Chap 3 Shiny

Elvira Lucas

2023-03-17

```
knitr::opts_chunk$set(
  eval = FALSE
)
```

Introduction

Shiny ce n'est que du html

```
require(shiny)
runExample("01_hello")
```

Interactif, quand on fait un code dans R : x <- 9 y <- x+1

x <- 10 y

y n'est pas actualisé, il n'y a pas d'interactivité. Dans shiny, on peut l'ajouter, mais c'est important de bien le faire pour ne pas avoir de problèmes.

Interface de Shiny

Besoin de deux scripts, un pour l'utilisateur (ui) et un pour le serveur. Le serveur c'est ce que shiny va calculer et ui c'est la où je spécifie comment je veux que la page s'organise. Entre chaque élèment de mon code, chaque ligne, il faut une ",". Je rajoute un "input" ou "output" en fin de ma fonction pour spécifier ce qui sort.

```
# Exemple de partie "ui"
ui <- fluidPage(

    # Application title
    titlePanel("Old Faithful Geyser Data"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                        "Number of bins:",
                        min = 1,
                        max = 50,
                        value = 30) # "bins" est le nom du slider
        ),

        # Show a plot of the generated distribution
        mainPanel(
```

```
            plotOutput("distPlot") # nom d'une figure
        )
    )
)
```

Agit comme une boucle, donc ne sert à rien de rajouter une valeur par défaut dedans, la mettre avant la function (un peu comme on spécifie la valeur d'une variable avant de l'utiliser dans une boucle).

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

    output$distPlot <- renderPlot({
        # generate bins based on input$bins from ui.R
        x    <- faithful[, 2]
        bins <- seq(min(x), max(x), length.out = input$bins + 1)

        # draw the histogram with the specified number of bins
        hist(x, breaks = bins, col = 'darkgray', border = 'white',
             xlab = 'Waiting time to next eruption (in mins)',
             main = 'Histogram of waiting times')
    })
}
```

Functions

Layouts

```
## ------------------------------------------------
## User interfaces
## ------------------------------------------------

sliderInput <- sliderInput(
  inputId = "bins",
  label = "Number of bins:",
  min = 1,
  max = 50,
  value = 30
)

plotOutput <- plotOutput(outputId = "distPlot")

# En sortie, je veux un distplot qui va être utilisé dans "Server"

# layout 1
ui1 <- fluidPage(
  sliderInput,
  plotOutput
)

# layout 2
ui2 <- fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
```

```
      sliderInput
    ),
    mainPanel(
      plotOutput
    )
  )
)

# Layout 2 je rajoute un titre et comment le graphique et l'élément interactif sont disposés

## ------------------------------------------------
## Server
## ------------------------------------------------

x <- faithful$waiting

server <- function(input, output) {
  output$distPlot <- renderPlot({
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins)
  })
}


## ------------------------------------------------
##  Build the app with layout 1 or 2
## ------------------------------------------------

shinyApp(ui = ui1, server = server)

shinyApp(ui = ui2, server = server)
```

Shiny un peu inutile mais juste pour montrer qqch

```
#------  UI 1  ------
ui1 <- fluidPage(
  titlePanel("Fluid page"), # title
  # contents ----
  paste(rep(LETTERS, 20), collapse = " "),
  paste(1:150, collapse = " "),
  paste(rep(letters, 40), collapse = " "),
  paste(rep(LETTERS, 40), collapse = " "),
  paste(1:300, collapse = " "),
  paste(rep(letters, 40), collapse = " ")
)

# Ici illisible tout se met n'importe comment

#------  UI 2  ------
ui2 <- fluidPage(
  titlePanel("Rows"),
  # row 1 ----
```

```
  fluidRow(
    # contents of row 1 ----
    paste(rep(LETTERS, 20), collapse = " "),
    paste(1:150, collapse = " "),
    paste(rep(letters, 40), collapse = " "),
    paste(rep(LETTERS, 40), collapse = " ")
  ),
  br(), # line break
  # row 2 ----
  fluidRow(
    # contents of row 2 ----
    paste(1:300, collapse = " "),
    paste(rep(letters, 40), collapse = " ")
  )
)

# Ici on rajoute un break entre 2 paragraphes

#------  UI 3  ------
ui3 <- fluidPage(
  titlePanel("Rows & Columns"),
  # row 1 ----
  fluidRow(
    # column 1 ----
    column(
      4, # length of the column
      # contents of row 1, column 1 ----
      paste(rep(LETTERS, 20), collapse = " "),
      br(),
      br(),
      paste(1:150, collapse = " ")
    ),
    # column 2 ----
    column(
      4,
      # contents of row 1, column 2 ----
      paste(rep(letters, 40), collapse = " ")
    ),
    # column 3 ----
    column(
      4,
      # contents of row 1, column 3 ----
      paste(rep(LETTERS, 40), collapse = " ")
    )
  ),
  br(),
  # row 2 ----
  fluidRow(
    # column 1 ----
    column(
      4,
      paste(1:300, collapse = " ")
    ),
```

```
    # column 2 ----
    column(
      8,
      paste(rep(letters, 40), collapse = " ")
    )
  )
)

# Je définis des blocs dans ma page avec le fluidrow, mais toujours pas incroyable

#------  UI 4  ------
ui4 <- fluidPage(
  titlePanel("Panel"),
  fluidRow(
    column(
      4, # la largeur de la colonne
      # panel
      wellPanel(
        # contents of the panel
        paste(rep(LETTERS, 10), collapse = " "),
        br(),
        br(),
        paste(1:150, collapse = " ")
      ),
      paste(1:100, collapse = " ")
    ),
    column(
      4,
      paste(rep(letters, 40), collapse = " ")
    ),
    column(
      4,
      paste(rep(LETTERS, 40), collapse = " ")
    )
  )
)

# Ici je rajoute des colonnes en plus de mes blocs en lignes où je définis leur largeur

#------  UI 5  ------
ui5 <- fluidPage(
  titlePanel("Panel + Tabs"),
  fluidRow(
    column(
      4,
      # panel
      wellPanel(
        paste(rep(LETTERS, 20), collapse = " "),
        br(),
        br(),
        paste(1:150, collapse = " ")
      ) #wellPanel permet de griser cette partie de la colonne
    ),
```

```
    column(
      8,
      # tabs layout
      tabsetPanel(
        # tab 1
        tabPanel(
          title = "Tab1",
          br(),
          paste(rep(letters, 50), collapse = " "),
          br(),
          br(),
          paste(rep(LETTERS, 30), collapse = " ")
        ),
        # tab 2
        tabPanel(
          title = "Tab2",
          br(),
          paste(1:300, collapse = " "),
          br(),
          br(),
          paste(rep(LETTERS, 50), collapse = " ")
        )
      )
    )
  )
)

# les tabPanel permette de réaliser des onglets

#------  UI 6  ------
ui6 <- fluidPage(
  titlePanel("Sidebar layout"),
  # sidebar layout
  sidebarLayout(
    # side panel
    sidebarPanel(
      width = 3, # width of the panel (default 4)
      paste(rep(LETTERS, 10), collapse = " "),
      br(),
      br(),
      paste(1:150, collapse = " ")
    ),
    # main panel
    mainPanel(
      width = 9,
      paste(rep(letters, 50), collapse = " "),
      br(),
      br(),
      paste(rep(LETTERS, 30), collapse = " ")
    )
  )
)
```

```
# Je spécifie un main panel

## ------------------------------------------------
## Server
## ------------------------------------------------

server <- function(input, output) { }

## ------------------------------------------------
## Create Shiny app
## ------------------------------------------------

# replace ui by ui1, ui2,..., ui6.
# shinyApp(ui = ui, server = server)
```

!! Un même nom ne peut pas être utilisé 2X pour nommer une fonction

Aller voir sur le site Shiny la gallerie des widgets pour voir les codes disponibles. Quand on spécifie des valeurs, utilisez des valeurs connues de R ("red", "green"...) pour ne pas devoir redéfinir ces mêmes valeurs dans la partie serveur plus tard.

On peut rajouter de l'interactivité dans les graphiques et afficher du texte suivant où on se trouve dessus (p.ex afficher la valeur des points dans un scatterplot).

textOutput -> pour afficher du texte où des calculs (mean, sd...) dans le slider verbaTimoutput -> inscrire un résultat de code dans le slider.

Server Script

```
x <- faithful$waiting

ui <- fluidPage(
  sliderInput(inputId = "bins",
              label = "Nmbre de classes",
              min=1, max = length(x),
              value = 10),
  plotOutput(outputId = "distPlot"))

server <- function(input, output) {
  output$distPlot <- renderPlot({
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins)
  })
}
shinyApp(ui,server)
```

Render est utilisé pour afficher quelque chose d'interactif, exemple d'erreur commune :

```
ui <- fluidPage(
  sliderInput(
    inputId = "bins",
    label = "Number of bins:",
    min = 1,
    max = 50,
    value = 30
```

```
  )
)

server1 <- function(input, output) {
  print(input$bins)  # ERROR
  renderPlot({hist(x)})
}

server2 <- function(input, output) {
  output$distPlot <- hist(x) # ERROR
}

shinyApp(ui, server1)
#shinyApp(ui, server2)
```

Ici rien ne marche car on utilise des fonctions de bases non interactives, shiny ne peut rien faire avec. De plus, on ne définit l'histogramme que par x (sans breaks=...), celui-ci ne va pas changer suivant la valeur de bins. Comme le distPlot n'est pas intégré dans le sliderInput mais à côté, il n'est pas recalculé à chaque valeur différente de bins.

En effet :

```
x <- faithful$waiting
bins <- 20
hist(x, breaks = bins)
```

Créer du shiny avec Rmarkdown

Il est aussi possible de réaliser des interfaces Shiny avec du Rmarkdown. Mais il est important de savoir que même si le doc HTML ressemble à un html normal, le doc que l'on obtient est un fichier temportaire qui subsiste dans le server de Shiny en train de tourner, mais dès que celui-ci s'arrête notre html disparait.

Exemple

```
sliderInput(
  inputId = "bins",
  label = "Number of bins:",
  min = 1,
  max = 50,
  value = 10
)
```

```
renderPlot({
  bins <- seq(min(x), max(x), length.out = input$bins + 1)
  hist(x, breaks = bins)
})
```