

TEMA 3

REACTIVIDAD BÁSICA

# Reactividad

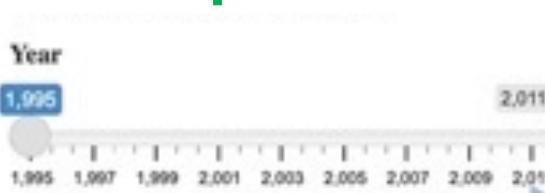
## Google Charts demo



# Como funciona la reactividad en el contexto de Shiny

Los valores reactivos trabajan  
junto con las **funciones reactivas\***

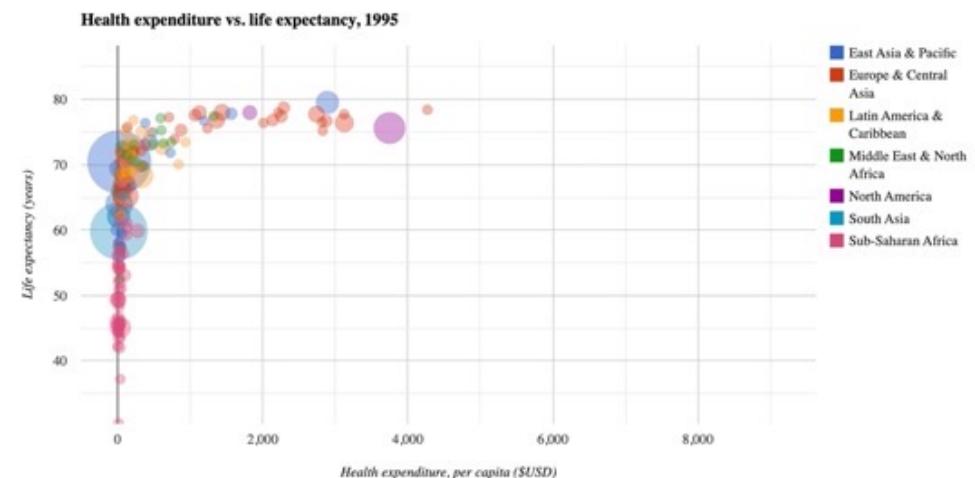
input\$x



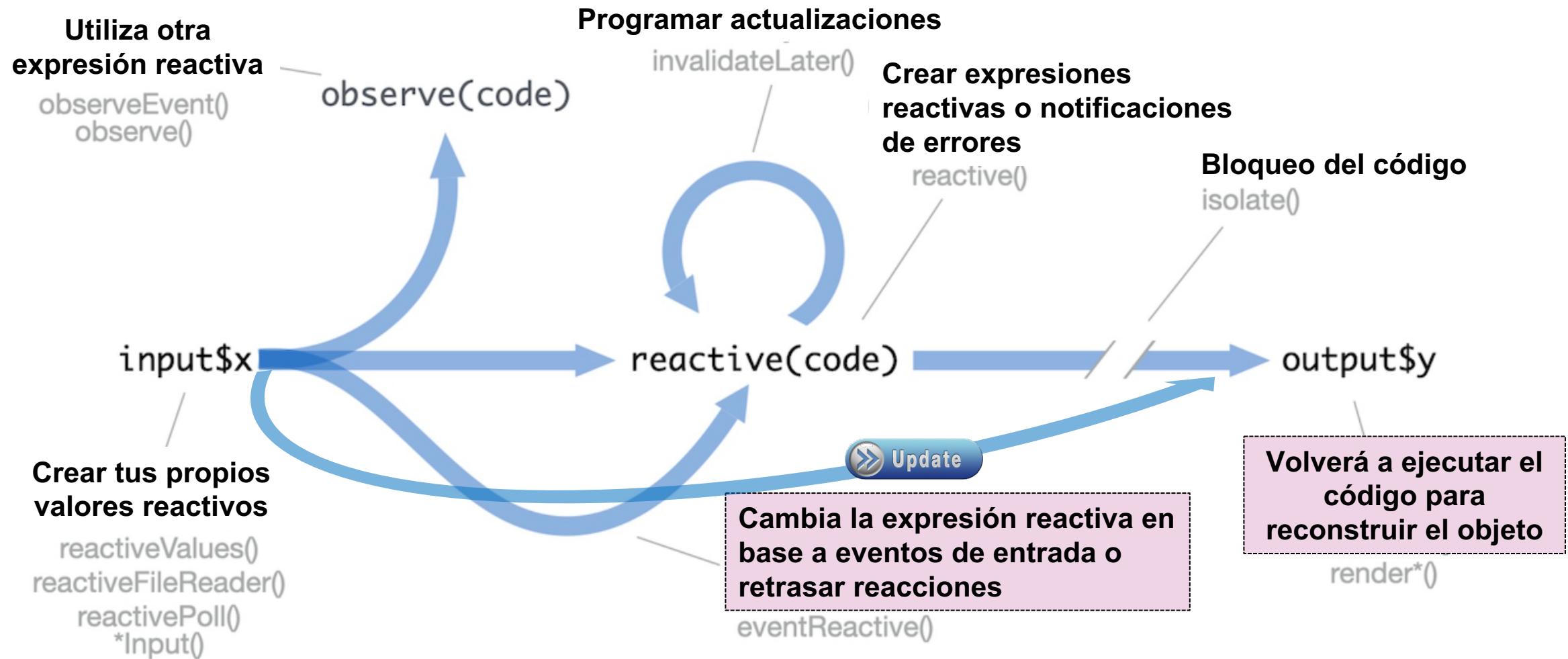
reactive ( )\*



output\$y



# Funciones reactivas\*



## Función render ( )\*

| Función             | Objeto                        |
|---------------------|-------------------------------|
| renderDataTable ( ) | Una tabla interactiva         |
| renderImage ( )     | Imagen                        |
| renderPlot ( )      | Un gráfico                    |
| renderPrint ( )     | Un bloque de código de salida |
| renderTable ( )     | Una tabla                     |
| renderText ( )      | Una cadena de texto           |
| renderUI ( )        | Un elemento de Shiny UI       |

# Construir objetos de salida para mostrar en UI

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
  label = "Choose a number",
  value = 25, min = 1, max = 100),
  plotOutput("hist")
)
server <- function(input, output) {
  output$hist <- renderPlot ({
    hist(rnorm(input$num))
  })
}
shinyApp(ui = ui, server = server)
```

Objeto que responderá a cada valor reactivo en el código

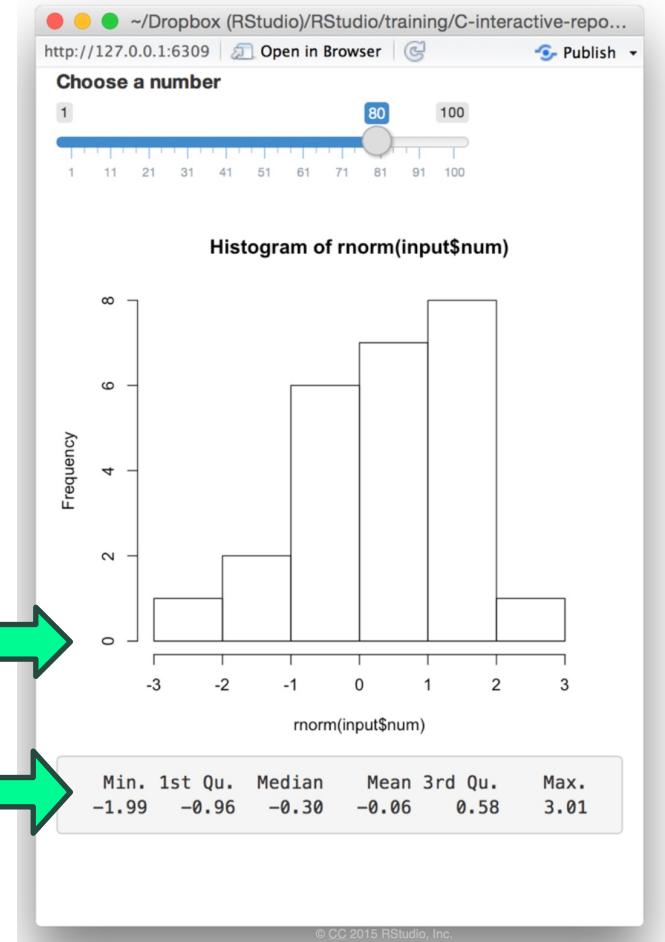
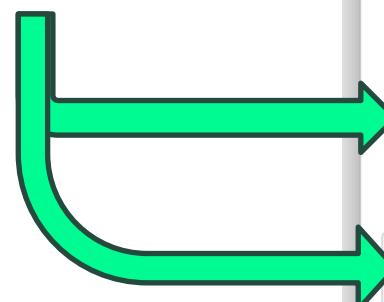
`renderPlot ({ hist(rnom(input$num)) })`

Se ejecuta todo el código para construir (y reconstruir) el objeto

# Dos outputs... diferente representación de los mismos datos

```
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)
```

Estos dos outputs representan dos variantes de los mismos datos



# Construcción de objetos reactivos

```
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)
```

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

Utilizamos **data ( )** como función reactiva  
(funciona como el cache)

```
output$hist <- renderPlot ({
  hist(data())
})
```

# Botones de acción



Función de input

input ID  
(para uso interno)

Etiqueta para mostrar

```
actionButton(inputId = "go", label = "Click Me")
```

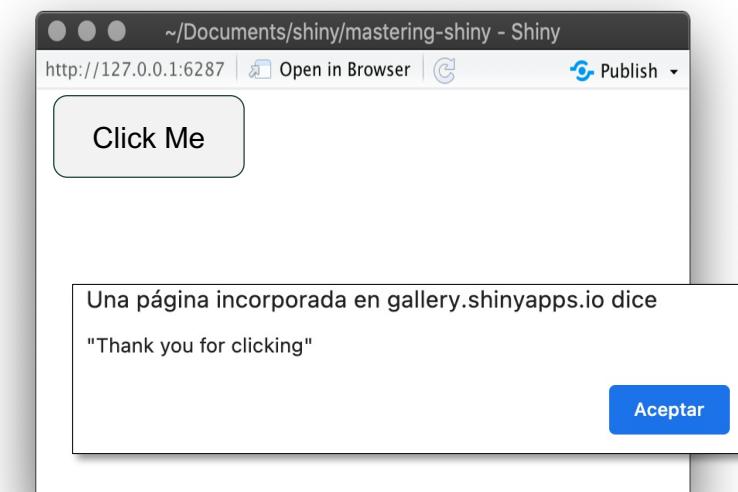
## Introducimos el botón en UI y definimos la acción en server

```
library(shiny)

ui <- fluidPage(
  actionButton(inputId = "clicks",
  label = "Click me")
)

server <- function(input, output) {
  observeEvent(input$clicks, {
    session$sendCustomMessage(type ='testmessage',
    message = 'Thank you for clicking')
  })
}

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

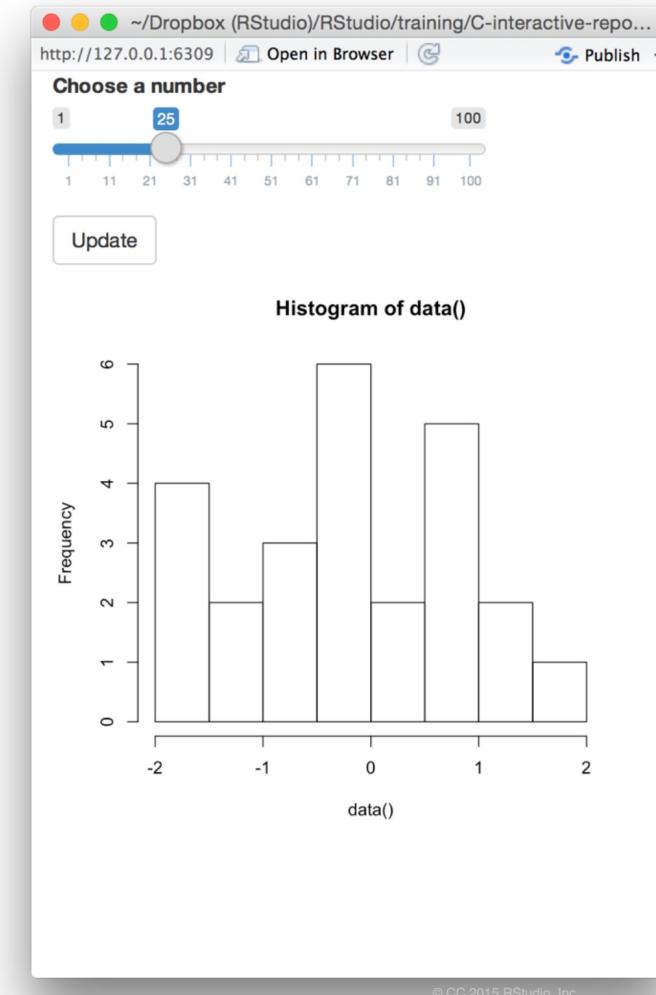


# Los botones de acción nos ayudan a controlar la frecuencia de reactividad

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
  label = "Choose a number",
  value = 25, min = 1, max = 100),
  actionButton(inputId = "go", label = "Update"),
  plotOutput("hist")
)
server <- function(input, output) {
  data <- eventReactive(input$go, {
    rnorm(input$num)
  })
  output$hist <- renderPlot ({
    hist(data())
  })
}
shinyApp(ui = ui, server = server)
```

Paso 1



Paso 3

Paso 2