

Comment fonctionne **Shiny** ?

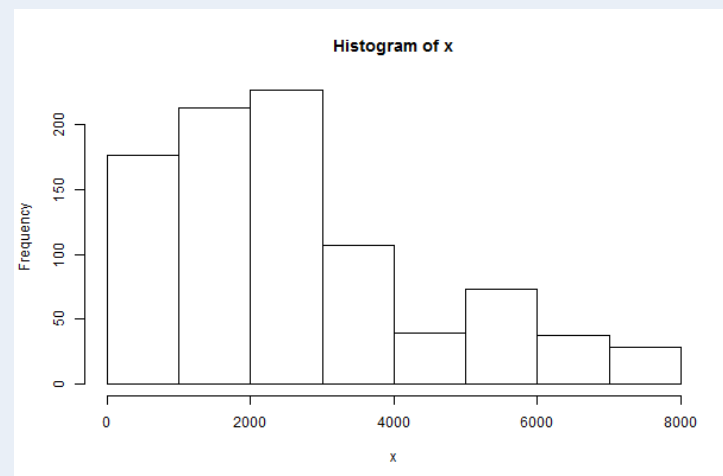
server

```
> output$myHist <-  
> renderPlot({  
>   hist(Nb_trains)  
> })
```

ui

➤ Affiche moi l'histogramme de la variable Nb_trains

```
> mainPanel(  
>   plotOutput("myHist")  
> )
```



Oups!
On a oublié quelque chose...

server

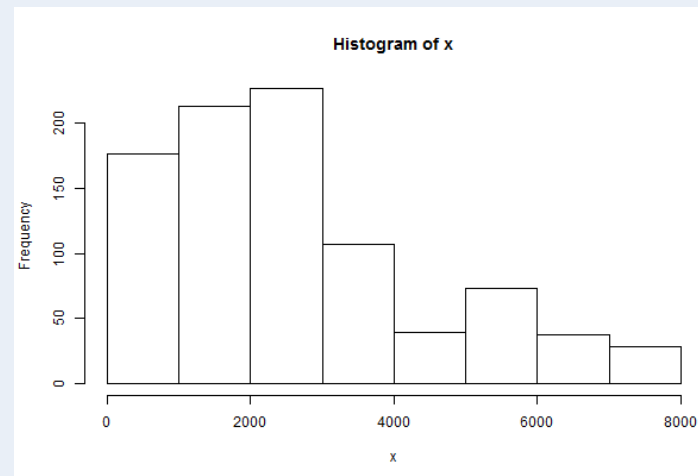
```
> myData <- read.csv(myFile)

> output$myHist <-
> renderPlot({
>     x <- myData$Nb_trains
>     hist(x)
> })
```

ui

➤ Affiche moi l'histogramme
de la variable Nb_trains

```
> mainPanel(
>     plotOutput("myHist")
> )
```



server

```
> myData <- read.csv(myFile)

> output$myTable <-
>   renderTable({
>     head(MyData)
>   })
```

ui

➤ Affiche moi les premiers
éléments de la table



```
> mainPanel(
>   tableOutput("myTable")
> )
```

Typeligne	Annee	Depart	Arrivee	Nb_Programmes	Nb_Annules	Nb_Retard	Tx_Regularite
TGV	2011	PARIS LYON	AIX EN PROVENCE TGV	1673	0	150	0.91
TGV	2011	PARIS MONTPARNASSE	ANGERS SAINT LAUD	1795	7	156	0.91
TGV	2011	PARIS MONTPARNASSE	ANGOULEME	1321	3	130	0.90
TGV	2011	PARIS LYON	ANNECY	812	1	58	0.93
TGV	2011	PARIS NORD	ARRAS	1451	0	126	0.91
TGV	2011	PARIS LYON	AVIGNON TGV	1552	1	151	0.90
TGV	2011	PARIS LYON	BELLEGARDE (AIN)	1049	0	143	0.86
TGV	2011	PARIS LYON	BESANCON FRANCHE COMTE TGV	712	0	53	0.93

server

```
> myData <- read.csv(myFile)

> output$myText <-
> renderText({
>   N_trains <- nrow(myData)
>   paste("N=",
>         N_trains, "trains")
> })
```

ui

➤ Affiche moi ...
..... du texte

```
> mainPanel(
>   textOutput("myText")
> )
```

N = 900 trains

Un peu de vocabulaire **Shiny**

server

```
› output$myOutput <-
```



ui

```
› mainPanel (  
›   )
```

```
› renderPlot ({           })
```

```
› renderTable ({         })
```

```
› renderText ({          })
```

Un graphique

```
› plotOutput ("myOutput")
```

Une Table

```
› tableOutput ("myOutput")
```

Du texte

```
› textOutput ("myOutput")
```

Oui, mais c'est pas sexy tout ça ...

server

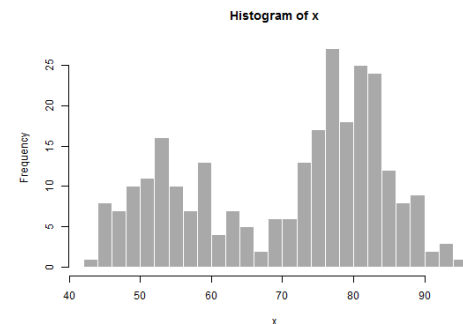
ui

- Affiche moi l'histogramme mais je choisis la largeur (bins)

bins???

```
> output$myHist <-  
> renderPlot({  
>   hist(Nb_trains,  
>         breaks = input$bins)  
> })
```

```
> sliderInput(  
>   inputId="bins",  
>   label="Nbre de boites",  
>   min = 1,  
>   max = 50,  
>   value = 30  
> )  
  
> mainPanel(  
>   plotOutput("myHist")  
> )
```



Et on peut faire quoi d'autre?

server

ui

- Affiche moi l'histogramme
mais je choisis la variable

variable???

```
> output$myHist <-  
> renderPlot({  
>   x <- myData[, input$var]  
>   hist(x )  
> })
```

```
> selectInput(  
>   inputId="var",  
>   label="Variable",  
>   choices = c("Nb_annules",  
                "Nb_retard")  
> )  
  
> mainPanel(  
>   plotOutput("myHist")  
> )
```

Variable y

Nb_Annules ▼

Un peu de vocabulaire **Shiny**

server

Keywords:

- › `output$xxx`
- › `renderYYYY`
- › `Input$zzzzz`

ui

Panels de l'interface

- › `mainPanel()`
- › `sidebarPanel()`
- › `tabsetPanel()`
- › `titlePanel()`

Les selecteurs

- › `sliderInput()`
- › `selectInput()`
- › `textInput()`

Structure générale

server

```
> library(shiny)
> ... Some R code...

> server <- function(input, output,
  session) {

  >   output$xxx <- renderPlot({
  >     some R code using input$zzz
  >     ...
  >   })

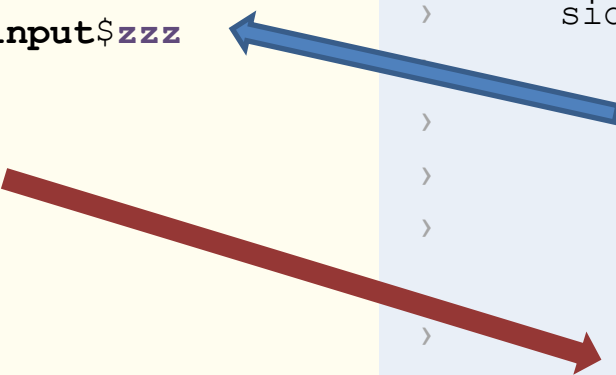
> }
```

ui

```
> library(shiny)
> ui <- fluidPage(
>   # Titre de la page
>   titlePanel("Mon titre"),

>   sidebarLayout(
>     # Définition du panneau latéral
>     sidebarPanel(
>       #Type de sélecteur
>       sliderInput("zzz",
>         ... )
>     ),

>     # Fenêtre principale
>     mainPanel(
>       plotOutput("xxx")
>     )
>   )
> }
```



Scope

server

```
> library(shiny)
> Some R code # Exécuté une seule fois, au démarrage de Shiny (disponible durant la session)
> server <- function(input, output) {
>   Some R code # Définis (et calculés) durant la session
>   output$xxx<- renderPlot({
>     # Les objets ici sont calculés à chaque fois que la fonction est appelée
>     some R code using input$zzz
>   })
> }
```

See: <http://shiny.rstudio.com/articles/scoping.html>

Peut on afficher plusieurs outputs?

- Affiche moi l'histogramme **et**
le début de la **table** (stp)!

Variable ?

```
> output$myHist <-  
> renderPlot({  
>   x <- myData[, input$var]  
>   hist(x)  
> })  
> output$myTable <-  
> renderTable({  
>   head(myData)  
> })
```

```
> selectInput(  
>   inputId="var",  
>   label="Variable",  
>   choices = c("Nb_annules" ,  
               "Nb_retard")  
> )  
  
> mainPanel(  
>   plotOutput("myHist")  
>   plotOutput("myTable")  
> )
```


- Affiche moi l'histogramme **et les statistiques** (stp)!

Variable ?

```
> output$myHist <-  
> renderPlot({  
>   x <- myData[, input$var]  
>   hist(x)  
> })  
> output$myStats <-  
> renderText({  
>   x <- myData[, input$var]  
>   summary(x)  
> })
```

```
> selectInput(  
>   inputId="var",  
>   label="Variable",  
>   choices = c("Nb_annules" ,  
               "Nb_retard")  
> )  
  
> mainPanel(  
>   plotOutput("myHist")  
>   plotOutput("myStats")  
> )
```

C'était plus long, non?

Deux éléments à fabriquer+ deux
éléments à afficher, forcément....

- Affiche moi l'histogramme et les statistiques (stp)!

Variable ?

```
> output$myHist <-  
> renderPlot({  
>   x <- myData[, input$var]  
>   hist(x)  
> })  
> output$myStats <-  
> renderText({  
>   x <- myData[, input$var]  
>   summary(x)  
> })
```

```
> selectInput(  
  inputId="var",  
  label="Variable",  
  choices = c("Nb_annules" ,  
              "Nb_retard")  
> )  
  
> mainPanel(  
  plotOutput("myHist")  
  plotOutput("myStats")  
> )
```

Rappel Scope

server

```
> library(shiny)
> Some R code # Exécuté une seule fois, au démarrage de Shiny (disponible durant la session)
> server <- function(input, output) {
>     Some R code # Définis (et calculés) durant la session
>     output$xxx<- renderPlot({
>         # Les objets ici sont calculés à chaque fois que la fonction est appelée
>         some R code using input$zzz
>     })
> }
```

See: <http://shiny.rstudio.com/articles/scoping.html>

- Affiche moi l'histogramme **et les statistiques (et vite stp)!**

Variable ?

```
> x <- reactive ({  
>   myData[, input$var]  
> })  
> output$myHist <-  
>   renderPlot({  
>     hist(x())  
>   })  
> output$myStats <-  
>   renderText({  
>     summary(x())  
>   })
```

```
> selectInput(  
>   inputId="var",  
>   label="Variable",  
>   choices = c("Nb_annules" ,  
               "Nb_retard")  
> )  
  
> mainPanel(  
>   plotOutput("myHist")  
>   plotOutput("myStats")  
> )
```

x n'est calculé qu'une fois !

Si on jouait aussi sur l'interface?

Peut-ont faire réagir l'interface de
sélection (ui) à des éléments calculés
par le serveur?

OK, attention, c'est déjà plus compliqué...



server

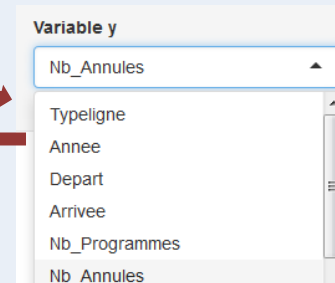
```
> observe ({  
>   updateSelectInput(  
>     session, "var",  
>     choices=colnames(myData))  
>   })
```

```
> x<-reactive ({  
>   myData[, input$var]  
> })  
> output$myHist <-  
> renderPlot({  
>   hist(x())  
> })  
> output$myStats <-  
> renderText({  
>   summary(x())  
> })
```

ui

- Je veux **choisir ma variable** dans la liste des variables disponibles et puis faire un histogramme

```
> selectInput(  
>   inputId="var",  
>   label="Variable",  
>   choices = NULL  
> )
```



```
> mainPanel(  
>   plotOutput("myHist")  
>   plotOutput("myStats")  
> )
```


Structure générale (rappel)

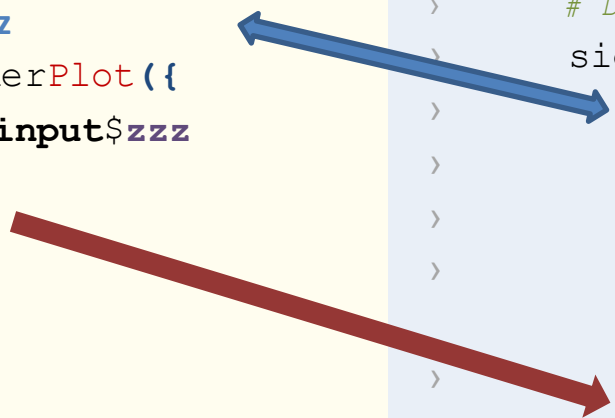
server

```
> library(shiny)
> ... Some R code (libraries, data, functions)

> server <- function(input, output,
  session) {
>   some R code (e.g. observe, reactive)
  some code using input$zzz
>   output$xxx <- renderPlot({
>     some R code using input$zzz
>     ...
>   })
> }
```

ui

```
> library(shiny)
> ui <- fluidPage(
>   # Titre de la page
>   titlePanel("Mon titre"),
>
>   sidebarLayout(
>     # Définition du panneau latéral
>     sidebarPanel(
>       #Type de sélecteur
>       sliderInput("zzz",
>         ... )
>     ),
>
>     # Fenêtre principale
>     mainPanel(
>       plotOutput("xxx")
>     )
>   )
> }
```



Vocabulaire **Shiny**

server

Keywords:

- › `input$zzz`
- › `output$xxx`
- › `renderPlot (Text, Table,...)`

Et puis aussi

- › `observe ({...})`
- › `reactive ({...})`

ui

Panels de l'interface

- › `mainPanel ()`
- › `sidebarPanel ()`
- › `tabsetPanel ()`
- › `titlePanel ()`

Les selecteurs

- › `sliderInput ()`
- › `selectInput ()`
- › `textInput ()`

ui affiche les éléments que le **server** prépare!



Organisation de la journée

- A vous de tester !
- Pré-requis
 - Installation R, Rstudio
 - Chargement package **shiny**
- Récupération des tutoriaux
 - Tests

===== Repas =====

- Exemples avancés
- Tutoriaux (1 à 5)
 - Premiers essais
 - Questions-réponses
 - Debrief
- Conclusion



Scope (English version)

server

```
> library(shiny)
> Some R code # Will load once, when Shiny starts, and will be available to each session
> server <- function(input, output) {
>   Some R code # Objects here are defined in each session
>   output$xxx<- renderPlot({
>     # Objects here are defined each time this function is called
>     some R code using input$zzz
>   })
> }
```

See: <http://shiny.rstudio.com/articles/scoping.html>