

Premiers pas avec shiny... et les packages de visualisation interactive

B.Thieurmél, benoit.thieurmél@quadratic-labs.com

Contents

1	Shiny : créer des applications web avec le logiciel R	3
2	Ma première application avec shiny	3
3	Structure d'une application	5
3.1	Un dossier avec un seul fichier	5
3.2	Un dossier avec deux fichiers	5
3.3	Données/fichiers complémentaires	6
4	Interactivité et communication	7
4.1	Introduction	7
4.2	Process	8
4.3	Notice	9
4.4	UI	9
4.5	Serveur	9
4.6	Retour sur le process	10
4.7	Partage ui <-> server	10
5	Les inputs	11
5.1	Vue globale	11
5.2	Valeur numérique	11
5.3	Chaîne de caractères	11
5.4	Liste de sélection	12
5.5	Checkbox	12
5.6	Checkboxes multiple	13
5.7	Radio boutons	13
5.8	Date	13
5.9	Période	14
5.10	Slider numérique : valeur unique	14
5.11	Slider numérique : range	15
5.12	Importer un fichier	15
5.13	Action Bouton	16
5.14	Aller plus loin : construire son propre input	16
6	Outputs	16
6.1	Vue globale	16
6.2	Les bonnes règles de construction	16
6.3	Print	17
6.4	Text	17
6.5	Table	18
6.6	DataTable	18

6.7	Définir des éléments de l'UI côté SERVER	Définition	19
6.8	Définir des éléments de l'UI côté SERVER	Exemple simple	19
6.9	Définir des éléments de l'UI côté SERVER	Exemple plus complexe	19
6.10	Aller plus loin : construire son propre output		20
7	Structurer sa page		20
7.1	sidebarLayout		20
7.2	wellPanel		20
7.3	navbarPage		21
7.4	tabsetPanel		22
7.5	navlistPanel		22
7.6	Grid Layout		23
7.7	shinydashboard		23
7.8	Combiner les structures		24
8	Graphiques interactifs		24
8.1	Utilisation dans shiny		25
9	Isolation		27
9.1	Définition		27
9.2	Exemple 1		28
9.3	Exemple 2		28
10	Expressions réactives		29
10.1	Exemple sans une expression réactive		29
10.2	Exemple avec une expression réactive		29
10.3	Note		30
10.4	Autres fonctions		30
11	Conditional panels		31
12	Observe & fonctions d'update		32
12.1	Introduction		32
12.2	Exemple sur un input		33
12.3	Exemple sur des onglets		34
12.4	observeEvent		35
13	HTML / CSS		35
13.1	Inclure du HTML		35
13.2	Quelques balises utiles		37
13.3	CSS : introduction		37
13.4	Avec un .css externe		37
13.5	Ajout de css dans le header		38
13.6	CSS sur un élément		39
14	Quelques bonnes pratiques		39
15	Débogage		40
15.1	Affichage console		40
15.2	Lancement manuel d'un browser		40
15.3	Lancement automatique d'un browser		41
15.4	Mode "showcase"		41
15.5	Reactive log		41
15.6	Communication client/server		42
15.7	Traçage des erreurs		42

1 Shiny : créer des applications web avec le logiciel R

Shiny est un package **R** qui permet la création simple d'applications web interactives depuis le logiciel open-source **R**.

- pas de connaissances *web* nécessaires
- le pouvoir de calcul de R et l'interactivité du web actuel
- pour créer des applications locales
- ou partagées avec l'utilisation de **shiny-server**, **shinyapps.io**, **shinyproxy**

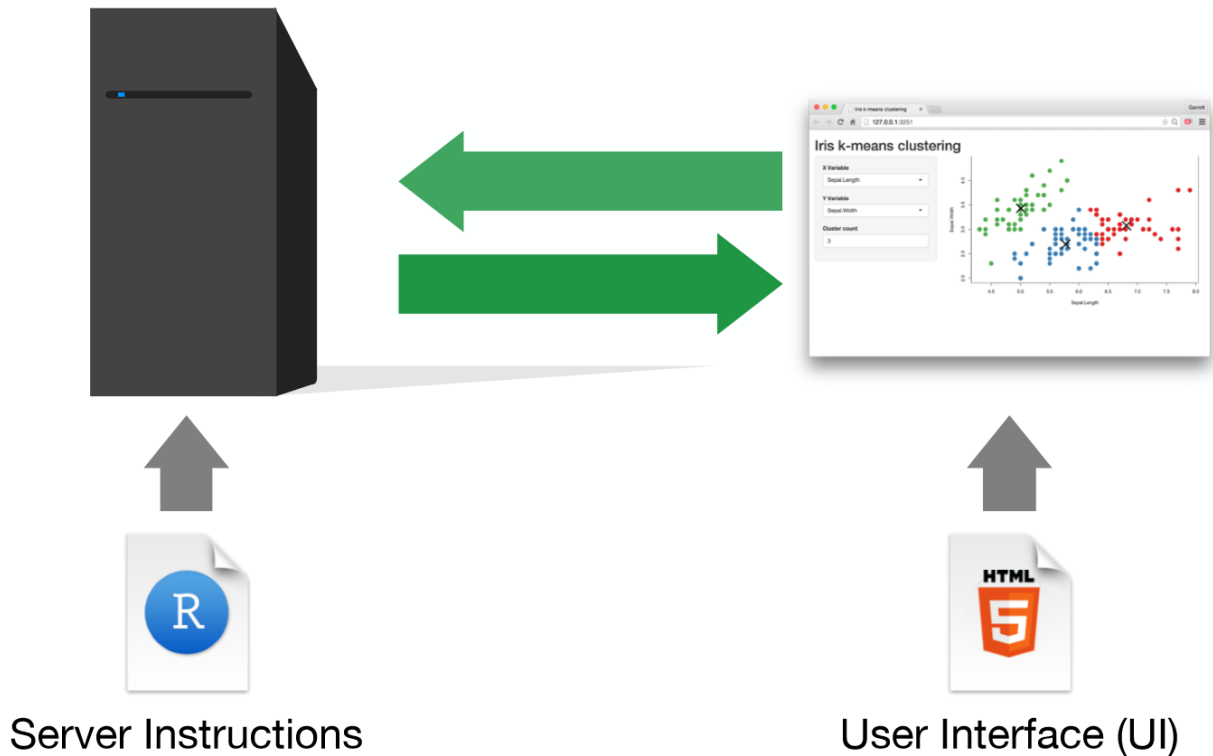
<http://shiny.rstudio.com>

<http://www.shinyapps.io/>

<https://www.shinyproxy.io/>

<https://www.rstudio.com/products/shiny/shiny-server/>

Une application **shiny** nécessite un ordinateur/un serveur exécutant **R**

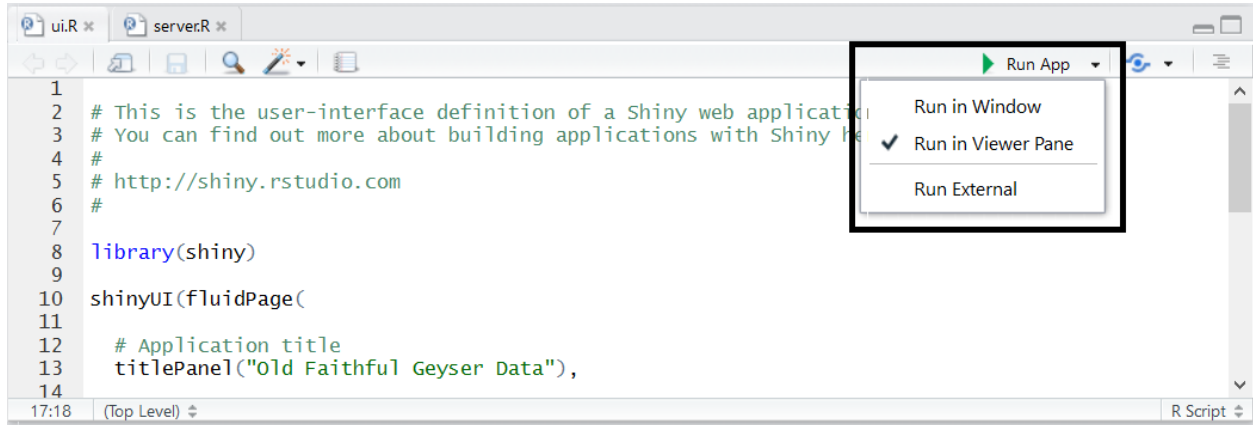


© CC 2015 RStudio, Inc.

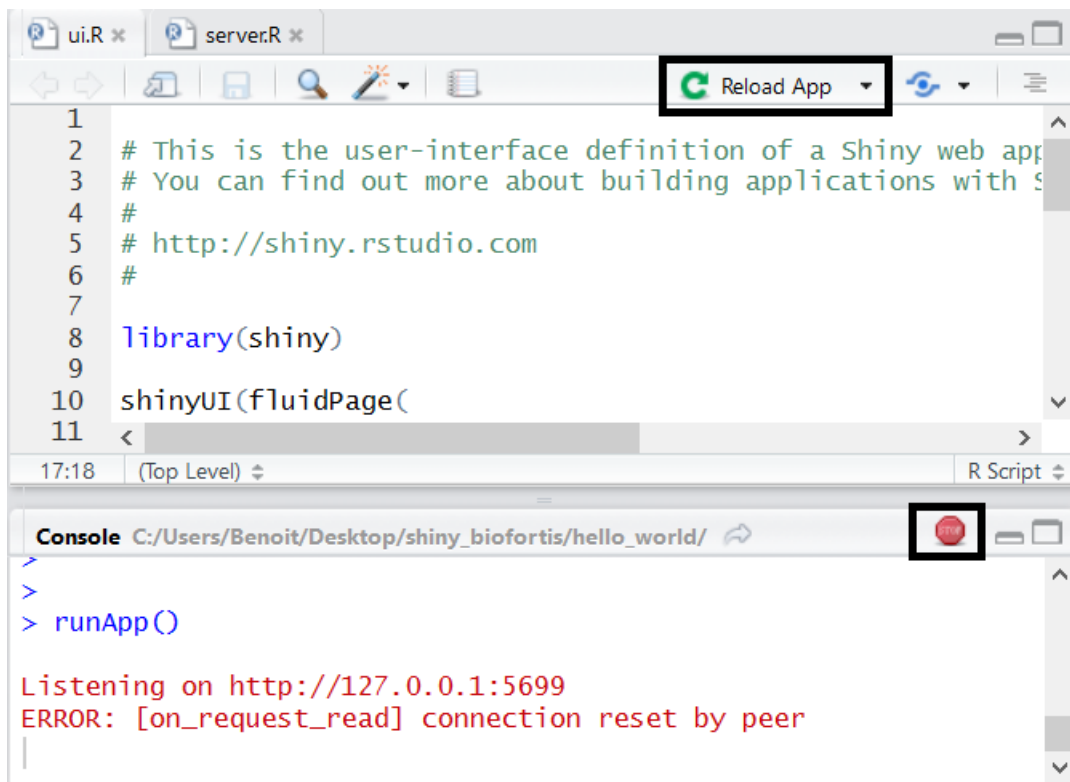
2 Ma première application avec shiny

- Initialiser une application est simple avec **RStudio**, en créant un nouveau projet
- File > New Project > New Directory > Shiny Web Application
- Basée sur deux scripts : **ui.R** et **server.R**
- Et utilisant par défaut le sidebar layout

- Commandes utiles :
- lancement de l'application : bouton **Run app**
- actualisation : bouton **Reload app**
- arrêt : bouton **Stop**



- **Run in Window** : Nouvelle fenêtre, utilisant l'environnement **RStudio**
- **Run in Viewer Pane** : Dans l'onglet *Viewer* de **RStudio**
- **Run External** : Dans le navigateur web par défaut

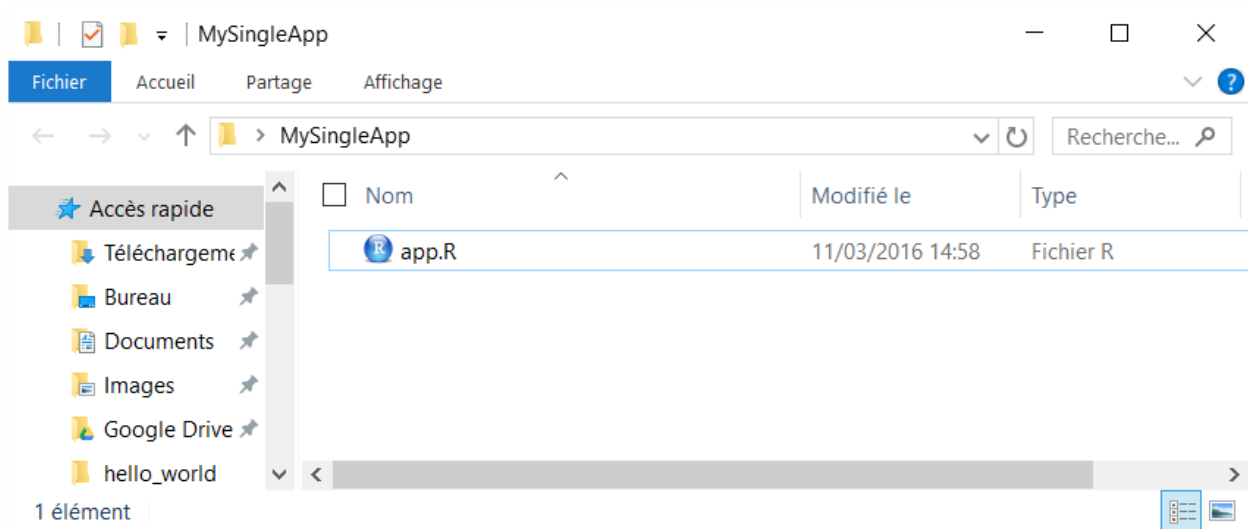


3 Structure d'une application

3.1 Un dossier avec un seul fichier

conventions :

- enregistré sous le nom **app.R**
- se terminant par la commande `shinyApp()`
- pour les **applications légères**

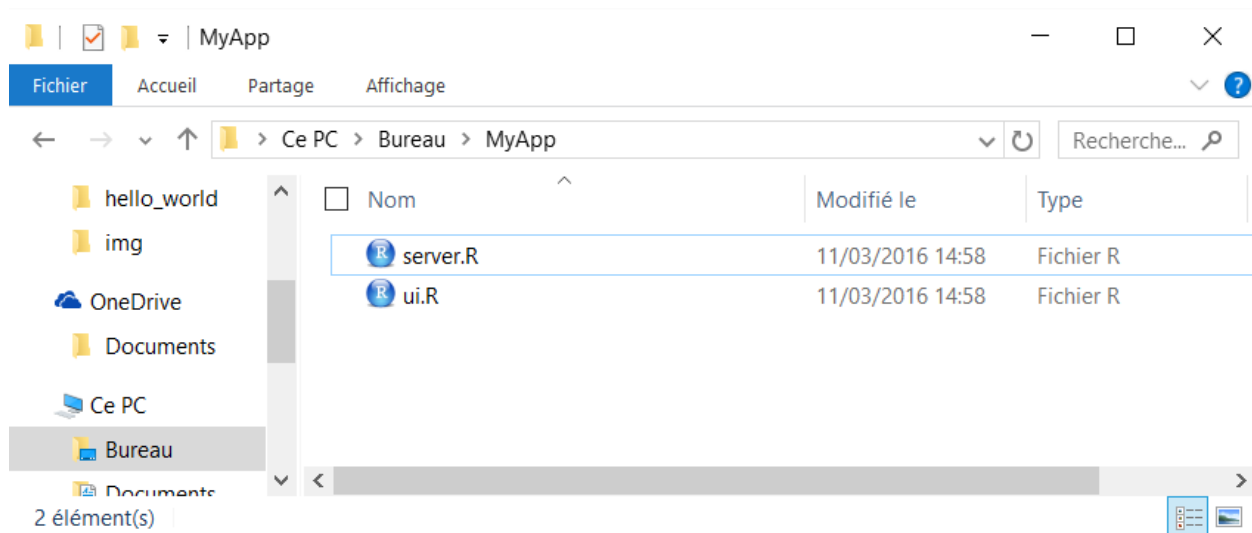


```
library(shiny)
ui <- fluidPage(
  sliderInput(inputId = "num", label = "Choose a number",
    value = 25, min = 1, max = 100),
  plotOutput("hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
shinyApp(ui = ui, server = server)
```

3.2 Un dossier avec deux fichiers

conventions :

- côté interface utilisateur dans le script **ui.R**
- côté serveur dans le script **server.R**
- structure à **privilégier**



ui.R

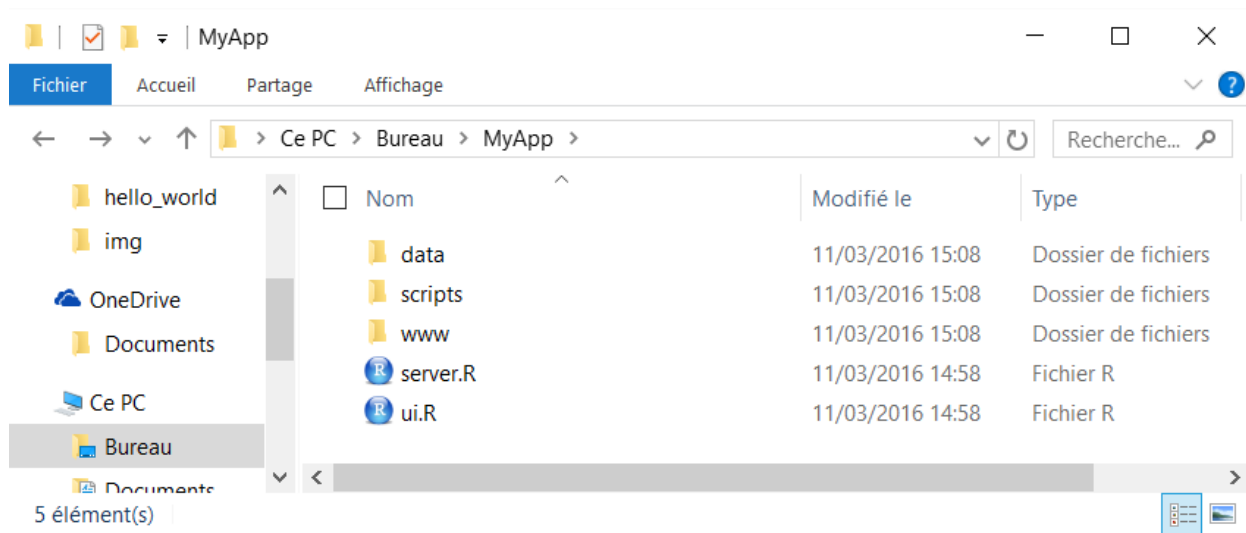
```
library(shiny)
fluidPage(
  sliderInput(inputId = "num", label = "Choose a number",
             value = 25, min = 1, max = 100),
  plotOutput("hist")
)
```

server.R

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

3.3 Données/fichiers complémentaires

- le code **R** tourne au niveau des scripts **R**, et peut donc accéder de façon relative à tous les objets présents dans le dossier de l'application
- l'application web, comme de convention, accède à tous les éléments présents dans le dossier **www** (css, images, javascript, documentation, ...)



4 Interactivité et communication

4.1 Introduction

ui.R:

```
library(shiny)

# Define UI for application that draws a histogram
shinyUI(fluidPage(
  # Application title
  titlePanel("Hello Shiny!"),
  # Sidebar with a slider input for the number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1, max = 50, value = 30)
    ),
    # Show a plot of the generated distribution
    mainPanel(plotOutput(outputId = "distPlot"))
  )
))
```

server.R:

```
library(shiny)

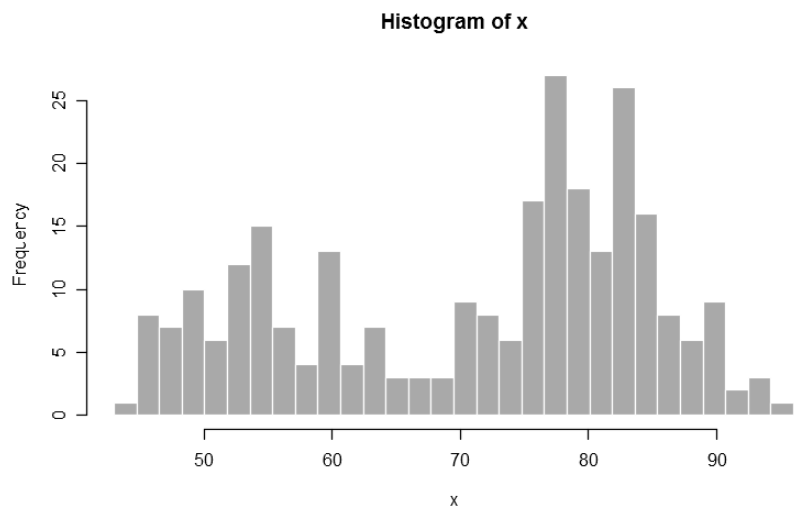
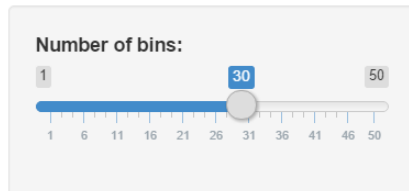
# Define server logic required to draw a histogram
shinyServer(function(input, output) {
  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  # 1) It is "reactive" and therefore should be automatically
  #    re-executed when inputs change
  # 2) Its output type is a plot
  output$distPlot <- renderPlot({
```

```

x <- faithful[, 2] # Old Faithful Geyser data
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')
})
})

```

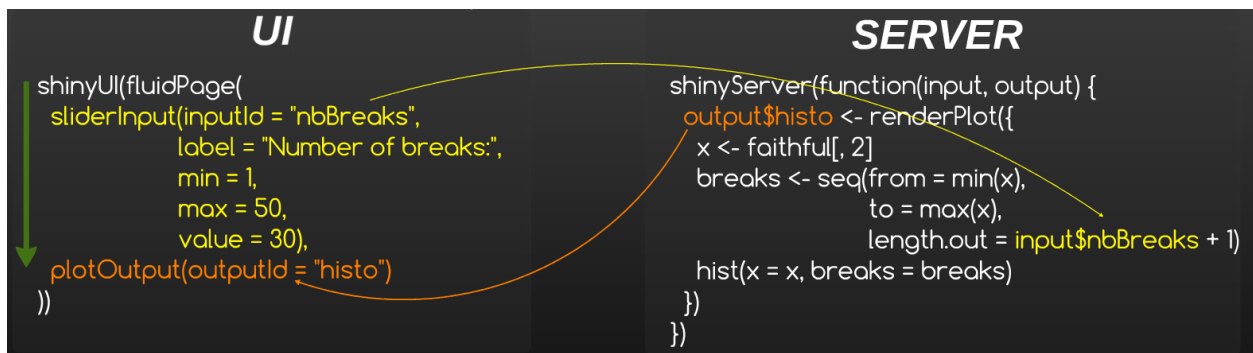
Hello Shiny!



Avec cette exemple simple, nous comprenons :

- Côté **ui**, nous définissons un slider numérique avec le code “`sliderInput(inputId = "bins",...)`” et on utilise sa valeur côté **server** avec la notation “`input$bins`” : c’est comme cela que le **ui** crée des variables disponibles dans le **server** !
- Côté **server**, nous créons un graphique “`output$distPlot <- renderPlot({...})`” et l’appelons dans le **ui** avec “`plotOutput(outputId = "distPlot")`”, c’est comme cela que le **server** retourne des objet à **ui** !

4.2 Process



Le server et l’ui communiquent uniquement par le biais des inputs et des outputs

Par défaut, un output est mis-à-jour chaque fois qu’un input en lien change

4.3 Notice

la définition de l'interface utilisateur : UI

- la déclaration des inputs
- la structure de la page, avec le placement des outputs

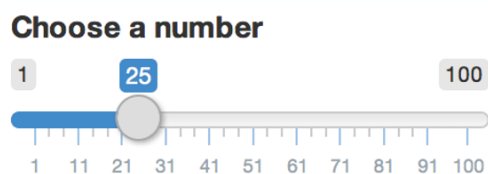
la partie serveur/calculs : SERVER

- la déclaration et le calcul des outputs

4.4 UI

Deux types d'éléments dans le UI

- `xxInput(inputId = ..., ...)`:
- définit un élément qui permet une action de l'utilisateur
- accessible côté serveur avec son identifiant `input$inputID`



`sliderInput(inputId = "num", label = "Choose a number", ...)`

input name
(for internal use)

Notice:
Id not ID

label to
display

input specific
arguments

- `xxOutput(outputId = ...)`:
- fait référence à un output créé et défini côté serveur
- en général : graphiques et tableaux

`plotOutput("hist")`

the type of output
to display

name to give to the
output object

4.5 Serveur

Définition des outputs dans le serveur

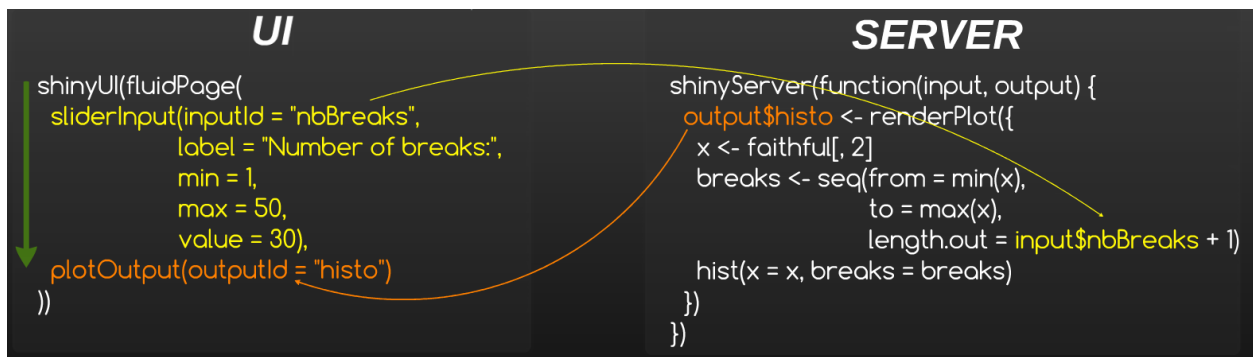
- `renderXX({expr})`:
- calcule et retourne une sortie, dépendante d'inputs, via une expression **R**

```
renderPlot({ hist(rnorm(100)) })
```

type of object to
build

code block that builds
the object

4.6 Retour sur le process



C'est plus clair ?

4.7 Partage ui <-> server

Le server et l'ui communiquent uniquement par le biais des inputs et des outputs

- Nous pouvons ajouter un script nommé **global.R** pour partager des éléments (variables, packages, ...) entre la partie **UI** et la partie **SERVER**
- Tout ce qui est présent dans le **global.R** est visible à la fois dans le **ui.R** et dans le **server.R**
- Le script **global.R** est chargé uniquement une seule fois au lancement de l'application
- Dans le cas d'une utilisation avec un **shiny-server**, les objets globaux sont également partagés entre les utilisateurs

5 Les inputs

5.1 Vue globale

Buttons



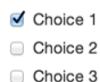
`actionButton()`
`submitButton()`

Single checkbox



`checkboxInput()`

Checkbox group



`checkboxGroupInput()`

Date input



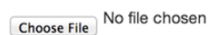
`dateInput()`

Date range



`dateRangeInput()`

File input



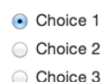
`fileInput()`

Password Input



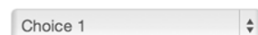
`passwordInput()`

Radio buttons



`radioButtons()`

Select box



`selectInput()`

Sliders



`sliderInput()`

Text input



`textInput()`

5.2 Valeur numérique

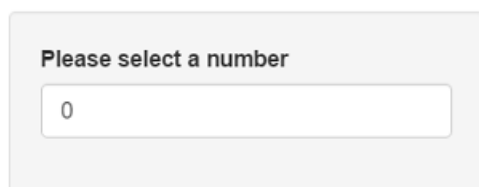
- La fonction

```
numericInput(inputId, label, value, min = NA, max = NA, step = NA)
```

- Exemple:

```
numericInput(inputId = "idNumeric", label = "Please select a number",
             value = 0, min = 0, max = 100, step = 10)
```

```
# For the server input$idNumeric will be of class "numeric"
# ("integer" when the parameter step is an integer value)
```



Value:

[1] 0

Class:

integer

5.3 Chaîne de caractères

- La fonction

```
textInput(inputId, label, value = "")
```

- Exemple:

```
textInput(inputId = "idText", label = "Enter a text", value = "")

# For the server input$idText will be of class "character"
```

Enter a text

Value:
Class:

5.4 Liste de sélection

- La fonction

```
selectInput(inputId, label, choices, selected = NULL, multiple = FALSE,
            selectize = TRUE, width = NULL, size = NULL)
```

- Exemple:

```
selectInput(inputId = "idSelect", label = "Select among the list: ", selected = 3,
            choices = c("First" = 1, "Second" = 2, "Third" = 3))

# For the server input$idSelect is of class "character"
# (vector when the parameter "multiple" is TRUE)
```

Select among the list:

Value:
Class:

Select among the list:

Value:
Class:

5.5 Checkbox

- La fonction

```
checkboxInput(inputId, label, value = FALSE)
```

- Exemple:

```
checkboxInput(inputId = "idCheck1", label = "Check ?")

# For the server input$idCheck1 is of class "logical"
```

checkboxInput

☒ Check ?

Value: [1] TRUE
Class: logical

5.6 Checkboxes multiple

- La fonction

```
checkboxGroupInput(inputId, label, choices, selected = NULL, inline = FALSE)
```

- Exemple:

```
checkboxGroupInput(inputId = "idCheckGroup", label = "Please select", selected = 3,
  choices = c("First" = 1, "Second" = 2, "Third" = 3))
```

For the server input\$idCheckGroup is a "character" vector

Please select

☐ First
☒ Second
☒ Third

Value: [1] "2" "3"
Class: character

5.7 Radio boutons

- La fonction

```
radioButtons(inputId, label, choices, selected = NULL, inline = FALSE)
```

- Exemple:

```
radioButtons(inputId = "idRadio", label = "Select one", selected = 3,
  choices = c("First" = 1, "Second" = 2, "Third" = 3))
```

For the server input\$idRadio is a "character"

Select one

☐ First
☐ Second
☒ Third

Value: [1] "3"
Class: character

5.8 Date

- La fonction

```
dateInput(inputId, label, value = NULL, min = NULL, max = NULL, format = "yyyy-mm-dd",
          startview = "month", weekstart = 0, language = "en")
```

- Exemple:

```
dateInput(inputId = "idDate", label = "Please enter a date", value = "12/08/2015",
          format = "dd/mm/yyyy", startview = "month", weekstart = 0, language = "fr")
```

For the server input\$idDate is a "Date"

<div style="border: 1px solid #ccc; padding: 10px; width: 250px;"> <p>Please enter a date</p> <input type="text" value="07/12/2015"/> </div>	<p>Value:</p> <div style="border: 1px solid #ccc; padding: 5px; width: 150px;">[1] "2015-12-07"</div>
	<p>Class:</p> <div style="border: 1px solid #ccc; padding: 5px; width: 150px;">Date</div>

5.9 Période

- La fonction

```
dateRangeInput(inputId, label, start = NULL, end = NULL, min = NULL, max = NULL,
               format = "yyyy-mm-dd", startview = "month", weekstart = 0,
               language = "en", separator = " to ")
```

- Exemple:

```
dateRangeInput(inputId = "idDateRange", label = "Please Select a date range",
               start = "2015-01-01", end = "2015-08-12", format = "yyyy-mm-dd",
               language = "en", separator = " to ")
```

For the server input\$idDateRange is a vector of class "Date" with two elements

<div style="border: 1px solid #ccc; padding: 10px; width: 250px;"> <p>Please Select a date range</p> <div style="display: flex; align-items: center;"> <input type="text" value="2015-01-01"/> to <input type="text" value="2015-08-12"/> </div> </div>	<p>Value:</p> <div style="border: 1px solid #ccc; padding: 5px; width: 150px;">[1] "2015-01-01" "2015-08-12"</div>
	<p>Class:</p> <div style="border: 1px solid #ccc; padding: 5px; width: 150px;">Date</div>

5.10 Slider numérique : valeur unique

- La fonction

```
sliderInput(inputId, label, min, max, value, step = NULL, round = FALSE,
            format = NULL, locale = NULL, ticks = TRUE, animate = FALSE,
            width = NULL, sep = ",", pre = NULL, post = NULL)
```

- Exemple:

```
sliderInput(inputId = "idSlider1", label = "Select a number", min = 0, max = 10,
            value = 5, step = 1)
```

For the server input\$idSlider1 is a "numeric"

(integer when the parameter "step" is an integer too)

Value: [1] 5

Class: integer

5.11 Slider numérique : range

- La fonction

```
sliderInput(inputId, label, min, max, value, step = NULL, round = FALSE,
            format = NULL, locale = NULL, ticks = TRUE, animate = FALSE,
            width = NULL, sep = ",", pre = NULL, post = NULL)
```

- Exemple:

```
sliderInput(inputId = "idSlider2", label = "Select a number", min = 0, max = 10,
            value = c(2,7), step = 1)
```

*# For the server input\$idSlider2 is a "numeric" vector
(integer when the parameter "step" is an integer too)*

Value: [1] 2 7

Class: integer

5.12 Importer un fichier

- La fonction

```
fileInput(inputId, label, multiple = FALSE, accept = NULL)
```

- Exemple:

```
fileInput(inputId = "idFile", label = "Select a file")
```

*# For the server input\$idFile is a "data.frame" with four "character" columns
(name, size, type and datapath) and one row*

Value:

	name	size	type	datapath
1	tab2.csv	40	application/vnd.ms-excel	C:\Users\Benoit\AppData

5.13 Action Bouton

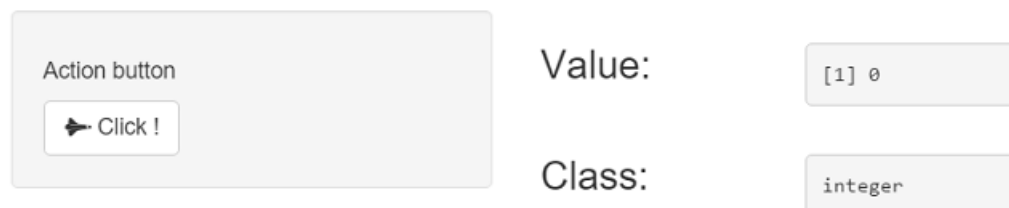
- La fonction

```
actionButton(inputId, label, icon = NULL, ...)
```

- Exemple:

```
actionButton(inputId = "idActionButton", label = "Click !",  
             icon = icon("hand-spock-o"))
```

For the server input\$idActionButton is an "integer"



5.14 Aller plus loin : construire son propre input

Avec un peu de compétences en HTML/CSS/JavaScript, il est également possible de construire des inputs personnalisés

Un tutoriel est disponible : <http://shiny.rstudio.com/articles/building-inputs.html>

Ainsi que deux applications d'exemples :

- <http://shiny.rstudio.com/gallery/custom-input-control.html>
- <http://shiny.rstudio.com/gallery/custom-input-bindings.html>

6 Outputs

6.1 Vue globale

server fonction	ui fonction	type de sortie
renderDataTable()	dataTableOutput()	une table interactive
renderImage()	imageOutput()	une image sauvegardée
renderPlot()	plotOutput	un graphique R
renderPrint()	verbatimTextOutput()	affichage type console R
renderTable()	tableOutput()	une table statique
renderText()	textOutput()	une chaîne de caractère
renderUI()	uiOutput()	un élément de type UI

6.2 Les bonnes règles de construction

- assigner l'output à afficher dans la liste **output**, avec un nom permettant l'identification côté **UI**
- utiliser une fonction **renderXX({expr})**
- la dernière expression doit correspondre au type d'objet retourné
- accéder aux inputs, et amener la réactivité, en utilisant la liste **input** et l'identifiant : **input\$inputId**


```
#ui.R
selectInput("lettre", "Lettres:", LETTERS[1:3])
verbatimTextOutput(outputId = "selection")
#server.R
output$selection <- renderPrint({input$lettre})
```

6.3 Print

- ui.r:

```
verbatimTextOutput(outputId = "texte")
```

- server.r:

```
output$texte <- renderPrint({
  c("Hello shiny !")
})
```

```
[1] "Hello shiny !"
```

6.4 Text

- ui.r:

```
textOutput(outputId = "texte")
```

- server.r:

```
output$texte <- renderText({
  c("Hello shiny !")
})
```

Hello shiny !

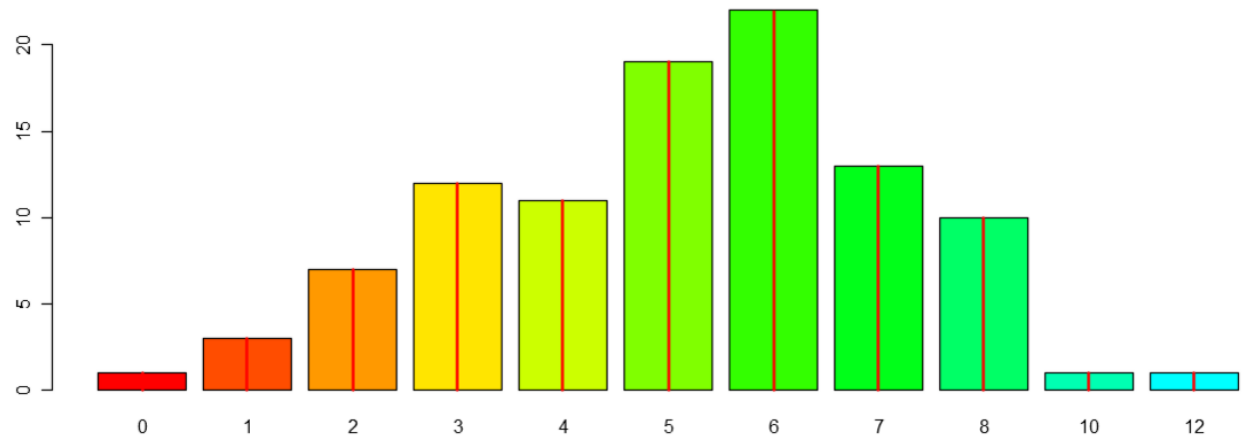
6.4.1 Plot

- ui.r:

```
plotOutput("myplot")
```

- server.r:

```
output$myplot <- renderPlot({
  hist(iris$Sepal.Length)
})
```



6.5 Table

- ui.r:

```
tableOutput(outputId = "table")
```

- server.r:

```
output$table <- renderTable({iris})
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa

6.6 DataTable

- ui.r:

```
dataTableOutput(outputId = "dataTable")
```

- server.r:

```
output$dataTable <- renderDataTable({
  iris
})
```

Show entries
 Search:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

Showing 1 to 5 of 5 entries

Previous 1 Next

6.7 Définir des éléments de l'UI côté SERVER | Définition

Dans certains cas, nous souhaitons définir des inputs ou des structures côté server

Cela est possible avec les fonctions `uiOutput` et `renderUI`

6.8 Définir des éléments de l'UI côté SERVER | Exemple simple

- **ui.r:**

```
uiOutput(outputId = "columns")
```

- **server.r:**

```
output$columns <- renderUI({
  selectInput(inputId = "sel_col", label = "Column", choices = colnames(data))
})
```

dataset :

faithful ▼

Column

eruptions ▲

eruptions

waiting

dataset :

iris ▼

Column

Sepal.Length ▲

Sepal.Length

Sepal.Width

Petal.Length

6.9 Définir des éléments de l'UI côté SERVER | Exemple plus complexe

- On peut également renvoyer un élément plus complexe de l'UI, par exemple :
 - tout en layout
 - ou une `fluidRow`
- **ui.r:**

```
uiOutput(outputId = "fluidRow_ui")
```

- **server.r:**

```
output$fluidRow_ui <- renderUI(
  fluidRow(
    column(width = 3, h3("Value:")),
    column(width = 3, h3(verbatimTextOutput(outputId = "slinderIn_value")))
  )
)
```

6.10 Aller plus loin : construire son propre output

Avec un peu de compétences en HTML/CSS/JavaScript, il est également possible de construire des outputs personnalisés

Un tutoriel est disponible : <http://shiny.rstudio.com/articles/building-outputs.html>

On peut donc par exemple ajouter comme output un graphique construit avec la librairie d3.js. Un exemple est disponible dans le dossier `shinyApps/build_output`.

7 Structurer sa page

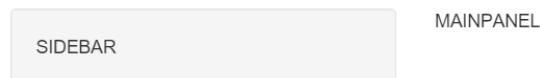
7.1 sidebarLayout

Le template basique `sidebarLayout` divise la page en deux colonnes et doit contenir :

- `sidebarPanel`, à gauche, en général pour les inputs
- `mainPanel`, à droite, en général pour les outputs

```
shinyUI(fluidPage(
  titlePanel("Old Faithful Geyser Data"), # title
  sidebarLayout(
    sidebarPanel("SIDEBAR"),
    mainPanel("MAINPANEL")
  )
))
```

My first app



7.2 wellPanel

Comme avec le `sidebarPanel` précédent, on peut grouper un ensemble d'éléments en utilisant un `wellPanel` :

```
shinyUI(fluidPage(
  titlePanel("Old Faithful Geyser Data"), # title
  wellPanel(
    sliderInput("num", "Choose a number", value = 25, min = 1, max = 100),
    textInput("title", value = "Histogram", label = "Write a title")
  ),
)
```

```
plotOutput("hist")
))
```

Without wellPanel

Choose a number

1 25 100

1 11 21 31 41 51 61 71 81 91 100

Write a title

Histogram

With wellPanel

Choose a number

1 25 100

1 11 21 31 41 51 61 71 81 91 100

Write a title

Histogram

7.3 navbarPage

Utiliser une barre de navigation et des onglets avec `navbarPage` et `tabPanel`:

```
shinyUI(
  navbarPage(
    title = "My first app",
    tabPanel(title = "Summary",
             "Here is the summary"),
    tabPanel(title = "Plot",
             "some charts"),
    tabPanel(title = "Table",
             "some tables")
  )
)
```

Nous pouvons rajouter un second niveau de navigation avec un `navbarMenu` :

```
shinyUI(
  navbarPage(
    title = "My first app",
    tabPanel(title = "Summary",
             "Here is the summary"),
    tabPanel(title = "Plot",
             "some charts"),
    navbarMenu("Table",
               tabPanel("Table 1"),
               tabPanel("Table 2")
    )
  )
)
```

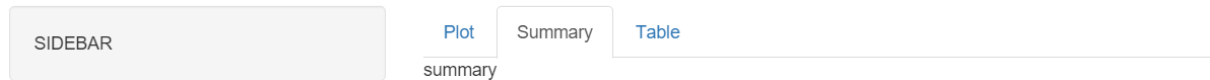


7.4 tabsetPanel

Plus généralement, nous pouvons créer des onglets à n'importe quel endroit en utilisant `tabsetPanel` & `tabPanel`:

```
shinyUI(fluidPage(
  titlePanel("Old Faithful Geyser Data"), # title
  sidebarLayout(
    sidebarPanel("SIDEBAR"),
    mainPanel(
      tabsetPanel(
        tabPanel("Plot", plotOutput("plot")),
        tabPanel("Summary", verbatimTextOutput("summary")),
        tabPanel("Table", tableOutput("table"))
      )
    )
  )
))
```

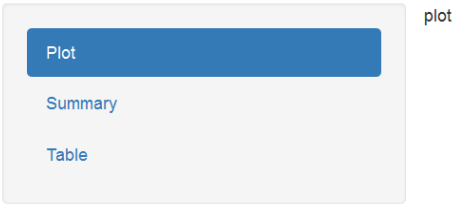
My first app



7.5 navlistPanel

Une alternative au `tabsetPanel`, pour une disposition verticale plutôt qu'horizontale : `navlistPanel`

```
shinyUI(fluidPage(
  navlistPanel(
    tabPanel("Plot", plotOutput("plot")),
    tabPanel("Summary", verbatimTextOutput("summary")),
    tabPanel("Table", tableOutput("table"))
  )
))
```

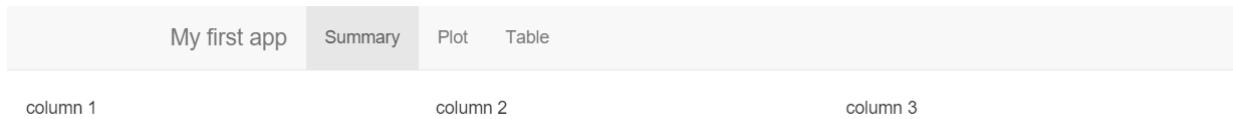


7.6 Grid Layout

Créer sa propre organisation avec `fluidRow()` et `column()`

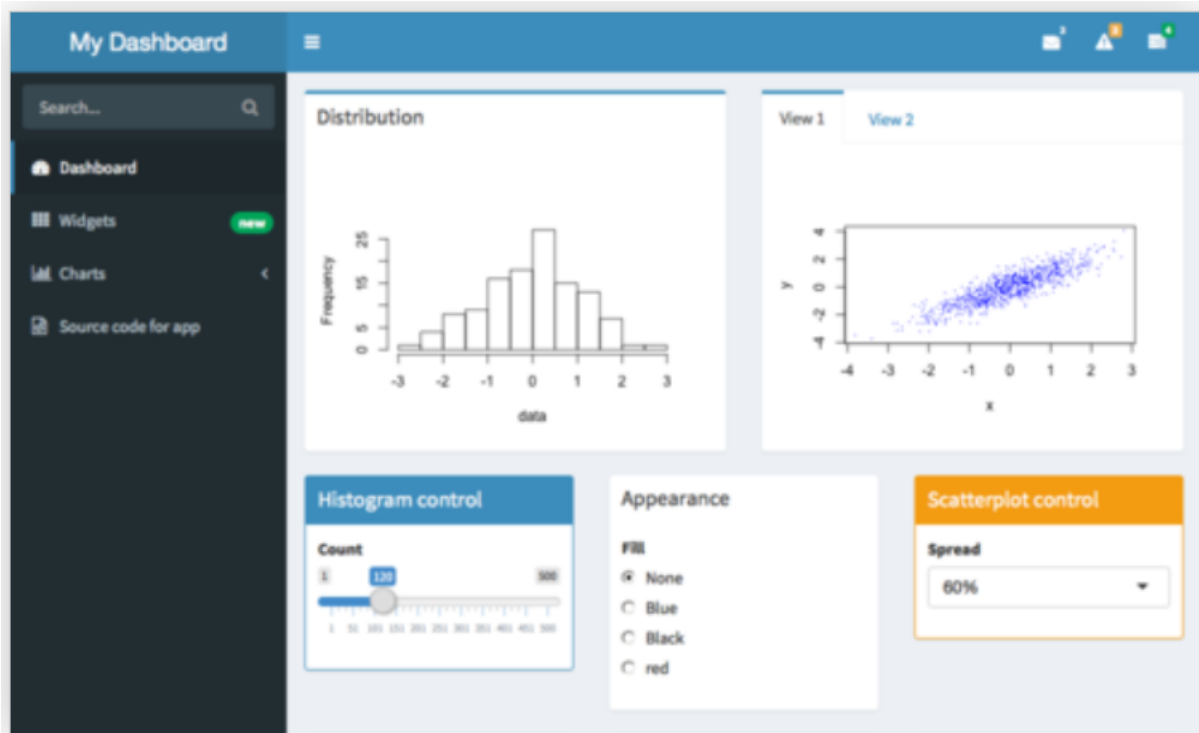
- chaque ligne peut être divisée en 12 colonnes
- le dimensionnement final de la page est automatique en fonction des éléments dans les lignes / colonnes

```
tabPanel(title = "Summary",
  # A fluid row can contain from 0 to 12 columns
  fluidRow(
    # A column is defined necessarily
    # with its argument "width"
    column(width = 4, "column 1"),
    column(width = 4, "column 2"),
    column(width = 4, "column 3"),
  ))
```



7.7 shinydashboard

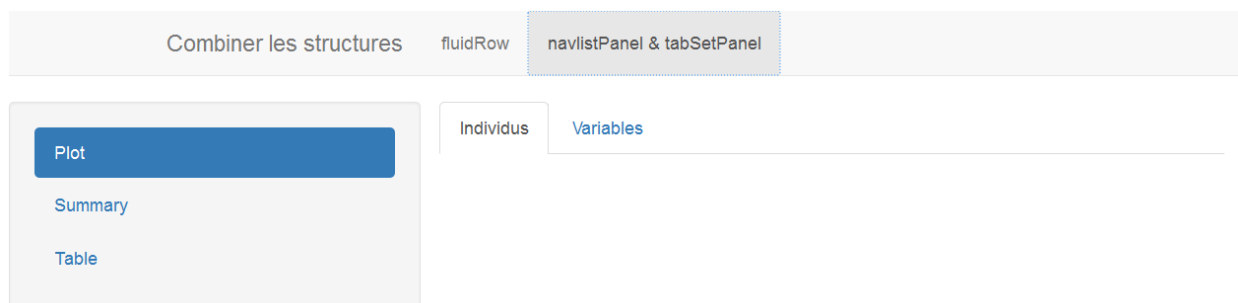
Le package `shinydashboard` propose d'autres fonctions pour créer des tableaux de bords :



<https://rstudio.github.io/shinydashboard/>

7.8 Combiner les structures

Toutes les structures peuvent s'utiliser en même temps !



8 Graphiques interactifs

Avec notamment l'arrivée du package `htmlwidgets`, de plus en plus de fonctionnalités de bibliothèques javascript sont accessibles sous **R** :

- `dygraphs` (time series)
- `DT` (interactive tables)
- `Leaflet` (maps)
- `d3heatmap`
- `threejs` (3d scatter & globe)
- `rAmCharts`
- `visNetwork`
- ...

Plus généralement, jeter un oeil sur la gallerie suivante!

8.1 Utilisation dans shiny

Tous ces packages sont utilisables simplement dans **shiny**. En effet, ils contiennent les deux fonctions nécessaires :

- **renderXX**
- **xxOutput**

Par exemple avec le package **dygraphs** :

```
# Server
output$dygraph <- renderDygraph({
  dygraph(predicted(), main = "Predicted Deaths/Month")
})
# Ui
dygraphOutput("dygraph")
```

Ces packages arrivent souvent avec des méthodes permettant d'interagir avec le graphique, en créant des inputs dans **shiny** afin de déclencher des actions . Par exemple :

- **DT** : création de *input\$tableId_rows_selected*, nous informant sur la/les lignes sélectionnée(s)
- **Leaflet** : valeurs du zoom, des clicks, de la latitude/longitude, ...
- **visNetwork** : noeuds / groupes sélectionnés, ...

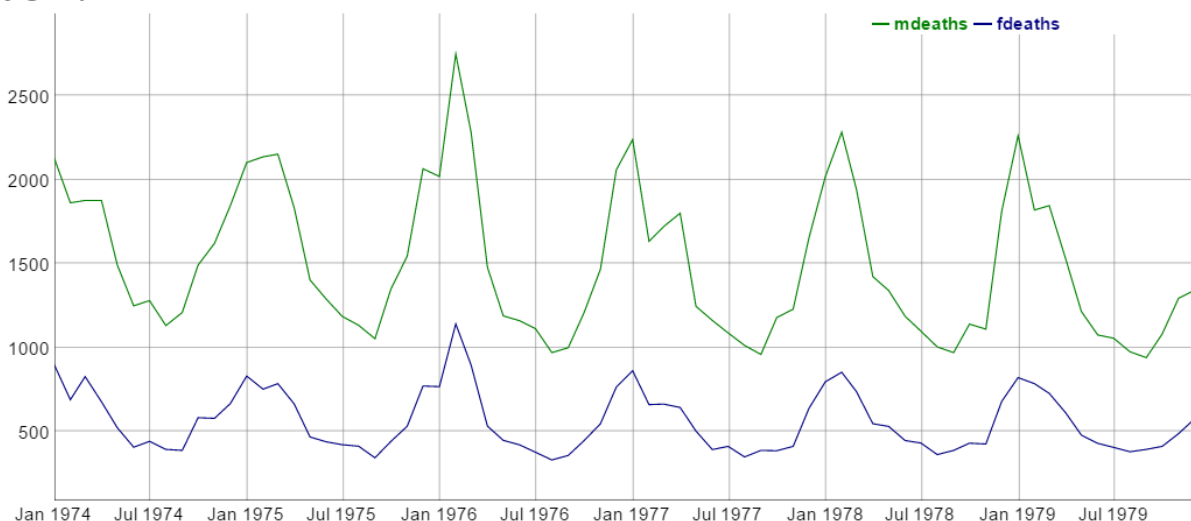
Ces points sont (en général) expliqués sur les pages web des différents packages...

De plus, il est également possible d'utiliser de nombreux événements javascripts, et de créer des nouvelles interactions avec **shiny** en utilisant *Shiny.onInputChange* :

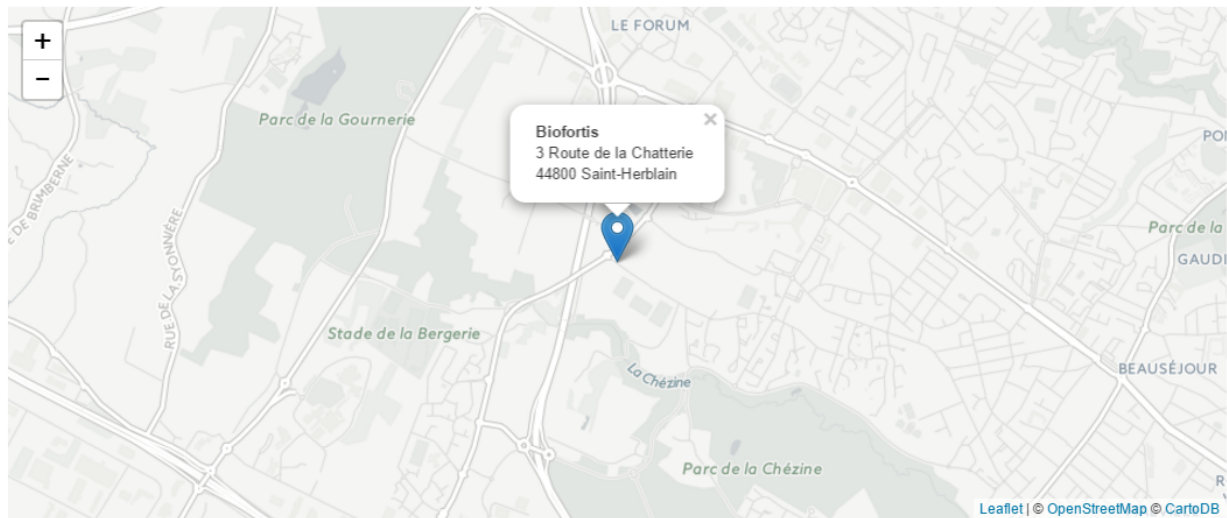
```
visNetwork(nodes, edges) %>%
  visEvents(hoverNode = "function(nodes) {
    Shiny.onInputChange('current_node_id', nodes);
  }")
```

<https://shiny.rstudio.com/articles/js-send-message.html>

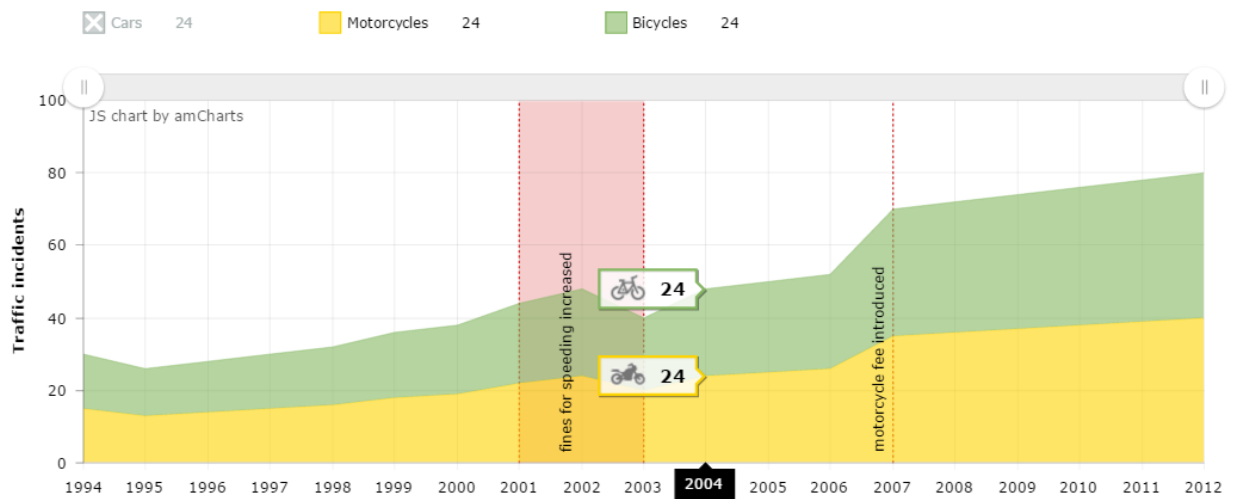
dygraphs



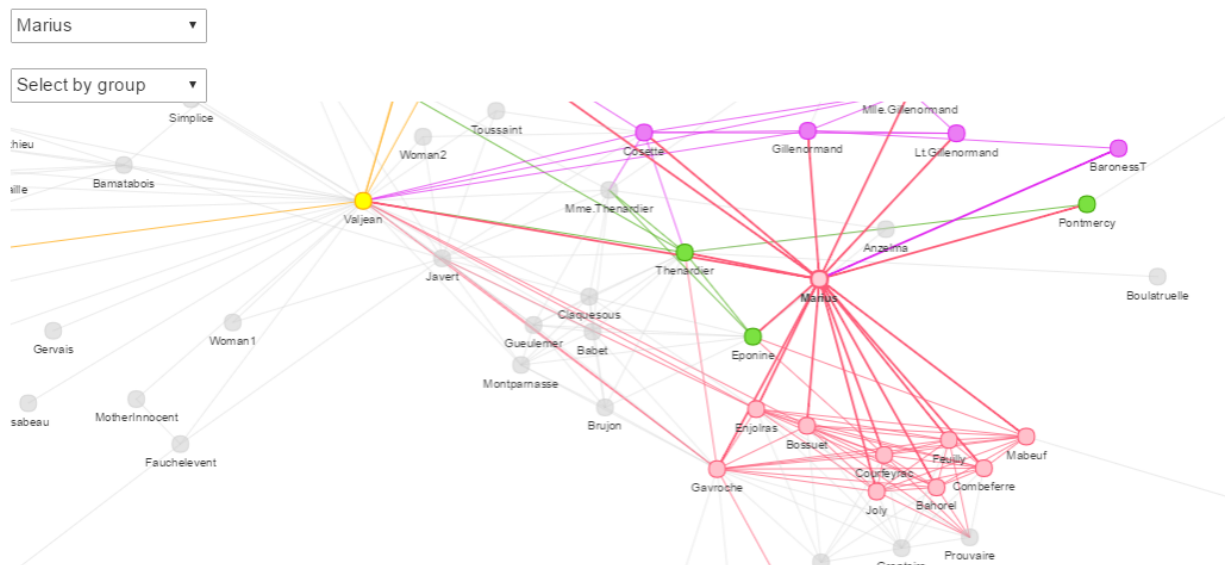
leaflet



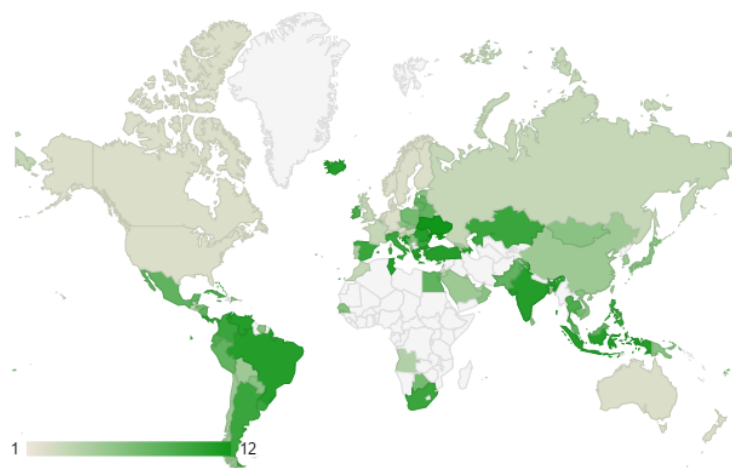
rAmCharts



visNetwork



googleVis Example



Data from:

https://en.wikipedia.org/wiki/List_of_countries_by_credit_rating

9 Isolation

9.1 Définition

Par défaut, les outputs et les expressions réactives se mettent à jour automatiquement quand un des inputs présents dans le code change de valeur. Dans certains cas, on aimerait pouvoir contrôler un peu cela.

Par exemple, en utilisant un bouton de validation (**actionButton**) des inputs pour déclencher le calcul des sorties.

- un input peut être isolé comme cela `isolate(input$id)`
- une expression avec la notation suivante `isolate({expr})` et l'utilisation de `{}`

9.2 Exemple 1

- **ui.r:** Trois inputs : **color** et **bins** pour l'histogramme, et un **actionButton** :

```
shinyUI(fluidPage(  
  titlePanel("Isolation"),  
  sidebarLayout(  
    sidebarPanel(  
      radioButtons(inputId = "col", label = "Choose a color", inline = TRUE,  
                   choices = c("red", "blue", "darkgrey")),  
      sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),  
      actionButton("go_graph", "Update !")  
    ),  
    mainPanel(plotOutput("distPlot"))  
  )  
))
```

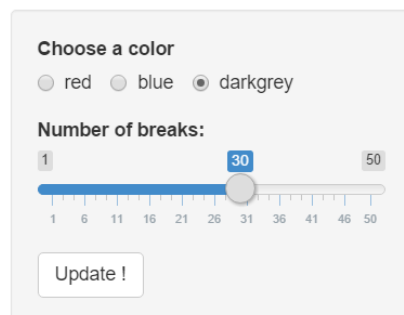
- **server.r:**

On isole tout le code sauf l'**actionButton** :

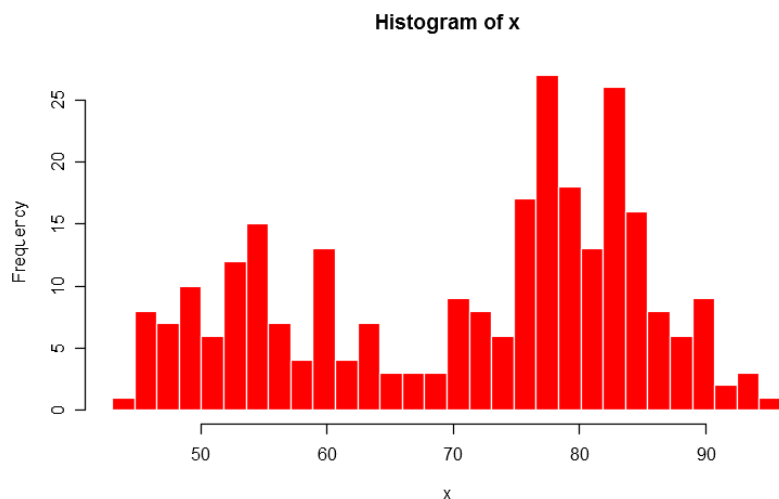
```
shinyServer(function(input, output) {  
  output$distPlot <- renderPlot({  
    input$go_graph  
    isolate({  
      inputColor <- input$color  
      x <- faithful[, 2]  
      bins <- seq(min(x), max(x), length.out = input$bins + 1)  
      hist(x, breaks = bins, col = inputColor, border = 'white')  
    })  
  })  
})
```

L'histogramme sera donc mis-à-jour quand l'utilisateur cliquera sur le bouton.

Isolation



The UI consists of a sidebar with two controls: a radio button group titled "Choose a color" with options "red", "blue", and "darkgrey" (selected), and a slider input titled "Number of breaks:" with a range from 1 to 50 and a current value of 30. Below the slider is an "Update !" button.



9.3 Exemple 2

- **server.r:**

```
output$distPlot <- renderPlot({
  input$go_graph
  inputColor <- input$color
  isolate({
    x <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = inputColor, border = 'white')
  })
})
```

Même résultat en isolant seulement le troisième et dernier input `input$bins`

```
input$go_graph
x <- faithful[, 2]
bins <- seq(min(x), max(x), length.out = isolate(input$bins) + 1)
hist(x, breaks = bins, col = input$color, border = 'white')
```

L'histogramme sera donc mis-à-jour quand l'utilisateur cliquera sur le bouton ou quand la couleur changera.

10 Expressions réactives

Les expressions réactives sont très utiles quand on souhaite utiliser le même résultat/objet dans plusieurs outputs, en ne faisant le calcul qu'une fois.

Il suffit pour cela d'utiliser la fonction `reactive` dans le **server.R**

Par exemple, nous voulons afficher deux graphiques à la suite d'une ACP:

- La projection des individus
- La projection des variables

10.1 Exemple sans une expression réactive

- **server.R**: le calcul est réalisé deux fois...

```
require(FactoMineR) ; data("decathlon")

output$graph_pca_ind <- renderPlot({
  res_pca <- PCA(decathlon[,input$variables], graph = FALSE)
  plot.PCA(res_pca, choix = "ind", axes = c(1,2))
})

output$graph_pca_var <- renderPlot({
  res_pca <- PCA(decathlon[,input$variables], graph = FALSE)
  plot.PCA(res_pca, choix = "var", axes = c(1,2))
})
```

10.2 Exemple avec une expression réactive

- **server.R** : Le calcul est maintenant effectué qu'une seule fois !

```
require(FactoMineR) ; data("decathlon")

res_pca <- reactive({
  PCA(decathlon[,input$variables], graph = FALSE)
})
```

```

output$graph_pca_ind <- renderPlot({
  plot.PCA(res_pca(), choix = "ind", axes = c(1,2))
})

output$graph_pca_var <- renderPlot({
  plot.PCA(res_pca(), choix = "var", axes = c(1,2))
})

```

10.3 Note

- Une expression réactive va nous faire gagner du temps et de la mémoire
- **Utiliser des expressions réactives seulement quand cela dépend d'inputs** (pour d'autres variables : <http://shiny.rstudio.com/articles/scoping.html>)
- **Comme un output** : mis-à-jour chaque fois qu'un input présent dans le code change
- **Comme un input** dans un *renderXX* : l'output est mis-à-jour quand l'expression réactive change
- On récupère sa valeur comme un appel à une fonction, avec des "()".

10.4 Autres fonctions

Il existe des alternatives à l'utilisation de `reactive` avec `reactiveValues` ou `reactiveVal`.

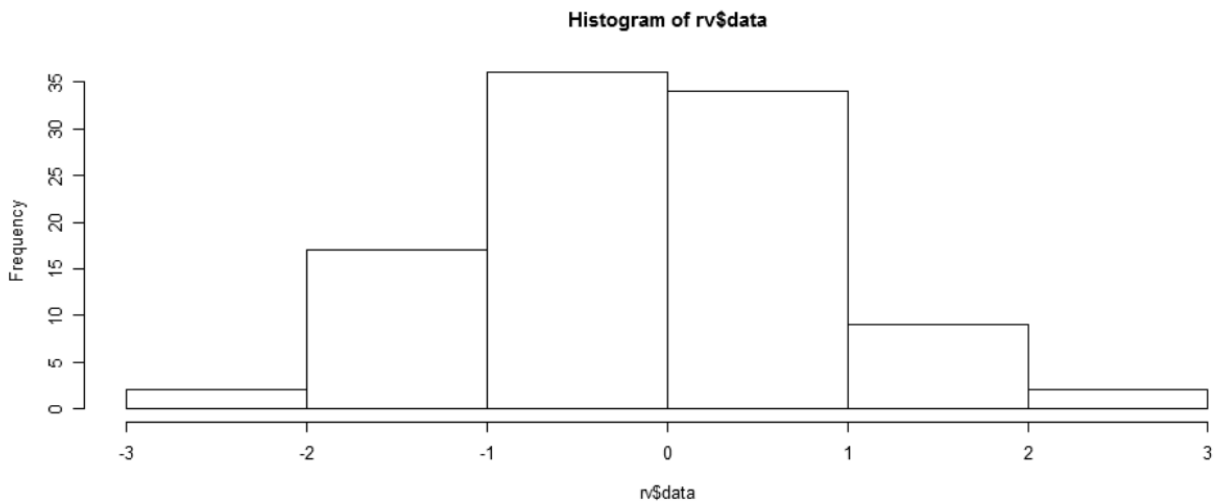
- `reactiveValues` : initialiser une liste d'objets réactifs
- `reactiveVal` : initialiser un seul objet réactif
- Modification de la valeur des objets avec des `observe` ou des `observeEvent`

```

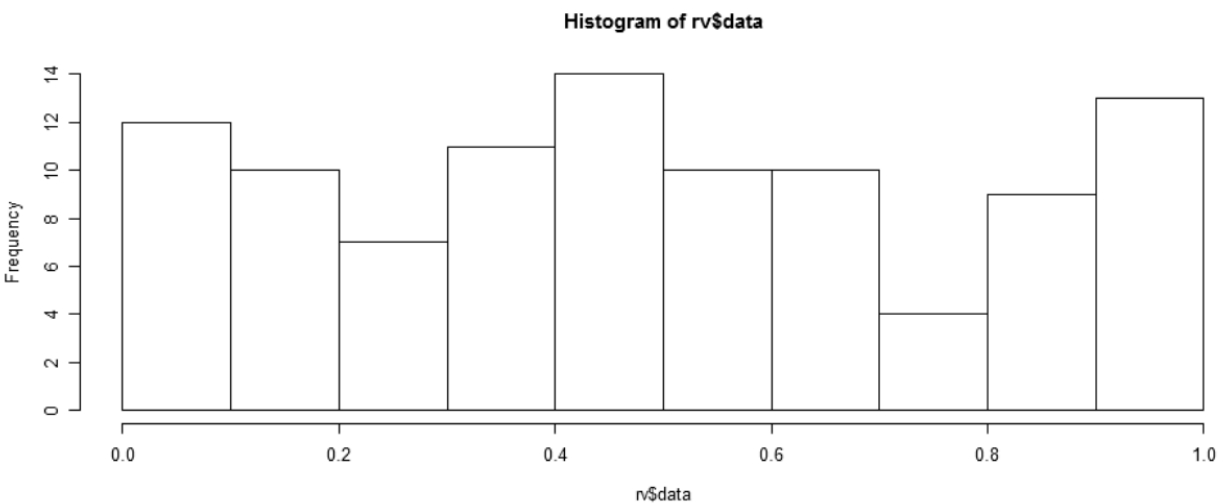
shinyApp(ui = fluidPage(
  actionButton(inputId = "norm", label = "Normal"),
  actionButton(inputId = "unif", label = "Uniform"),
  plotOutput("hist")
),
server = function(input, output) {
  rv <- reactiveValues(data = rnorm(100))
  observeEvent(input$norm, { rv$data <- rnorm(100) })
  observeEvent(input$unif, { rv$data <- runif(100) })
  output$hist <- renderPlot({ hist(rv$data) })
})

```

Normal Uniform



Normal Uniform



11 Conditional panels

- Il est possible d'afficher conditionnellement ou non certains éléments :

```
conditionalPanel(condition = [...], )
```

- La condition peut se faire sur des inputs ou des outputs
- Elle doit être rédigée en **javascript**...

```
conditionalPanel(condition = "input.checkbox == true", [...])
```

```
library(shiny)
shinyApp(
  ui = fluidPage(
    fluidRow(
      column(
        width = 4,
```

```

    align = "center",
    checkboxInput("checkbox", "View other inputs", value = FALSE)
  ),
  column(
    width = 8,
    align = "center",
    conditionalPanel(
      condition = "input.checkbox == true",
      sliderInput("slider", "Select value", min = 1, max = 10, value = 5),
      textInput("txt", "Enter text", value = "")
    )
  )
),
server = function(input, output) {}
)

```

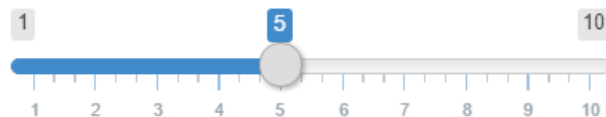
Condition FALSE

☐ View other inputs

Condition TRUE

☒ View other inputs

Select value



Enter text

12 Observe & fonctions d'update

12.1 Introduction

- Il existe une série de fonctions pour mettre à jour les inputs et certaines structures
- les fonctions commencent par `update...`
- On les utilise généralement à l'intérieur d'un `observe({expr})`
- La syntaxe est similaire à celle des fonctions de création
- **Attention** : il est nécessaire d'ajouter un argument `"session"` dans la définition du `server`

```
shinyServer(function(input, output, session) {...})
```

Sur des inputs :

- `updateCheckboxGroupInput`
- `updateCheckboxInput`
- `updateDateInput` `Change`
- `updateDateRangeInput`
- `updateNumericInput`
- `updateRadioButtons`

- `updateSelectInput`
- `updateSelectizeInput`
- `updateSliderInput`
- `updateTextInput`

Pour changer dynamiquement l'onglet sélectionné :

- `updateNavbarPage`, `updateNavlistPanel`, `updateTabsetPanel`

12.2 Exemple sur un input

```
shinyUI(fluidPage(
  titlePanel("Observe"),
  sidebarLayout(
    sidebarPanel(
      radioButtons(inputId = "id_dataset", label = "Choose a dataset", inline = TRUE,
                   choices = c("cars", "iris", "quakes"), selected = "cars"),
      selectInput("id_col", "Choose a column", choices = colnames(cars)),
      textOutput(outputId = "txt_obs")
    ),
    mainPanel(fluidRow(
      dataTableOutput(outputId = "dataset_obs")
    ))
  )
))
```

```
shinyServer(function(input, output, session) {
  dataset <- reactive(get(input$id_dataset, "package:datasets"))

  observe({
    updateSelectInput(session, inputId = "id_col", label = "Choose a column",
                      choices = colnames(dataset()))
  })

  output$txt_obs <- renderText(paste0("Selected column : ", input$id_col))

  output$dataset_obs <- renderDataTable(
    dataset(),
    options = list(pageLength = 5)
  )
})
```

Observer

Choose a dataset

☒ cars ☐ iris ☐ quakes

Choose a column

speed

speed

dist

Show 10 entries

Search:

speed	dist
4	2
4	10
7	4
7	22
8	16

speed

dist

Showing 1 to 5 of 50 entries

Previous

1

2

3

4

5

...

10

Next

Observer

Choose a dataset

☐ cars ☒ iris ☐ quakes

Choose a column

Sepal.Length

Sepal.Length

Sepal.Width

Petal.Length

Petal.Width

Species

Show 10 entries

Search:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

Sepal.Length

Sepal.Width

Petal.Length

Petal.Width

Species

Showing 1 to 5 of 150 entries

Previous

1

2

3

4

5

...

30

Next

12.3 Exemple sur des onglets

Il faut rajouter un id dans la structure

```
shinyUI(  
  navbarPage(  
    id = "idnavbar", # need an id for observe & update  
    title = "A NavBar",  
    tabPanel(title = "Summary",  
      actionButton("goPlot", "Go to plot !")),  
    tabPanel(title = "Plot",  
      actionButton("goSummary", "Go to Summary !"))  
  )  
)  
  
shinyServer(function(input, output, session) {  
  observe({  
    input$goPlot
```

```

    updateTabsetPanel(session, "idnavbar", selected = "Plot")
  })
  observe({
    input$goSummary
    updateTabsetPanel(session, "idnavbar", selected = "Summary")
  })
})

```

12.4 observeEvent

- Une variante de la fonction `observe` est disponible avec la fonction `observeEvent`
- On définit alors de façon explicite l'expression qui représente l'événement *et* l'expression qui sera exécutée quand l'événement se produit

```

# avec un observe
observe({
  input$goPlot
  updateTabsetPanel(session, "idnavbar", selected = "Plot")
})

# idem avec un observeEvent
observeEvent(input$goSummary, {
  updateTabsetPanel(session, "idnavbar", selected = "Summary")
})

```

13 HTML / CSS

13.1 Inclure du HTML

De nombreuses de balises **html** sont disponibles avec les fonctions `tags` :

```
names(shiny::tags)
```

##	[1]	"a"	"abbr"	"address"
##	[4]	"animate"	"animateMotion"	"animateTransform"
##	[7]	"area"	"article"	"aside"
##	[10]	"audio"	"b"	"base"
##	[13]	"bdi"	"bdo"	"blockquote"
##	[16]	"body"	"br"	"button"
##	[19]	"canvas"	"caption"	"circle"
##	[22]	"cite"	"clipPath"	"code"
##	[25]	"col"	"colgroup"	"color-profile"
##	[28]	"command"	"data"	"datalist"
##	[31]	"dd"	"defs"	"del"
##	[34]	"desc"	"details"	"dfn"
##	[37]	"dialog"	"discard"	"div"
##	[40]	"dl"	"dt"	"ellipse"
##	[43]	"em"	"embed"	"eventsSource"
##	[46]	"feBlend"	"feColorMatrix"	"feComponentTransfer"
##	[49]	"feComposite"	"feConvolveMatrix"	"feDiffuseLighting"
##	[52]	"feDisplacementMap"	"feDistantLight"	"feDropShadow"
##	[55]	"feFlood"	"feFuncA"	"feFuncB"
##	[58]	"feFuncG"	"feFuncR"	"feGaussianBlur"
##	[61]	"feImage"	"feMerge"	"feMergeNode"

## [64]	"feMorphology"	"feOffset"	"fePointLight"
## [67]	"feSpecularLighting"	"feSpotLight"	"feTile"
## [70]	"feTurbulence"	"fieldset"	"figcaption"
## [73]	"figure"	"filter"	"footer"
## [76]	"foreignObject"	"form"	"g"
## [79]	"h1"	"h2"	"h3"
## [82]	"h4"	"h5"	"h6"
## [85]	"hatch"	"hatchpath"	"head"
## [88]	"header"	"hgroup"	"hr"
## [91]	"html"	"i"	"iframe"
## [94]	"image"	"img"	"input"
## [97]	"ins"	"kbd"	"keygen"
## [100]	"label"	"legend"	"li"
## [103]	"line"	"linearGradient"	"link"
## [106]	"main"	"map"	"mark"
## [109]	"marker"	"mask"	"menu"
## [112]	"meta"	"metadata"	"meter"
## [115]	"mpath"	"nav"	"noscript"
## [118]	"object"	"ol"	"optgroup"
## [121]	"option"	"output"	"p"
## [124]	"param"	"path"	"pattern"
## [127]	"picture"	"polygon"	"polyline"
## [130]	"pre"	"progress"	"q"
## [133]	"radialGradient"	"rb"	"rect"
## [136]	"rp"	"rt"	"rtc"
## [139]	"ruby"	"s"	"samp"
## [142]	"script"	"section"	"select"
## [145]	"set"	"slot"	"small"
## [148]	"solidcolor"	"source"	"span"
## [151]	"stop"	"strong"	"style"
## [154]	"sub"	"summary"	"sup"
## [157]	"svg"	"switch"	"symbol"
## [160]	"table"	"tbody"	"td"
## [163]	"template"	"text"	"textarea"
## [166]	"textPath"	"tfoot"	"th"
## [169]	"thead"	"time"	"title"
## [172]	"tr"	"track"	"tspan"
## [175]	"u"	"ul"	"use"
## [178]	"var"	"video"	"view"
## [181]	"wbr"		

```
tags$a(href = "www.rstudio.com", "RStudio")
```

the list
named tags

the function/tag name
(followed by parentheses)

named arguments
appear as tag attributes
(set boolean attributes to NA)

unnamed arguments
appear inside the tags
(call tags\$...() to create nested tags)

```
<a href="www.rstudio.com">RStudio</a>
```

C'est également possible de passer du code **HTML** directement en utilisant la fonction du même nom :

```
fluidPage(  
  HTML("<h1>My Shiny App</h1>")  
)
```

13.2 Quelques balises utiles

- `div(..., align = "center")` : centrer les éléments
- `br()` : saut de ligne
- `hr()` : trait horizontal
- `img(src="img/logo.jpg", title="Popup", width = "80%")` : insertion d'une image présente dans **www/img**
- `a(href="https://r2018-rennes.sciencesconf.org/", target="_blank", "Rencontres R")` : lien vers un site
- `a(href = './doc/guide.pdf', target="_blank", class = "btn", icon("download"), 'Télécharger le guide utilisateur')` : lien de téléchargement d'un document présent dans **www/doc**

13.3 CSS : introduction

Shiny utilise Bootstrap pour la partie **CSS**.

Comme dans du développement web “classique”, nous pouvons modifier le **CSS** de trois façons :

- en faisant un lien vers un fichier .css externe, en ajoutant des feuilles de style dans le répertoire **www**
- en ajoutant du **CSS** dans le header **HTML**
- en écrivant individuellement du CSS aux éléments.

Il y a une notion d'ordre et de priorité sur ces trois informations : le **CSS** “individuel” l'emporte sur le **CSS** du header, qui l'emporte sur le **CSS** externe

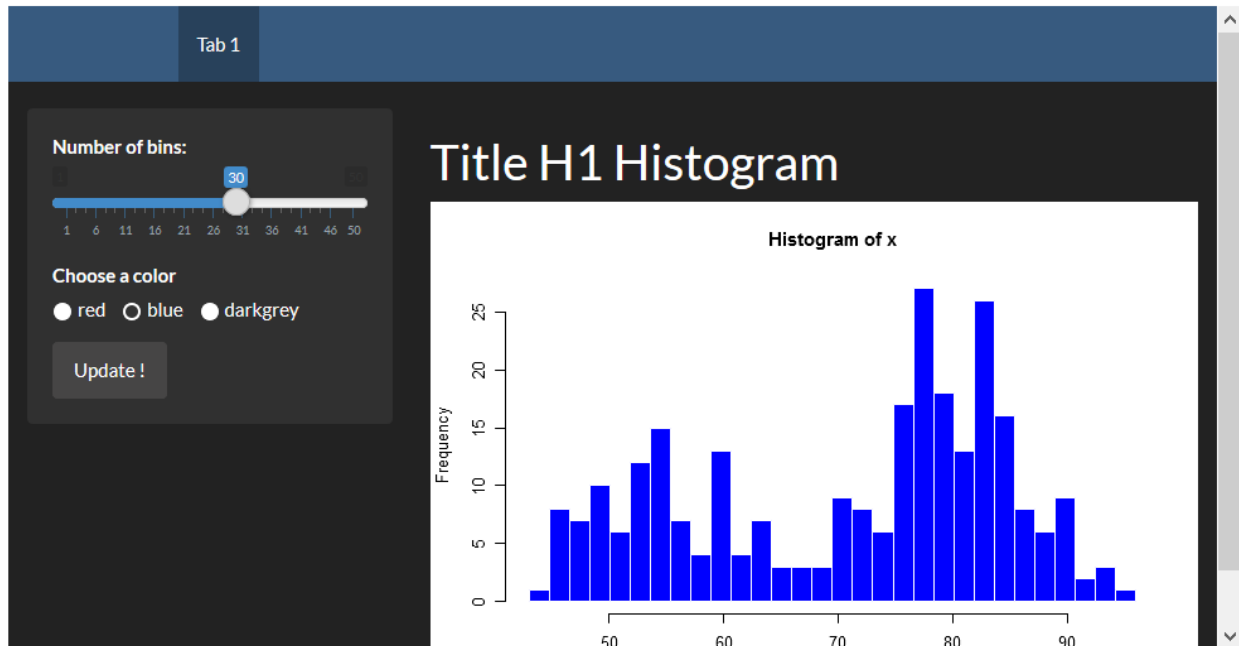
On peut aussi utiliser le package `shinythemes`

13.4 Avec un .css externe

On peut par exemple aller prendre un thème sur `bootswatch`.

- Deux façons pour le renseigner : + argument `theme` présent dans certaines fonctions (`fluidPage`, `navbarPage`, ...) + ou avec un tags html : `tags$head` et `tags$link`

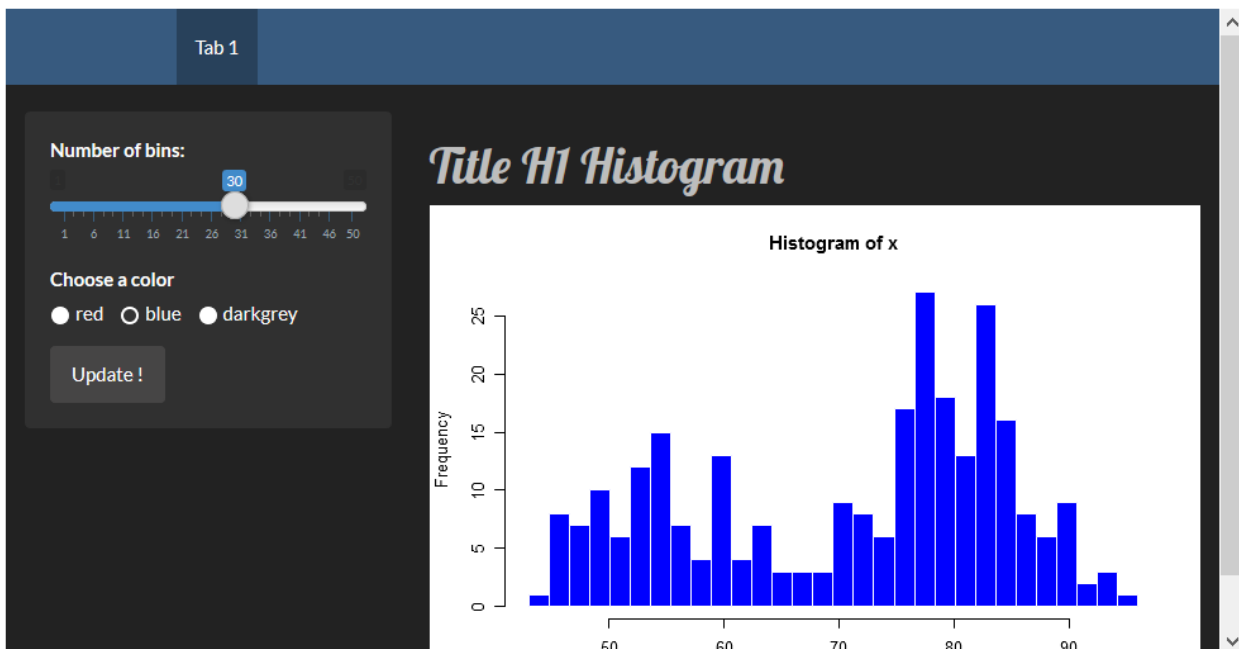
```
library(shiny)  
ui <- fluidPage(theme = "mytheme.css",  
  # ou avec un tags  
  tags$head(  
    tags$link(rel = "stylesheet", type = "text/css", href = "mytheme.css")  
  ),  
  # reste de l'application  
)
```



13.5 Ajout de css dans le header

- Le **CSS** inclut dans le header sera prioritaire au **CSS** externe
- inclusion avec les tags html : `tags$head` et `tags$style`

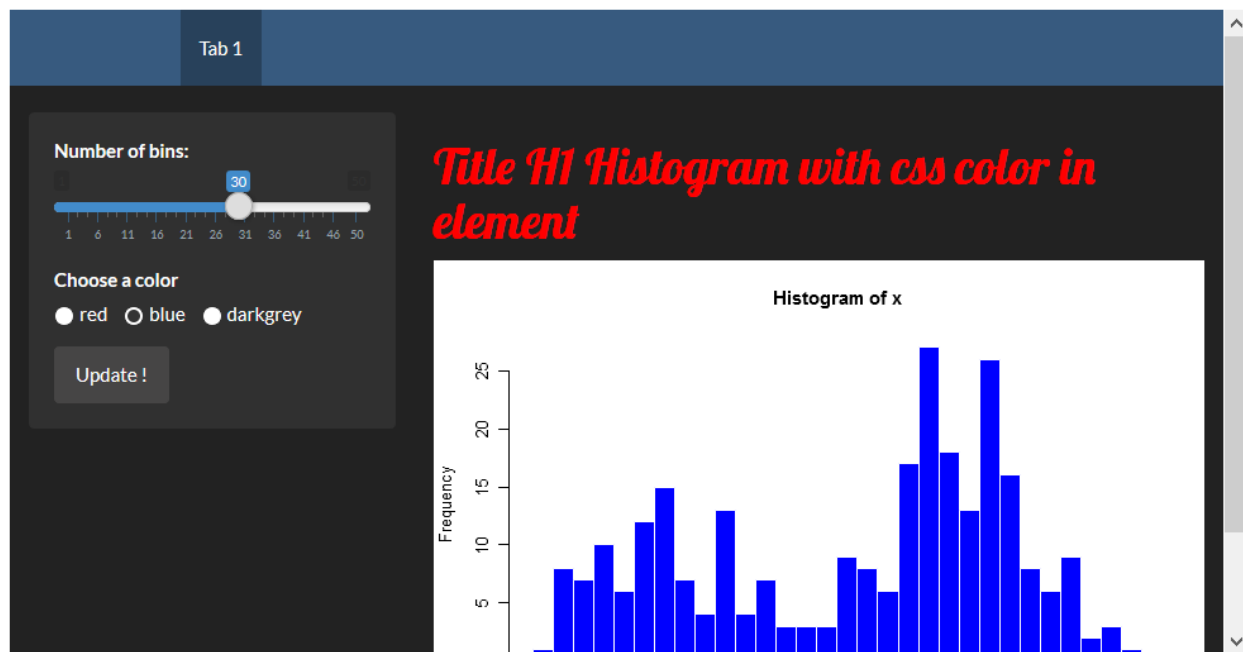
```
library(shiny)
tags$head(
  tags$style(HTML("h1 { color: #48ca3b;}"))
),
# reste de l'application
)
```



13.6 CSS sur un élément

Pour finir, on peut également passer directement du **CSS** aux éléments **HTML** :

```
library(shiny)
h1("Mon titre", style = "color: #48ca3b;")
# reste de l'application
)
```



14 Quelques bonnes pratiques

- Préférer l'underscore (`_`) au point (`.`) comme séparateur dans le nom des variables. En effet, le `.` peut amener de mauvaises interactions avec d'autres langages, comme le **JavaScript**
- Faire bien attention à l'**unicité des différents identifiants** des inputs/outputs
- Pour éviter des problèmes éventuels avec **des versions différentes de packages**, et notamment dans le cas de **plusieurs applications shiny** et/ou différents environnements de travail, essayer d'utiliser **packrat**
- Mettre toute la **partie "calcul"** dans des **fonctions/un package** et effectuer des tests (`testthat`)
- Diviser la partie **ui.R** et **server.R** en plusieurs scripts, un par onglet par exemple :

```
# ui.R
shinyUI(
  navbarPage("Divide UI & SERVER",
    source("src/ui/01_ui_plot.R", local = TRUE)$value,
    source("src/ui/02_ui_data.R", local = TRUE)$value
  )
)

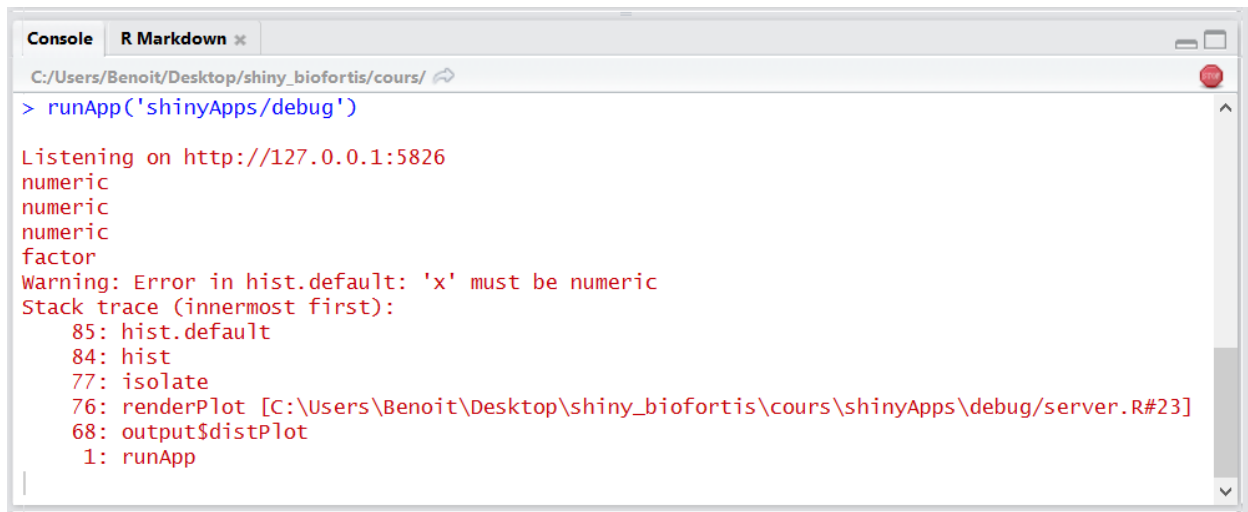
# server.R
shinyServer(function(input, output, session) {
  source("src/server/01_server_plot.R", local = TRUE)
  source("src/server/02_server_data.R", local = TRUE)
})
```

15 Débogage

15.1 Affichage console

- Un des premiers niveaux de débogage est l'utilisation de `print` console au-sein de l'application shiny.
- Cela permet d'afficher des informations lors du développement et/ou de l'exécution de l'application
- Dans **shiny**, on utilisera de préférence `cat(file=stderr(), ...)` pour être sûr que l'affichage marche dans tous les cas d'outputs, et également dans les logs avec **shiny-server**

```
output$distPlot <- renderPlot({  
  x <- iris[, input$variable]  
  cat(file=stderr(), class(x)) # affichage de la classe de x  
  hist(x)  
})
```



15.2 Lancement manuel d'un browser

- On peut insérer le lancement d'un `browser()` à n'importe quel moment
- On pourra alors observer les différents objets et avancer pas-à-pas

```
output$distPlot <- renderPlot({  
  x <- iris[, input$variable]  
  browser() # lancement du browser  
  hist(x)  
})
```

- Ne pas oublier de l'enlever une fois le développement terminé...!


```

Console R Markdown x
C:/Users/Benoit/Desktop/shiny_biofortis/cours/shinyApps/debug/
Next Previous Run Continue Stop
Browse[1]> ls()
[1] "x"
Browse[1]> head(x)
[1] setosa setosa setosa setosa setosa setosa
Levels: setosa versicolor virginica
Browse[1]> class(x)
[1] "factor"
Browse[1]>

```

15.3 Lancement automatique d'un browser

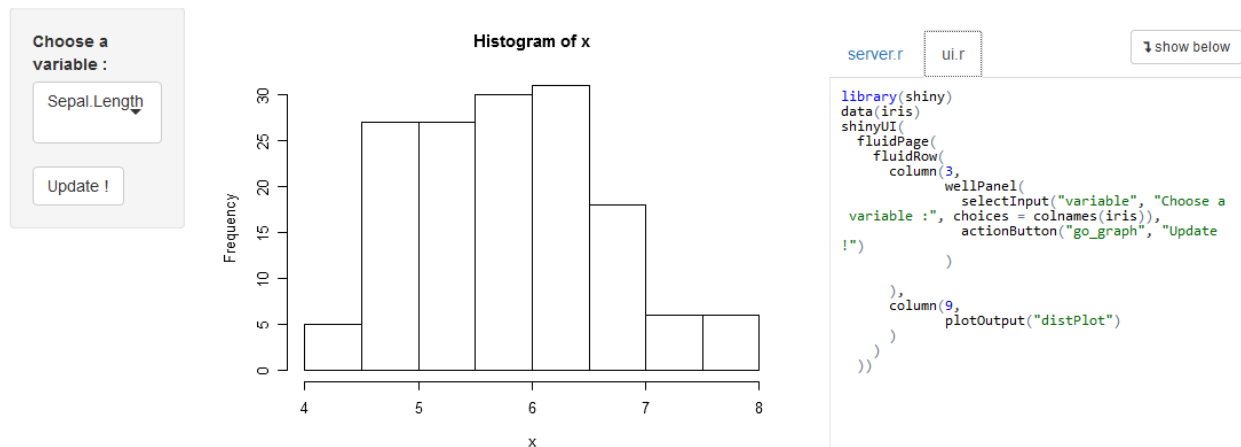
- L'option `options(shiny.error = browser)` permet de lancer un `browser()` automatiquement lors de l'apparition d'une erreur

```
options(shiny.error = browser)
```

15.4 Mode "showcase"

- En lançant une application avec l'option `display.mode="showcase"` et l'utilisation de la fonction `runApp()`, on peut observer en direct l'exécution du code :

```
runApp("path/to/myapp", display.mode="showcase")
```



15.5 Reactive log

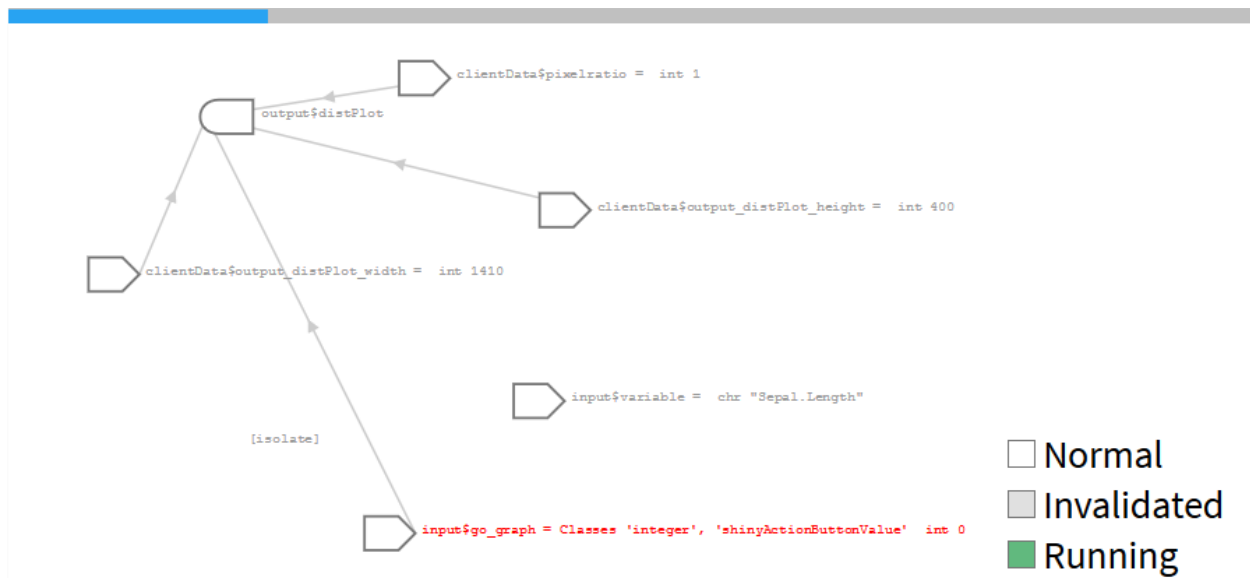
- En activant l'option `shiny.reactlog`, on peut visualiser à tous instants les dépendances et les flux entre les objets réactifs de **shiny**
- soit en tapant `ctrl+F3` dans le navigateur web
- soit en insérant `showReactLog()` au-sein du code shiny

```
options(shiny.reactlog=TRUE)
```

```

output$distPlot <- renderPlot({
  x <- iris[, input$variable]
  showReactLog() # launch shiny.reactlog
  hist(x)
})

```



15.6 Communication client/server

- Toutes les communications entre le client et le server sont visibles en utilisant l'option `shiny.trace`

```
options(shiny.trace = TRUE)
```

```

Console  R Markdown x
C:/Users/Benoit/Desktop/shiny_biofortis/cours/
> runApp('shinyApps/debug')

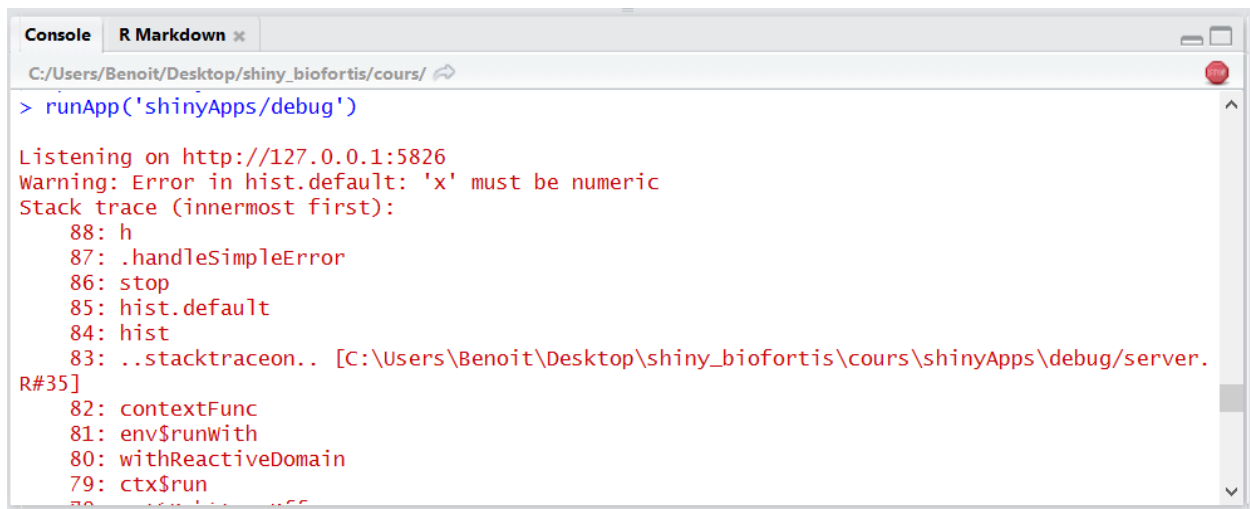
Listening on http://127.0.0.1:5826
SEND {"config":{"workerId":"","sessionId":"d881eec9a56887dd66d5d6bf2f8776ed"}}
RECV {"method":"init","data":{"go_graph:shiny.action":0,"variable":"Sepal.Length","clientdata_
output_distPlot_width":816,".clientdata_output_distPlot_height":400,".clientdata_output_distPlo
t_hidden":false,".clientdata_pixelratio":1,".clientdata_url_protocol":"http:",".clientdata_url_
hostname":"127.0.0.1",".clientdata_url_port":"5826",".clientdata_url_pathname":"/",".clientdata_
url_search":"",".clientdata_url_hash_initial":"",".clientdata_singletons":"",".clientdata_allo
wDataUriScheme":true}}
SEND {"custom":{"busy":"busy"}}
SEND {"custom":{"recalculating":{"name":"distPlot","status":"recalculating"}}}
SEND {"custom":{"recalculating":{"name":"distPlot","status":"recalculated"}}}
SEND {"custom":{"busy":"idle"}}
SEND {"errors":[],"values":{"distPlot":{"src":"data:image/png;base64 data","width":816,"height":400,"coordmap":[{"domain":{"left":3.84,"right":8.16,"bottom":-1.24,"top":32.24},"range":{"left":59.04,"right":785.76,"bottom":325.56,"top":58.04},"log":{"x":null,"y":null},"mapping":{}}]},"inputMessages":[]}}
RECV {"method":"update","data":{"variable":"Petal.Length"}}

```

15.7 Traçage des erreurs

- Depuis shiny_0.13.1, on récupère la stack trace quand une erreur se produit
- Si besoin, on peut récupérer une stack trace encore plus complète, comprenant les différents fonctions internes, avec `options(shiny.fullstacktrace = TRUE)`

```
options(shiny.fullstacktrace = TRUE)
```



The screenshot shows an R console window with the following content:

```
Console R Markdown x
C:/Users/Benoit/Desktop/shiny_biofortis/cours/
> runApp('shinyApps/debug')

Listening on http://127.0.0.1:5826
Warning: Error in hist.default: 'x' must be numeric
Stack trace (innermost first):
 88: h
 87: .handleSimpleError
 86: stop
 85: hist.default
 84: hist
 83: ..stacktraceon.. [C:\Users\Benoit\Desktop\shiny_biofortis\cours\shinyApps\debug\server.R#35]
 82: contextFunc
 81: env$runWith
 80: withReactiveDomain
 79: ctx$run
 78: ...
```

15.8 Références / Tutoriaux / Exemples

- <http://shiny.rstudio.com/>
- <http://shiny.rstudio.com/articles/>
- <http://shiny.rstudio.com/tutorial/>
- <http://shiny.rstudio.com/gallery/>
- <https://www.rstudio.com/products/shiny/shiny-user-showcase/>
- <http://www.showmeshiny.com/>