Organizar los elementos en una cuadrícula de columnas de igual anchura

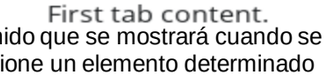
Múltiples paneles

navset_tab()



navset_pill()

navset_underline()



nav_panel() Contenido que se mostrará cuando se seleccione un elemento determinado

nav_menu() Crear un menú de elementos de navegación

nav_item() Colocar contenido arbitrario en el panel de navegación

nav_spacer() Adición de espaciado entre elementos de navegación

También actualice dinámicamente los contenedores de navegación con **nav_select()**, **nav_insert()**, **nav_remove()**, **nav_show()**, **nav_hide()**.

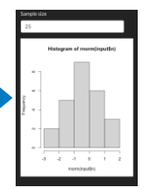
Diseño de la barra lateral

sidebar() **layout_sidebar()** **toggle_sidebar()**

Temas

Use el paquete bslib para agregar temas existentes a la interfaz de usuario de su aplicación Shiny o cree los suyos propios.

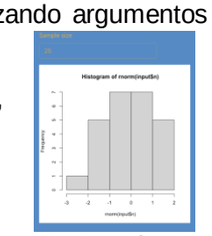
```
library(bslib)
ui <- fluidPage(
 theme = bs_theme(
  bootswatch = "darkly",
  ...
 )
)
```



bootswatch_themes() Obtén una lista de temas.

Crea tu propio tema personalizando argumentos individuales.

bs_theme(bg = "#558AC5", fg = "#F9B02D", …**)**



?bs_theme para obtener una lista completa de argumentos.

bs_themer() Colóquelo dentro de la función de servidor para usar el widget de temas interactivo.