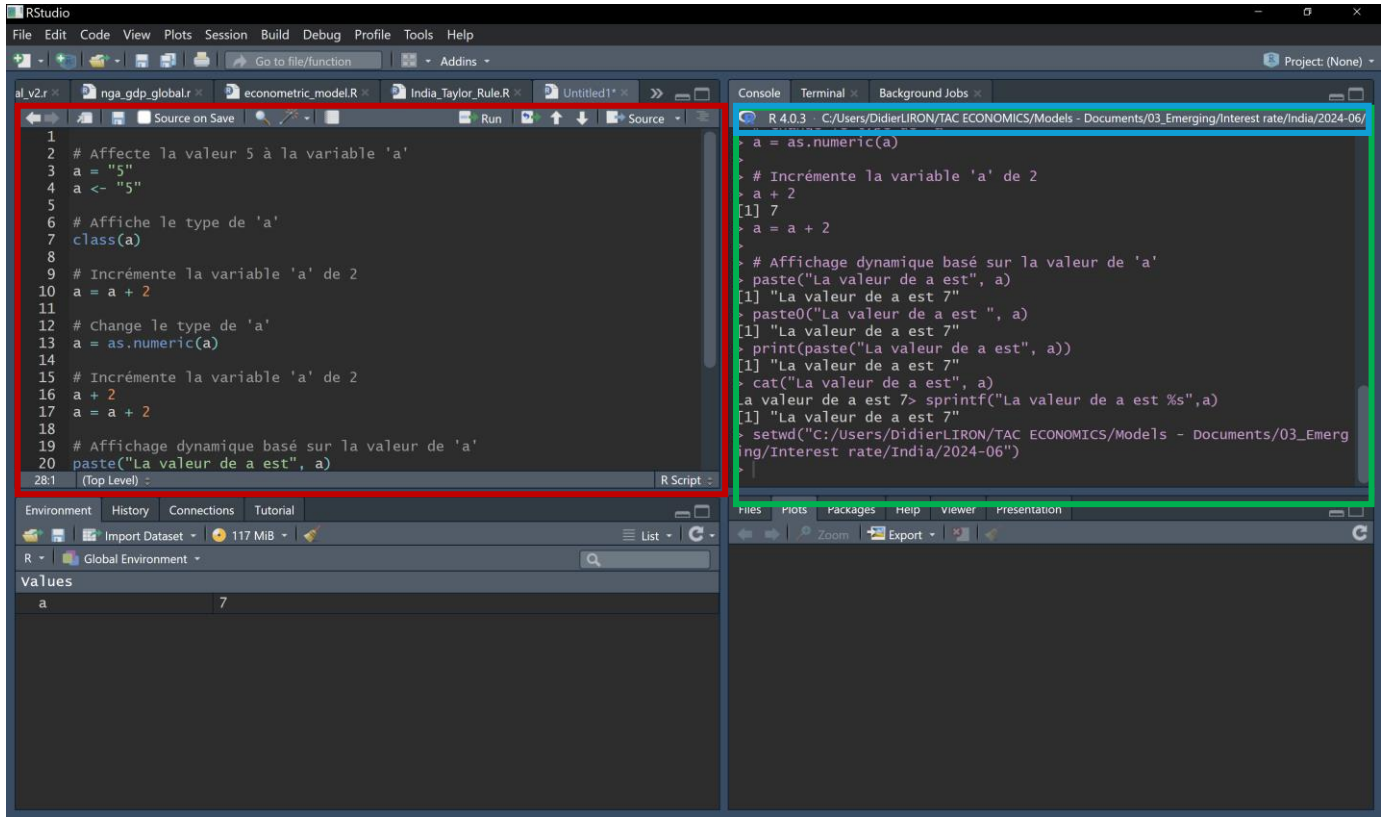
Introduction à R et Python

M1 IEF - 2024

TAC ECONOMICS

1. Qu'est ce que R ?
2. Qu'est ce que Rstudio ?
3. Différence **script** vs **console**
4. Créer un environnement sain de développement pour un projet
5. Qu'est ce qu'un **repertoire de travail** ?
6. Qu'est ce qu'une librairie (et le CRAN) ?



Plan

01 Les variables

02 Les structures de contrôle

03 Les fonctions

04 Manipuler les dataframes

05 Les séries temporelles

06 Les APIs

1. Les types
   - Chaines de caractères: **character**
   - Numériques: **numeric**
   - Booleens: **logical**
   - Vide: **NULL**

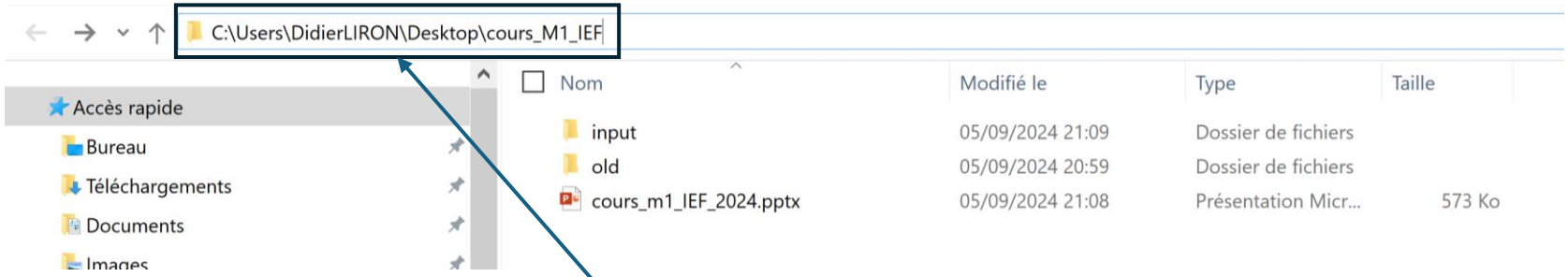
2. Les vecteurs: c(1, 2, « Hello »)

3. Afficher un résultat: **paste**, **paste0**, **print**, **cat**, **sprintf**

4. Importer un fichier
   - Chemin relatif vs absolue (./, ../)
   - csv, xlsx
   - Vérifier les types des colonnes

```
# Affecte la valeur 5 à la variable 'a'
a = "5"
a <- "5"

# Affiche le type de 'a'
class(a)

# Incrémente la variable 'a' de 2
a = a + 2

# Change le type de 'a'
a = as.numeric(a)

# Incrémente la variable 'a' de 2
a + 2
a = a + 2

# Affichage dynamique basé sur la valeur de 'a'
paste("La valeur de a est", a)
paste0("La valeur de a est ", a)
print(paste("La valeur de a est", a))
cat("La valeur de a est", a)
sprintf("La valeur de a est %s",a)
```

Etant donné un repertoire de travail fixé à cet endroit :



Chemin absolu : « C:\Users\DidierLIRON\Desktop\cours_M1_IEF »
Chemin relatif : « . »

Pour aller chercher un fichier dans le repertoire input :

./input/[monFichier]

→ Pointe sur le repertoire **courant**

Pour allercherche un fichier dans le repertoire précédent (Desktop):

../[monFichier]

→ Pointe sur le repertoire **précédent**

```
# Fixe le repertoire de travail
setwd(« C:\Users\DidierLIRON\Desktop\cours_M1_IEF »)

# Lire le fichier data.csv situé dans le repertoire input
# Chemin absolu
read.csv(« C:\Users\DidierLIRON\Desktop\cours_M1_IEF
/input/data.csv » )
# Chemin relatif
read.csv(« ./input/data.csv »)

# Lire le fichier dataprev.csv situé dans le repertoire
Desktop
# Chemin absolu
read.csv(« C:\Users\DidierLIRON\Desktop\dataPrev.csv » )
# Chemin relatif
read.csv(« ../dataPrev.csv »)
```

Fichier data.csv:

Separateur de colonnes « , »
Separateur de décimales « . »

```
# Importer le fichier csv
read.csv(« data.csv », sep=« , », dec= « . »)
```

Notepad++

```
"date","usgdp","sp500","usbond10","uscpi","tusgdp","usinfl"
1947-01-01,2034.45,NA,NA,21.7,NA,NA
1947-04-01,2029.024,NA,NA,22.01,NA,NA
1947-07-01,2024.834,NA,NA,22.49,NA,NA
1947-10-01,2056.508,NA,NA,23.1266666666667,NA,NA
1948-01-01,2087.442,NA,NA,23.6166666666667,2.60473346604733,8.83256528417819
1948-04-01,2121.899,NA,NA,23.9933333333333,4.57732387591276,9.01105558079662
1948-07-01,2134.056,NA,NA,24.3966666666667,5.39412119709566,8.47784200385355
1948-10-01,2136.44,NA,NA,24.1733333333333,3.88678283770354,4.5257999423465
1949-01-01,2107.001,NA,NA,23.9433333333333,0.936984117403039,1.3832039520113
1949-04-01,2099.814,NA,NA,23.9166666666667,-1.04081296989159,-0.319533203667677
1949-07-01,2121.493,NA,NA,23.7166666666667,-0.588691205854019,-2.78726601994808
1949-10-01,2103.688,NA,NA,23.66,-1.53301754320271,-2.12355212355212
1950-01-01,2186.365,NA,NA,23.5866666666667,3.76668069924977,-1.48962828901574
1950-04-01,2253.045,NA,NA,23.7666666666667,7.29736062336952,-0.627177700348436
1950-07-01,2340.112,NA,NA,24.2033333333333,10.3049597618281,2.05200281096276
1950-10-01,2384.92,NA,NA,24.6933333333333,13.3685223284061,4.36742744435052
```

Excel

date	usgdp	sp500	usbond10	uscpi	tusgdp	usinfl
01/01/1947	2034.45	NA	NA	21.7	NA	NA
01/04/1947	2029.024	NA	NA	22.01	NA	NA
01/07/1947	2024.834	NA	NA	22.49	NA	NA
01/10/1947	2056.508	NA	NA	23.1266667	NA	NA
01/01/1948	2087.442	NA	NA	23.6166667	2.60473347	8.83256528
01/04/1948	2121.899	NA	NA	23.9933333	4.57732388	9.01105558
01/07/1948	2134.056	NA	NA	24.3966667	5.3941212	8.477842
01/10/1948	2136.44	NA	NA	24.1733333	3.88678284	4.52579994
01/01/1949	2107.001	NA	NA	23.9433333	0.93698412	1.38320395
01/04/1949	2099.814	NA	NA	23.9166667	-1.040813	-0.3195332
01/07/1949	2121.493	NA	NA	23.7166667	-0.5886912	-2.787266
01/10/1949	2103.688	NA	NA	23.66	-1.5330175	-2.1235521
01/01/1950	2186.365	NA	NA	23.5866667	3.7666807	-1.4896283
01/04/1950	2253.045	NA	NA	23.7666667	7.29736062	-0.6271777
01/07/1950	2340.112	NA	NA	24.2033333	10.3049598	2.05200281
01/10/1950	2384.92	NA	NA	24.6933333	13.3685223	4.36742744

1. Les boucles
   - for : parcours tous les éléments d'un vecteur et execute autant de fois un code donné

```
for(i in vecteur) {
    code
}
```

   - while : Exécute un codé donné tant qu'une condition n'est pas remplie

```
while(condition){
    code
}
```

2. Les conditions: execute ou non un code en fonction d'une condition

```
if (condition) {
    code
} else if {
    code
} else {
    code
}
```

```
# parcours le vecteur c(1,2,3,4) et affiche les valeurs
for(i in c(1,2,3,4)) {
  print(i)
}
# parcours un autre vecteur et affiche les valeurs
for(i in c("Bonjour","tout","le","monde","!")) {
  print(i)
}

# Incrémente i de 1 jusqu'a ce que i vaille 5
i = 1
while( i < 5) {
  print(i)
  i = i + 1
}

# Affichage des valeurs de a sous certaines conditions
a = 7

if(a == 7) {
  print("a = 7")
} else if(a == 10) {
  print("a = 10")
} else {
  print("a n'est ni égal à 7 ni à 10.")
}
```

a	b	a ET b
F	F	F
F	V	F
V	F	F
V	V	V

a	b	a OU b
F	F	F
F	V	V
V	F	V
V	V	V

A	B	C	A ET B	A ET C	(A ET B) OU (A ET C)
F	F	F	F	F	F
F	F	V	F	F	F
F	V	F	F	F	F
F	V	V	F	F	F
V	F	F	F	F	F
V	F	V	F	V	V
V	V	F	V	F	V
V	V	V	V	V	V

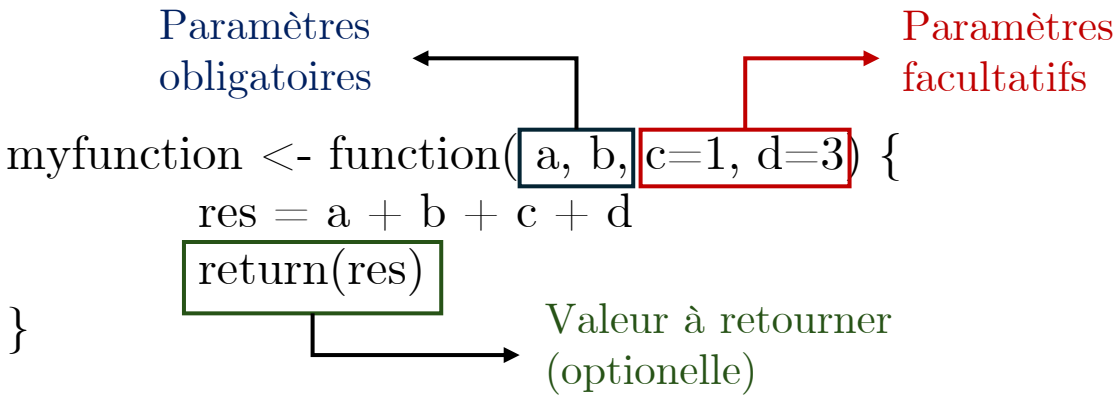
```
# Quelques exemples R des combinaisons de booléens

> TRUE & TRUE
[1] TRUE

> TRUE & FALSE
[1] FALSE

> TRUE | TRUE
[1] TRUE

> TRUE | FALSE
[1] TRUE

> (TRUE & FALSE) | FALSE
[1] TRUE
```

1. Vérifier si un nombre est premier ou pas. Pour savoir si un nombre est divisible par un autre, utilisez le modulo (%% en R)

2. Les deux premiers termes de la suite de Fibonacci sont tous deux un. Les termes suivants de la séquence sont trouvés en additionnant les deux termes immédiatement précédents. Écrivez un code qui écrit les n termes de la séquence de Fibonacci pour tout $n \geq 3$

3. Écrivez un code qui simule le lancer d'une pièce avec une probabilité p associée aux piles, jusqu'à obtenir fâce, et renvoie le nombre de piles obtenues. Ensuite, reproduisez l'exécution de ce code 100 fois et afficher le nombre maximum de piles obtenus lors d'une même séquence. Pour générer une probabilité entre 0 et 1, utilisez runif(1).

Une fonction informatique est un ensemble d'instructions regroupées sous un nom et s'exécutant à la demande (l'appel de la fonction).



Toutes les variables contenues dans une fonction (paramètres inclus) sont 'locales' et ne sont accessibles que dans cette fonction. Elles sont créées à l'appel de la fonction et detruites à la fin de son exécution. Par opposition aux variables déclarées partout ailleurs dans le code, dites 'globales' et accessibles partout (y compris dans une fonction).

```
# Déclaration d'une fonction
myfunction <- function(a, b, c = 1, d = 3) {
 res = a + b + c + d
 return(res)
}

# Différents appels à ma fonction
myfunction(a = 1, b = 2)
myfunction(1, 2)
xx = myfunction(a = 1, c = 2, b = 3)

# Illustrations variables locales/globales
res = 2
myfunction(1, 2)
print(res)

myfunction <- function(a, b, c = 1, d = 3) {
 res <<- a + b + c + d
 return(res)
}
res = 2
myfunction(1, 2)
print(res)
```

1. Vérifier si un nombre est premier ou pas. Pour savoir si un nombre est divisible par un autre, utilisez le modulo (%% en R)

   => Transformer votre réponse en utilisant une fonction avec un seul parametre **obligatoire** définissant le nombre à vérifier.

2. Les deux premiers termes de la suite de Fibonacci sont tous deux un. Les termes suivants de la séquence sont trouvés en additionnant les deux termes immédiatement précédents. Écrivez un code qui écrit les n termes de la séquence de Fibonacci pour tout $n \geq 3$

   => Transformer votre réponse en utilisant une fonction avec un seul parametre **obligatoire** n définissant le nombre de termes à ajouter à la suite de Fibonacci.

3. Écrivez un code qui simule le lancer d'une pièce avec une probabilité p associée aux piles, jusqu'à obtenir fâce, et renvoie le nombre de piles obtenues. Ensuite, reproduisez l'exécution de ce code 100 fois et afficher le nombre maximum de piles obtenus lors d'une même séquence. Pour générer une probabilité entre 0 et 1, utilisez runif(1).

   => Transformer votre réponse en utilisant une fonction avec deux paramètres **facultatifs**. Un premier pour définir la probabilité associée aux piles (par défaut 0.5), et un second pour le nombre de lancers (par défaut 100).