# Explications

Lafortune Pierre

# Contents

# 1 Shiny

On peut construire facilement une application R avec Shiny.

## 1.1 C'est quoi Shiny

Shiny est une API(Application Programming Interface) qui permet de créer des sites web sans avoir une connaissance préalable en language du web.

## 1.2 Comment marche une application Shiny

Analogue au système client-serveur. Un serveur reçoit des instruction à partir d'un fichier R. Il construit la page web. L'utilisateur interagit avec cette page web. La page web intéragit avec le serveur à son tour qui puise ses instructions du fichier R.

## 1.3 Concrètement, une application shiny a deux parties

```
library(shiny)
ui <- fluidPage()

server <- function(input, output){}

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

- La partie ui(User Interface) regroupe toutes les instructions de mise en page. C'est l'interface destiné à être vue par l'utilisateur.
- La partie server regroupe toutes les instructions pour plotter les graphes.

Shiny facilite l'interaction entre les deux et met en place un système de `reactive` pour intéragir avec l'utilisateur et garder les plots à jour.

## 1.4 Comment construire une application?

La construction d'une application doit se faire autour de ces deux mots: entrée et sortie. Il faut determiner pour chaque variables, vues, plots si c'est une entrée ou une sortie de notre application.

- Les plots sont toujours(ou presque) des sorties
- Les boutons des entrées

Il y a pour cela deux groupes de fonctions fournis par shiny: *Input et *Output.
Pour chacune de ces fonctions, ils sont nommés de la sorte: [TypeOfDisplay][Output/Input](id,arg...)
Tout les objets *Output sont construite dans la partie `server`.
On se référe à un objet graphique dans laquelle on va afficher de la sorte: [output]$[id]
Example : output$hist <- # code
Utiliser la fonction render*() pour créer les outputs.
On accède aux champs d'entrée avec input. Example:

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

## 1.5 La réactivité - Reactivity

La réactivité: une action/modification sur une entrée provoque immédiatement le calcul de la sortie associé.
Variable réactive marche avec les fonctions réactive. Ne marche pas à l'extérieur.
Marche en deux étapes:

- les valeurs réactive notifie les fonctions qui les utilisent quand ils deviennent invalide.
- l'object créér par réactivité réagit.

## 1.6 Récap Server

### 1.6.1 sauvegarder output qu'on construit dans output$*

### 1.6.2 construire les output avec les fonctions render*()

### 1.6.3 accéder aux valeurs d'entrée avec input$*

## 1.7 Objet réactive - reactive expression

Un objet réactive, de la même façon qu'une variable réactive, réagit en fonction des entrées qui se trouvent dans son corps.

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

On peut appeler un objet réactive comme une fonction.

```
data()
```

Les objet réactive `cache` leur valeur. Ils retournent leur plus récente valeur.
Utilie pour modulariser le code.

## 1.8 Les observeurs

Trigger du code à éxécuter coté serveur quand une valeur réactive à été invalidé.

## 1.9 Les eventReactive

Utile pour retarder une réaction.

## 1.10 Créer nos propres variable réactive

reactiveValues() créer une liste de valeurs réactive qu'on peut utliser.

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

observeEvent(input$norm, { rv$data <- rnorm(100) })
observeEvent(input$unif, { rv$data <- runif(100) })

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

## 1.11 Combien de fois sont éxécuté les codes

Les codes qui sont avant la méthode server sont éxécutés qu'une seule fois par session R.
Les codes qui sont dans la fonction server sont éxécutés une fois par connection utilisateur.
Les codes qui sont dans les fonction réactive sont éxécutés à chaque réaction qui les concernent.

## 1.12 Ajout de contenu HTML

Utiliser tags$* pour ajouter des tags en HTML

```
tags$h1("Cecil est un titre")
tags$a(href = "www.rstudio.com", "RStudio")
```

## 1.13 Utilisation des layout pour mieux positioner les éléments dans la page html

- fluidRow() ajout une nouvelle ligne à la grille. Chaque nouvelle ligne est ajouté en dessous la précédente.
- column(width, offset) permet de decouper une ligne en colonne. Une ligne est divisé en 12 colonne par défaut.
- Utiliser des panels pour grouper des éléments graphique de façon logique. Exemple: navlistPanel, sidebarPanel, tabPanel etc.