

# **TALLER SHINY**

**Diplomado Data Science**  
**Felipe Peña Graf**



**REACTIVIDAD**



# REACTIVIDAD

	A	B	C	D
1				
2				=A3*B3
3	21	5		105
4				



	A	B	C	D
1				
2				=A3*B3
3	21	10		210
4				
5				

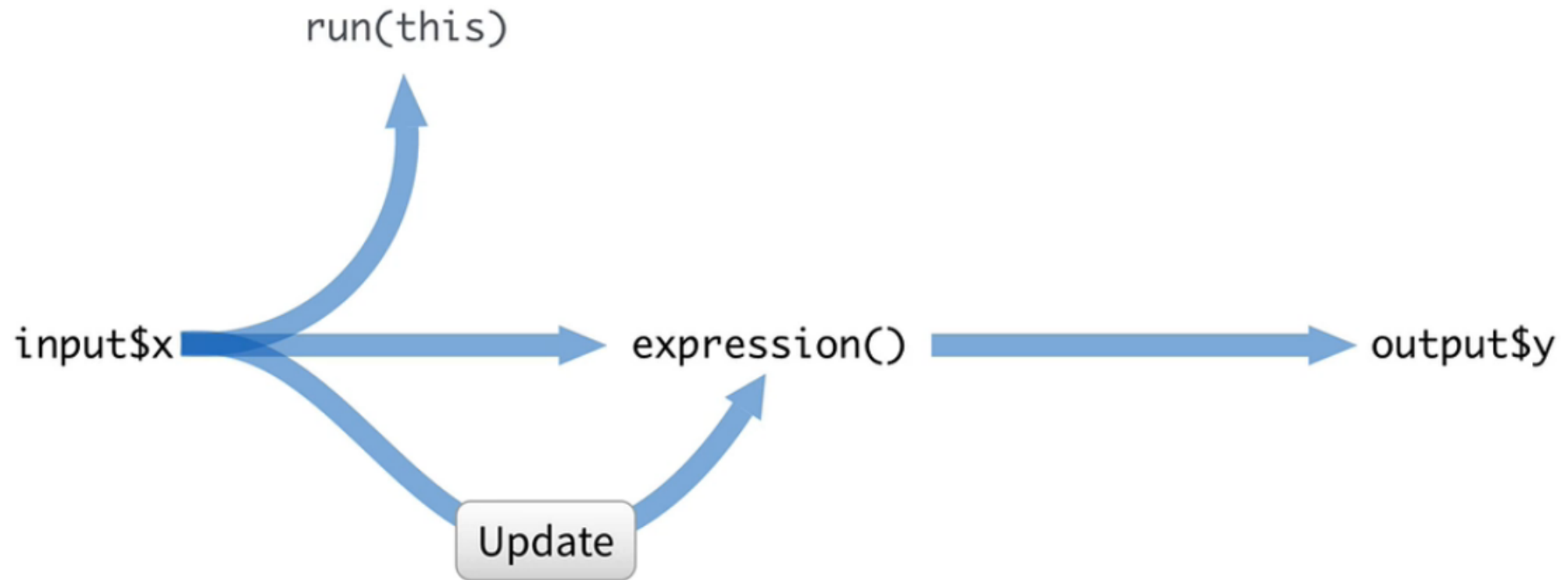


	A	B	C	D
1				
2				=A3*B3
3	15	10		150
4				
5				

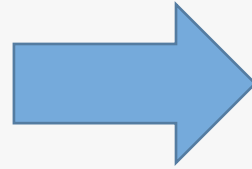
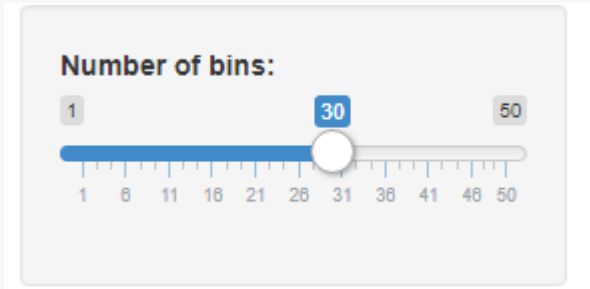
La reactividad en shiny es similar:

- Actualización o cambios en los input alterará o recalculará los nuevos outputs

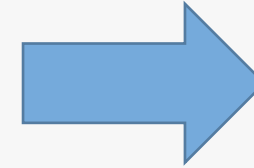
# REACTIVIDAD



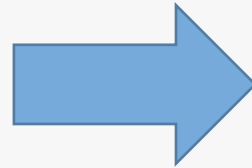
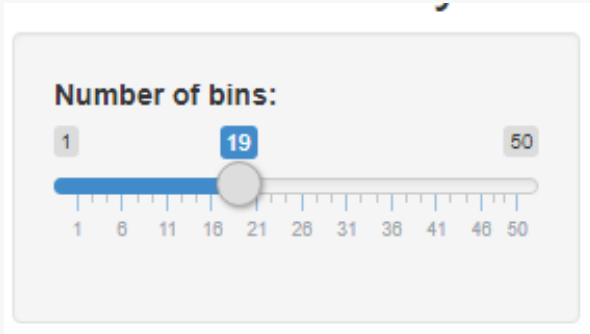
# REACTIVE VALUES



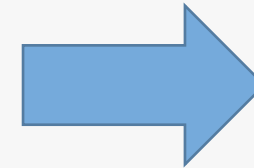
**input\$bins = 30**



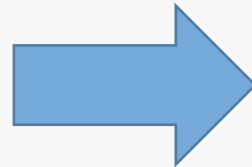
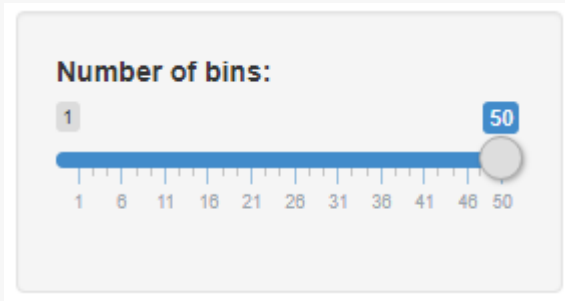
**renderPlot()**



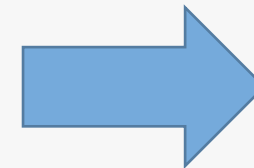
**input\$bins = 19**



**renderPlot()**



**input\$bins = 50**



**renderPlot()**

# REACTIVE VALUES

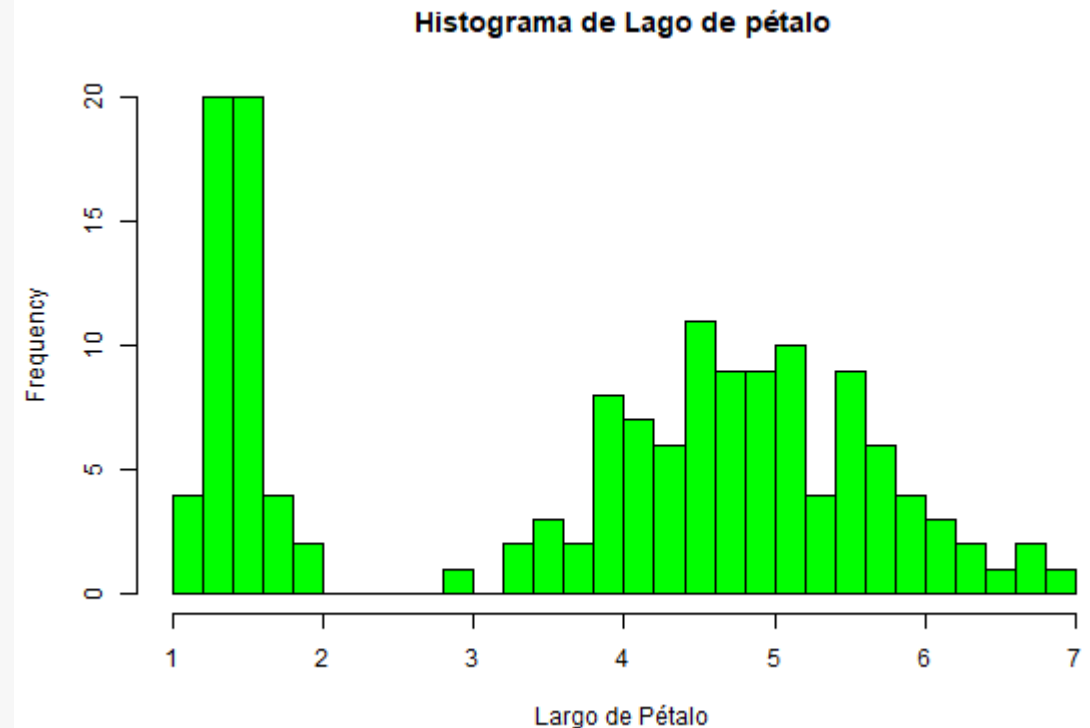
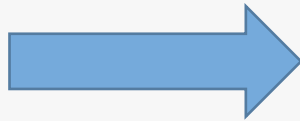
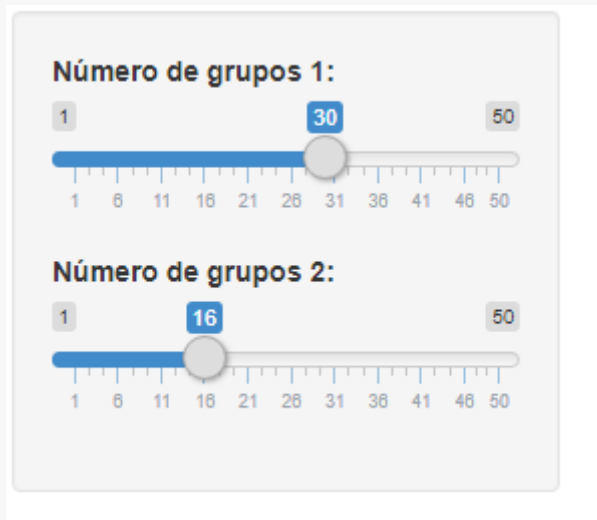
## REGLA GENERAL

- Los valores reactivos del input solo funcionarán con funciones reactivas
- No es possible llamar un valor reactive fuera de las funciones reactivas

```
47: server [C:\ResearchCode\Ejemplo2/app.R#62]  
Error in .getReactiveEnvironment()$currentContext() :  
  operation not allowed without an active reactive context. (You tried to do something  
  that can only be done from inside a reactive expression or observer.)
```

# REACTIVIDAD EN DOS PASOS

- Los **valores reactivos notifican** a las funciones de los cambios
- Los objetos creados con funciones reactivas **responden a los cambios**



# **FUNCIONES REACTIVAS**

---

**1. Usan un trozo de código para construir y reconstruir un objeto**

**2. El objeto responderá y cambiará a los cambios de un conjunto de valores reactivos**



# RENDER

Función de salida	Crea
<code>renderDataTable()</code>	Tabla interactiva (de fuentes diversas)
<code>renderImage()</code>	Una imagen (enlace a archivo)
<code>renderPlot()</code>	Un gráfico
<code>renderPrint()</code>	Un bloque de código que imprime salida
<code>renderTable</code>	Una tabla común
<code>renderText</code>	Texto
<code>renderUI</code>	Un elemento Shiny UI (Html)

Las funciones de **render\*()** nos permiten crear el tipo de salida que queremos de nuestro programa

# RENDER

---

## render\*()

Crea una salida reactiva para mostrar en la interfaz de usuario

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

**Cuando a la función se le notifica que es inválida, el objeto se  
ejecutará nuevamente**

# MODULARIZACIÓN



# REACTIVE

---

## reactive()

Retorna el resultado como un valor reactivo

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

Objeto responderá a cada valor reactivo en el código

Código utilizado para construir objeto.  
Puede ser multilinea.

# USANDO EL ELEMENTO REACTIVO

---

`data()`

- Se debe llamar al elemento siempre entre paréntesis. El elemento es una función.
- Reaccionará de forma automática a los cambios
- Guarda los valores en memoria (caché)

# ISOLATE

---

## isolate()

Retorna el resultado como un valor NO reactivo

```
isolate({rnorm(input$numero)})
```

Objeto NO responderá a ningún  
valor reactivo en el código

Código utilizado para construir objeto.  
Puede ser multilínea.

# OBSERVE EVENT

## observeEvent()

Activa código para que el servidor lo ejecute

**observeEvent**(**input\$clicks**, {**print(input\$clicks)**})

Valor o valores reactivos a los  
que responder

Código utilizado para  
construir objeto cuando el  
observador fue invalidado

Nota: Funciona como si  
fuera isolate()

El observador se invalida  
SOLO cuando el valor cambia.

# ACTION BUTTON

```
actionButton(inputId="button", label="Botón de acción")
```

Función Input

Nombre único identificador en código

Etiqueta/Nombre visible

Botón de acción





# OBSERVE()

---

## observe ()

Activa código para que el servidor lo ejecute

Usa la misma sintaxis que `render*()`, `reactive()` e `isolate()`

**observe({print(input\$clicks)})**

Responde a CADA valor reactive  
en el código

Código que se ejecutará cada  
vez que se invalide el  
observador

# EVENTREACTIVE()

---

## eventReactive()

Expresión reactiva que responde solo a valores específicos

```
data <- eventReactive(input$go, {rnorm(input$num)})
```

El valor sobre el cuál se reaccionará

Código que se ejecutará cada vez que se invalide el observador

Cómo si tuviera isolate()

# REACTIVE VALUE

---

## reactiveValues()

Crea una lista de valores reactivos para manipular programáticamente

```
rv <- reactiveValues(data = norm(100) )
```



Elementos de la lista (opcional)

# OBSERVERS Y REACTIVES

---

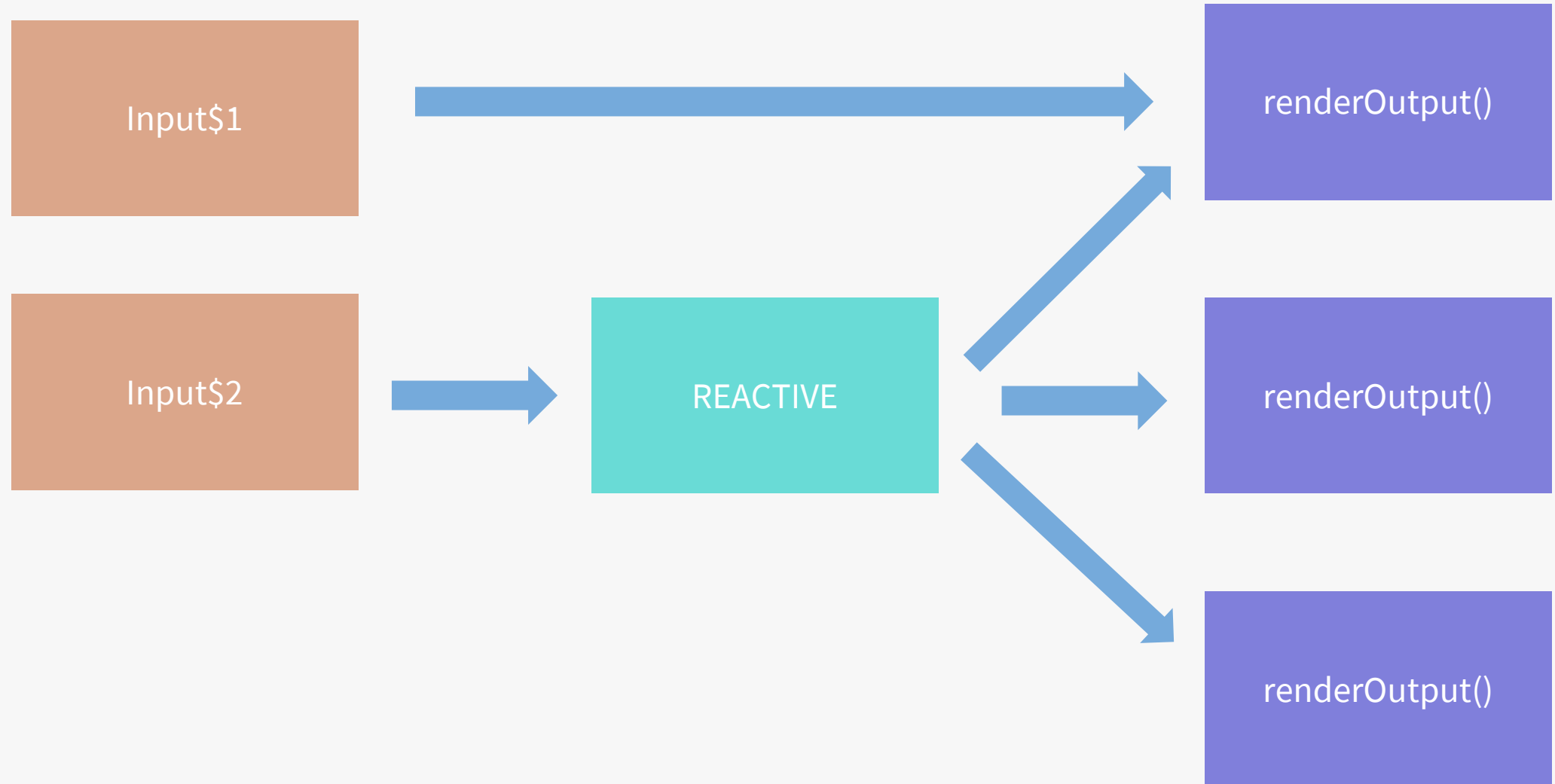
Ambos ejecutan expresiones para que se ejecuten

Observadores responden a eventos de salida

Expresiones reactivas devuelven valores, observadores no

# TIP: DATOS DE APLICACIÓN

---



# TIP: REDUCE LA REPETICIÓN DE CÓDIGO

---

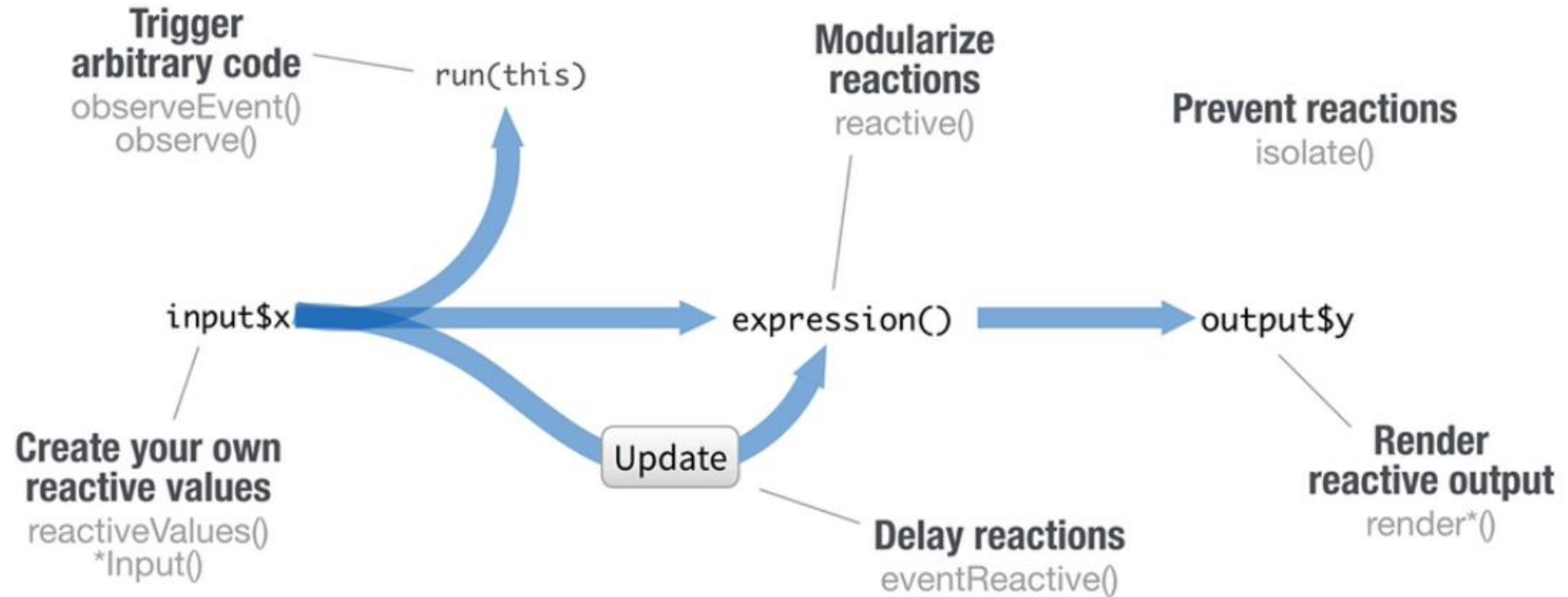
Intenta que tu código se ejecute la menor cantidad de veces que sea necesario para mejorar la velocidad

Código fuera de la función server se ejecutará una sola vez por sesión de R

Código dentro de la función servidor se ejecutará una vez por Usuario final (conexión)

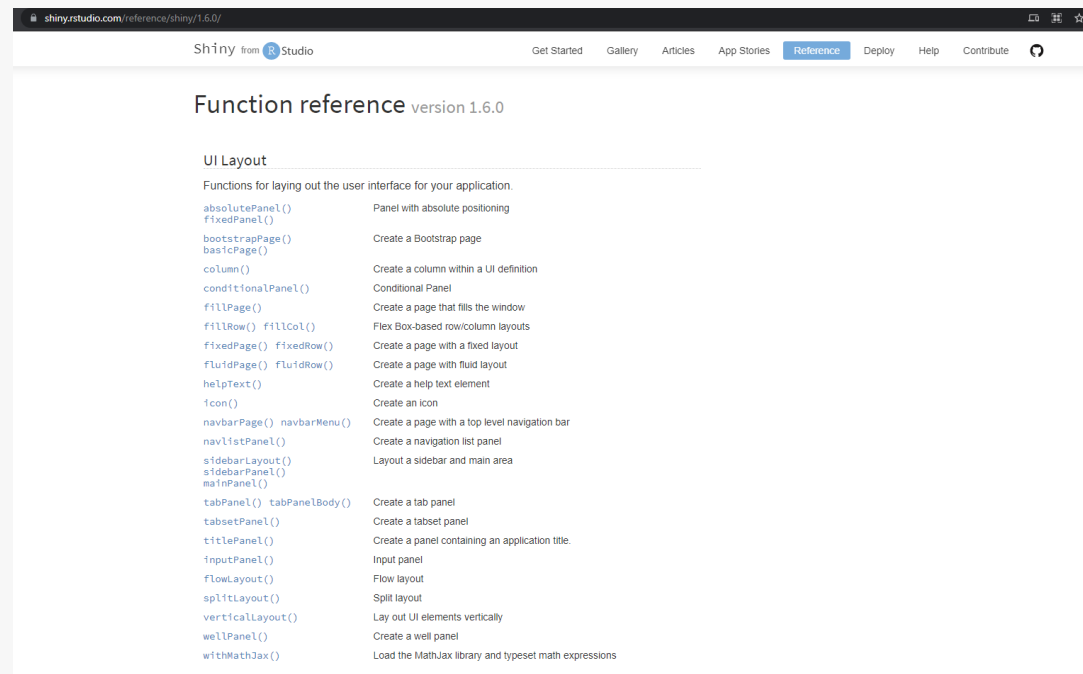
Código dentro de una función reactiva se ejecutará una vez por reacción (MUUUUCHAS VECES)

# RESUMIENDO



# ESTRATEGIAS PARA EL DESARROLLO

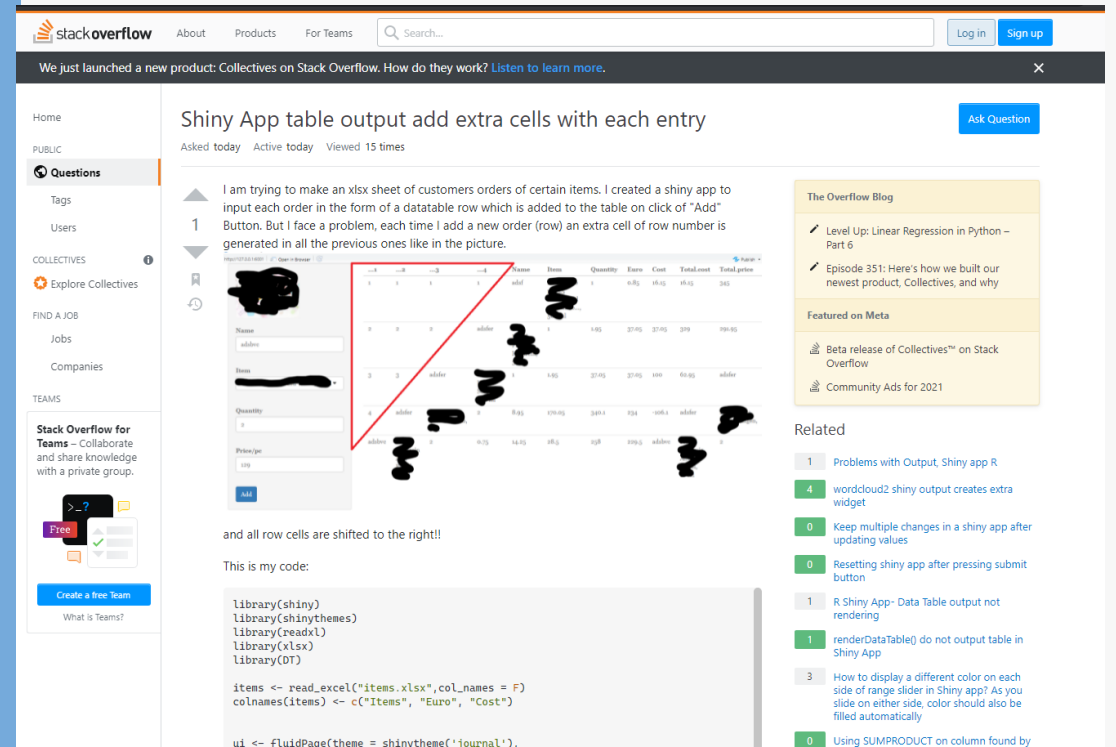
## Documentación



The screenshot shows the 'Function reference' page for Shiny version 1.6.0. The page is titled 'Function reference version 1.6.0' and lists various UI layout functions. The functions are organized into a table with two columns: the function name and a brief description of what it does.

Function	Description
<code>absolutePanel()</code>	Panel with absolute positioning
<code>fixedPanel()</code>	Panel with absolute positioning
<code>bootstrapPage()</code>	Create a Bootstrap page
<code>basicPage()</code>	Create a Bootstrap page
<code>column()</code>	Create a column within a UI definition
<code>conditionalPanel()</code>	Conditional Panel
<code>fillPage()</code>	Create a page that fills the window
<code>fillRow()</code> <code>fillCol()</code>	Flex Box-based row/column layouts
<code>fixedPage()</code> <code>fixedRow()</code>	Create a page with a fixed layout
<code>fluidPage()</code> <code>fluidRow()</code>	Create a page with fluid layout
<code>helpText()</code>	Create a help text element
<code>icon()</code>	Create an icon
<code>navBarPage()</code> <code>navBarMenu()</code>	Create a page with a top level navigation bar
<code>navListPanel()</code>	Create a navigation list panel
<code>sidebarLayout()</code> <code>sidebarPanel()</code> <code>mainPanel()</code>	Layout a sidebar and main area
<code>tabPanel()</code> <code>tabpanelBody()</code>	Create a tab panel
<code>tabsetPanel()</code>	Create a tabset panel
<code>titlePanel()</code>	Create a panel containing an application title
<code>inputPanel()</code>	Input panel
<code>flowLayout()</code>	Flow layout
<code>splitLayout()</code>	Split layout
<code>verticalLayout()</code>	Lay out UI elements vertically
<code>wellPanel()</code>	Create a well panel
<code>withMathJax()</code>	Load the MathJax library and typeset math expressions

## Google / StackOverflow



The screenshot shows a Stack Overflow question titled 'Shiny App table output add extra cells with each entry'. The question is asked by a user named '1' and has 15 views. The user is trying to create an Excel sheet of customer orders and is having trouble adding a new row to the table output. The user has provided a screenshot of their Shiny app interface, which shows a table with columns: Name, Items, Quantity, Extra, Cost, Total cost, and Total price. The table has 4 rows of data. The user is asking for help to add a new row to the table output.

The user's code is as follows:

```
library(shiny)
library(shinythemes)
library(readxl)
library(xlsx)
library(DT)

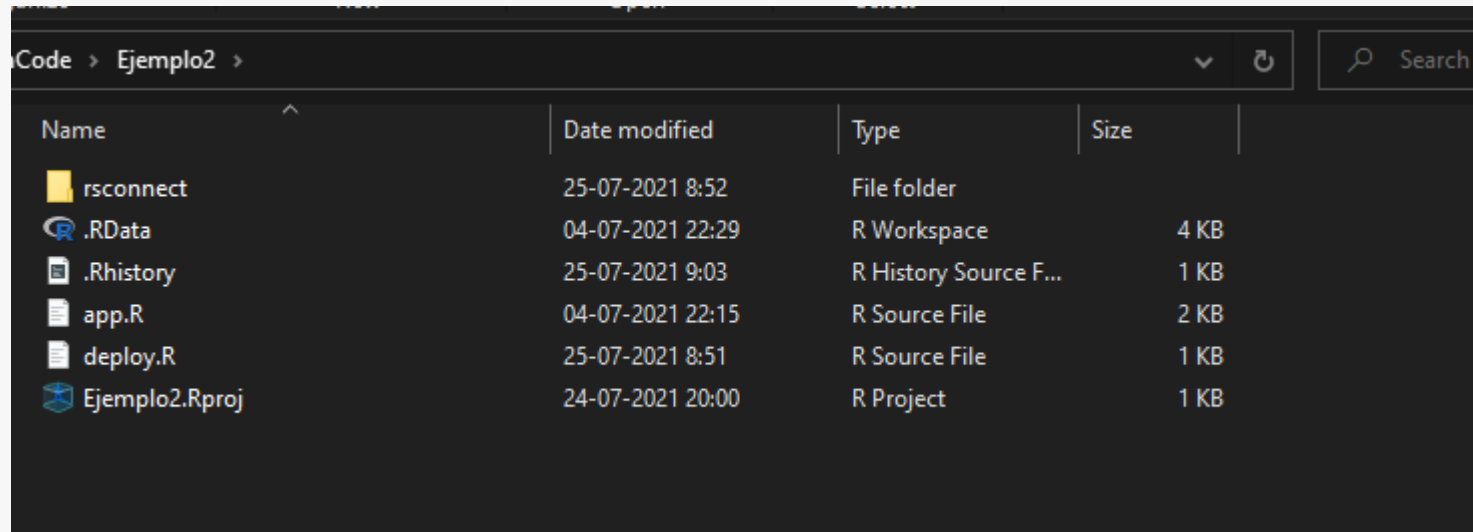
items <- read_excel("items.xlsx", col_names = F)
colnames(items) <- c("Items", "Euro", "Cost")

ui <- fluidPage(theme = shinytheme('journal'),
```



# GUARDANDO EL PROYECTO

- Para guardar la aplicación debes mantener toda los archivos, código e imágenes dentro de tu carpeta.
- El archivo principal debe llamarse app.R



The screenshot shows the 'Files' pane in RStudio for a project named 'Ejemplo2'. The pane displays a list of files and folders with their names, modification dates, types, and sizes.

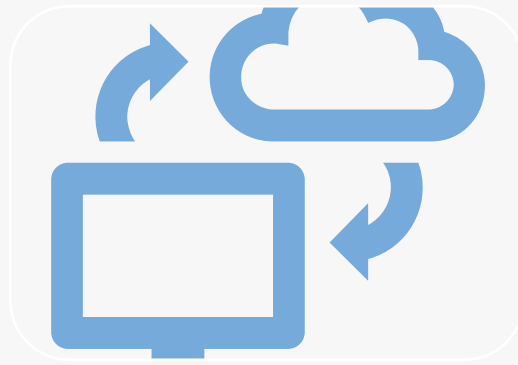
Name	Date modified	Type	Size
rsconnect	25-07-2021 8:52	File folder	
.RData	04-07-2021 22:29	R Workspace	4 KB
.Rhistory	25-07-2021 9:03	R History Source F...	1 KB
app.R	04-07-2021 22:15	R Source File	2 KB
deploy.R	25-07-2021 8:51	R Source File	1 KB
Ejemplo2.Rproj	24-07-2021 20:00	R Project	1 KB

# PUBLICANDO Y COMPARTIENDO

---



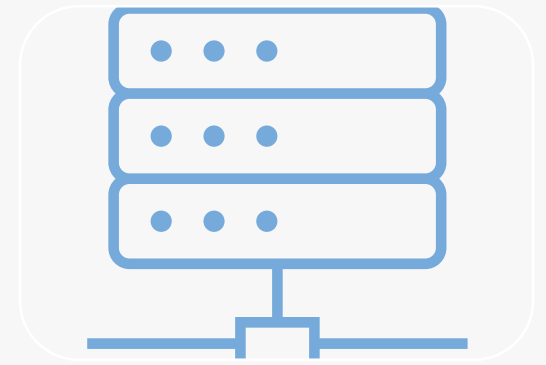
Archivo o carpeta  
comprimida para el uso  
en otro computador



Dejar todos los archivos  
en carpeta compartida  
en la nube



Publicar en  
[shinyapps.io](https://shinyapps.io)



Publicar en un servidor  
propio

# SHINYAPPS.IO

shinyapps.io

HelpAccount: felipepgrafFelipe Andres Peña Graf

Dashboard

Applications

Account

GETTING STARTED

Hi! You must be new here...

Thanks for trying out shinyapps.io! You'll need to install the `rsconnect` R package to get started. The `rsconnect` package enables you to deploy and manage your Shiny applications directly from your R console. To get started, fire up your favorite IDE, and follow the directions below.

STEP 1 – INSTALL RSCONNECT

The `rsconnect` package can be installed directly from CRAN. To make sure you have the latest version run following code in your R console:

```
install.packages('rsconnect')
```

STEP 2 – AUTHORIZE ACCOUNT

The `rsconnect` package must be authorized to your account using a token and secret. To do this, click the copy button below and we'll copy the whole command you need to your clipboard. Just paste it into your console to authorize your account. Once you've entered the command successfully in R, that computer is now authorized to deploy applications to your shinyapps.io account.

```
rsconnect::setAccountInfo(name='felipepgraf',
                           token='17FAE49394A953C0A01B177E30E03908',
                           secret='<SECRET>')
```

Show secret

Copy to clipboard

In the future, you can manage your tokens from the [Tokens](#) page the settings menu.

STEP 3 – DEPLOY

Once the `rsconnect` package has been configured, you're ready to deploy your first application. If you haven't written any applications yet, you can also checkout the [Getting Started Guide](#) for instructions on how to deploy our demo application. Run the following code in your R console.

```
library(rsconnect)
rsconnect::deployApp('path/to/your/app')
```

# EJERCICIO PRÁCTICO 2

- Cree una aplicación sencilla utilizando el dataset iris que despliegue dos gráficos a partir de una muestra del dataset
- Debe crear cuatro campos de input: 2 slider y uno de título y un botón para actualizar el muestreo
- El dataset debe remuestrarse solo al presionar el botón
- Genere un histograma y un scatterplot como se ve en la imagen
- Sube tu aplicación a shinyapps.io

Ejercicio Práctico 2 con Iris

Número de grupos 1:

1 10 20

1 3 5 7 9 11 13 15 17 19 20

Título de Gráfico

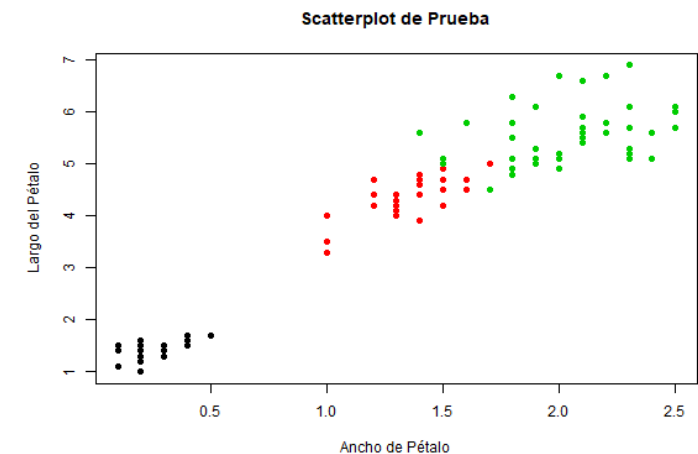
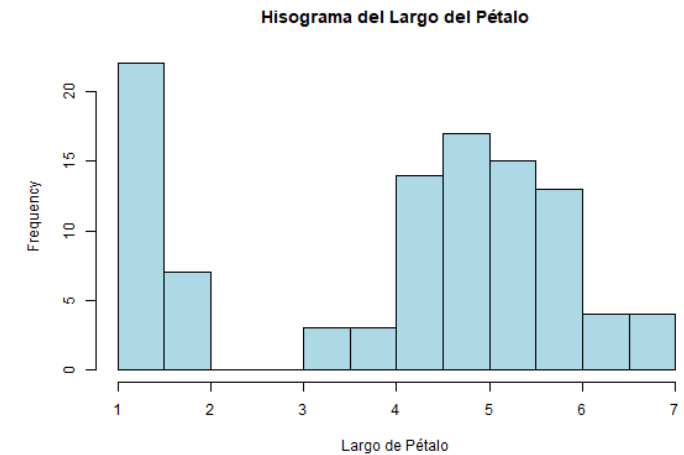
Hisograma del Largo del Pétalo

Número de elementos en dataset :

10 102 150

10 24 38 52 66 80 94 108 122 136 150

Calcular/Actualizar



# **TALLER SHINY**

**Diplomado Data Science**  
**Felipe Peña Graf**

