Logiciel **R** — apprentissage de la programmation

Hervé Seitz (email : herve.seitz@igh.cnrs.fr)

IGH (UMR 9002 CNRS et université de Montpellier)

16 novembre 2022

Ce diaporama est accessible à :
http://www.igh.cnrs.fr/equip/Seitz/Rprog.pdf

La programmation

La programmation

Une série d'instructions à exécuter :

La programmation

Une série d'instructions à exécuter :

- en automate, de manière insensible aux conditions ;

La programmation

Une série d'instructions à exécuter :

- en automate, de manière insensible aux conditions ;
- de manière adaptée aux *inputs*.

La programmation

Une série d'instructions à exécuter :

- en automate, de manière insensible aux conditions ;
- de manière adaptée aux *inputs*.

Un ≪ algorithme ≫ : une série d'instructions.

La programmation

Une série d'instructions à exécuter :

- en automate, de manière insensible aux conditions ;
- de manière adaptée aux *inputs*.

Un « algorithme » : une série d'instructions (ex. : « Suivre la rue jusqu'au carrefour ; si le feu est rouge ou orange, s'arrêter. Quand le feu est vert, avancer. Au carrefour suivant, tourner à droite ; ... »).

La programmation

Une série d'instructions à exécuter :

- en automate, de manière insensible aux conditions ;
- de manière adaptée aux *inputs*.

Un « algorithme » : une série d'instructions (ex. : « Suivre la rue jusqu'au carrefour ; si le feu est rouge ou orange, s'arrêter. Quand le feu est vert, avancer. Au carrefour suivant, tourner à droite ; ... »).

L'implémentation d'un algorithme : sa formulation de manière compréhensible par l'ordinateur (implique le choix d'un langage de programmation).

Structures de contrôle

Structures de contrôle

Le test *if* : conditionne une commande à un booléen.

Structures de contrôle

Le test *if* : conditionne une commande à un booléen.

```
> a=3
> if (a<2) print("a est plus petit que 2")
>
```

Structures de contrôle

Le test *if* : conditionne une commande à un booléen.

```
> a=3
> if (a<2) print("a est plus petit que 2")
> if (a>2) print("a est plus grand que 2")
[1] "a est plus grand que 2"
>
```

Structures de contrôle

```
> x=c(3.4,2.1,5.0,3.6,4.1)
> y=c(5.4,4.2,6.0,5.3)
> p=leveneTest(c(x,y),as.factor(c(rep('Condition_x',times=length(x)),rep('Condition_y',times=len
gth(y)))))$Pr[1]
> if (p<0.05)
+ {
+ print("Heterogeneous variances")
+ t.test(x,y,var.equal=FALSE)
+ } else
+ {
+ print("Homogeneous variances")
+ t.test(x,y,var.equal=TRUE)
+ }
[1] "Homogeneous variances"

        Two Sample t-test

data:  x and y
t = -2.5148, df = 7, p-value = 0.04012
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.07537667 -0.09462333
sample estimates:
mean of x mean of y
    3.640     5.225

>
```


Structures de contrôle

```
> x=c(3.4,2.1,5.0,3.6,4.1)
> y=c(5.4,4.2,6.0,5.3)
> p=leveneTest(c(x,y),as.factor(c(rep('Condition_x',times=length(x)),rep('Condition_y',times=len
gth(y)))))$Pr[1]
> if (p<0.05)
+ {
+ print("Heterogeneous variances")
+ t.test(x,y,var.equal=FALSE)
+ } else
+ {
+ print("Homogeneous variances")
+ t.test(x,y,var.equal=TRUE)
+ }
[1] "Homogeneous variances"

        Two Sample t-test

data:  x and y
t = -2.5148, df = 7, p-value = 0.04012
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.07537667 -0.09462333
sample estimates:
mean of x mean of y
    3.640     5.225

>
```

Les accolades permettent de conditionner plusieurs commandes au *if* (par défaut : seule la première commande est concernée).

Structures de contrôle

Les boucles *for* et *while* : répéter une série d'instructions.

Structures de contrôle

Les boucles *for* et *while* : répéter une série d'instructions.

```
> for (i in 1:10)
+ print(paste("now i =",i))
[1] "now i = 1"
[1] "now i = 2"
[1] "now i = 3"
[1] "now i = 4"
[1] "now i = 5"
[1] "now i = 6"
[1] "now i = 7"
[1] "now i = 8"
[1] "now i = 9"
[1] "now i = 10"
>
```

Structures de contrôle

Les boucles *for* et *while* : répéter une série d'instructions.

```
> r=1
> while (r<100)
+ {
+ r=2*r
+ print(paste("now r =",r))
+ }
[1] "now r = 2"
[1] "now r = 4"
[1] "now r = 8"
[1] "now r = 16"
[1] "now r = 32"
[1] "now r = 64"
[1] "now r = 128"
>
```

Structures de contrôle

Les boucles *for* et *while* : répéter une série d'instructions.

```
> r=1
> while (r<100)
+ {
+ r=2*r
+ print(paste("now r =",r))
+ }
[1] "now r = 2"
[1] "now r = 4"
[1] "now r = 8"
[1] "now r = 16"
[1] "now r = 32"
[1] "now r = 64"
[1] "now r = 128"
>
```

Attention ! Pour les opérations à effectuer sur chaque élément d'un vecteur, il est plus efficace d'appliquer l'opération à tout le vecteur, que de faire une boucle *for* pour le parcourir.

Structures de contrôle

Exercice : écrire une série d'instructions qui affiche, pour tous les entiers entre 0 et 30, ceux dont le cosinus est supérieur à 0,5.

Structures de contrôle

Exercice : écrire une série d'instructions qui affiche, pour tous les entiers entre 0 et 30, ceux dont le cosinus est supérieur à 0,5.

Possibilité n°1 : avec une boucle *for* sur une variable qui décrira tous ces entiers, comparer le cosinus de la variable à 0,5 et afficher la valeur de la variable en fonction de ce résultat.

Structures de contrôle

Exercice : écrire une série d'instructions qui affiche, pour tous les entiers entre 0 et 30, ceux dont le cosinus est supérieur à 0,5.

```
> for (i in 0:30)
+ if (cos(i)>0.5) print(i)
```

Structures de contrôle

Exercice : écrire une série d'instructions qui affiche, pour tous les entiers entre 0 et 30, ceux dont le cosinus est supérieur à 0,5.

```
> for (i in 0:30)
+ if (cos(i)>0.5) print(i)
[1] 0
[1] 1
[1] 6
[1] 7
[1] 12
[1] 13
[1] 18
[1] 19
[1] 25
[1] 26
>
```


Structures de contrôle

Exercice : écrire une série d'instructions qui affiche, pour tous les entiers entre 0 et 30, ceux dont le cosinus est supérieur à 0,5.

Possibilité n°2 : appliquer au vecteur *c(0:30)* la condition sur son cosinus (*i.e.* : définir un vecteur de booléens qui vaudra *TRUE* lorsque *cos(i)* sera supérieur à 0,5).

Structures de contrôle

Exercice : écrire une série d'instructions qui affiche, pour tous les entiers entre 0 et 30, ceux dont le cosinus est supérieur à 0,5.

```
> cos(c(0:30))>0.5
 [1]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
[13]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
[25] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
>
```

Structures de contrôle

Exercice : écrire une série d'instructions qui affiche, pour tous les entiers entre 0 et 30, ceux dont le cosinus est supérieur à 0,5.

```
> cos(c(0:30))>0.5
 [1]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
[13]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
[25] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
> c(0:30)[cos(c(0:30))>0.5]
 [1]  0  1  6  7 12 13 18 19 25 26
>
```

Structures de contrôle

Exercice : Dans la suite géométrique $(1\,;\,1/2\,;\,1/4\,;\,1/8\,;\,\ldots)$, afficher le premier terme inférieur à 0,001.

Structures de contrôle

Exercice : Dans la suite géométrique (1 ; 1/2 ; 1/4 ; 1/8 ; ...), afficher le premier terme inférieur à 0,001.

Avec une boucle *while* qui divise une variable par deux jusqu'à atteindre 0,001.

Structures de contrôle

Exercice : Dans la suite géométrique (1 ; 1/2 ; 1/4 ; 1/8 ; ...), afficher le premier terme inférieur à 0,001.

```
> x=1
> while(x>0.001)
+ x=x/2
> print(x)
[1] 0.0009765625
>
```


Créer ses propres fonctions

La commande *function* définit une nouvelle fonction, et le nombre d'arguments qu'elle prendra.

Créer ses propres fonctions

La commande *function* définit une nouvelle fonction, et le nombre d'arguments qu'elle prendra.

```
> somme_carres=function(x,y)
+ {
+ return(x^2+y^2)
+ }
>
```

Créer ses propres fonctions

La commande *function* définit une nouvelle fonction, et le nombre d'arguments qu'elle prendra.

```
> somme_carres(2,3)
[1] 13
>
```

Créer ses propres fonctions

La commande *function* définit une nouvelle fonction, et le nombre d'arguments qu'elle prendra.

```
> somme_carres(2,3,4)
Error in somme_carres(2, 3, 4) : unused argument (4)
>
```

Créer ses propres fonctions

La commande *function* définit une nouvelle fonction, et le nombre d'arguments qu'elle prendra.

```
> somme_carres(2)
Error in somme_carres(2) : argument "y" is missing, with no default
>
```

Créer ses propres fonctions

La commande *function* définit une nouvelle fonction, et le nombre d'arguments qu'elle prendra.

```
> somme_carres(2)
Error in somme_carres(2) : argument "y" is missing, with no default
>
```

Exercice : écrire une fonction qui calcule la différence entre la moyenne et la médiane d'un vecteur de valeurs numériques donné en entrée, et qui la retourne au programme principal.

Créer ses propres fonctions

La commande *function* définit une nouvelle fonction, et le nombre d'arguments qu'elle prendra.

```
> somme_carres(2)
Error in somme_carres(2) : argument "y" is missing, with no default
>
```

Exercice : écrire une fonction qui calcule la différence entre la moyenne et la médiane d'un vecteur de valeurs numériques donné en entrée, et qui la retourne au programme principal.

Exercice : écrire une fonction qui retourne un booléen (*TRUE* si les valeurs du vecteur donné en entrée semblent tirées d'une distribution normale, *FALSE* sinon).

Bonnes pratiques

Bonnes pratiques

Les objectifs :

Bonnes pratiques

Les objectifs :

- Rendre l'utilisation du programme flexible (pas spécifique à un fichier d'*input* particulier).

Bonnes pratiques

Les objectifs :

- Rendre l'utilisation du programme flexible (pas spécifique à un fichier d'*input* particulier).
- Rendre le programme facile à modifier si un paramètre doit être ajusté.

Bonnes pratiques

Les objectifs :

- Rendre l'utilisation du programme flexible (pas spécifique à un fichier d'*input* particulier).
- Rendre le programme facile à modifier si un paramètre doit être ajusté.
- Rendre le programme facile à débugger.

Bonnes pratiques

Les objectifs :

- Rendre l'utilisation du programme flexible (pas spécifique à un fichier d'*input* particulier).
- Rendre le programme facile à modifier si un paramètre doit être ajusté.
- Rendre le programme facile à débugger.
- Rendre le programme facilement compréhensible au lecteur (lecteur = quelqu'un d'autre, ou alors ... vous-même dans le futur !).

Bonnes pratiques

Bonnes pratiques

Le principe :

- éviter au maximum d'entrer littéralement des valeurs numériques (plutôt utiliser des variables : facilement modifiables) ;

Bonnes pratiques

Le principe :

- éviter au maximum d'entrer littéralement des valeurs numériques (plutôt utiliser des variables : facilement modifiables) ;
- éviter de copier-coller des blocs de commandes quand il s'agit d'appliquer les mêmes opérations à différents *inputs* (plutôt utiliser des boucles ou des fonctions : le débuggage est plus efficace) ;

Bonnes pratiques

Le principe :

- éviter au maximum d'entrer littéralement des valeurs numériques (plutôt utiliser des variables : facilement modifiables) ;
- éviter de copier-coller des blocs de commandes quand il s'agit d'appliquer les mêmes opérations à différents *inputs* (plutôt utiliser des boucles ou des fonctions : le débuggage est plus efficace) ;
- annoter le programme avec des commentaires pour en améliorer la lisibilité.

Bonnes pratiques

Fichier 'sample_data.csv' :

```
Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Bonnes pratiques

Fichier 'sample_data.csv' :

```
Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Fichier de commandes **R** :

```
data=read.csv('sample_data.csv',skip=3)
for (time_point in c(0,5,10,15))
{
if (length(data$mesure[data$temps==time_point])<=2)
{
print("Problème : pas assez de points expérimentaux au temps :")
print(time_point)
}
}
pdf('Mon_graphique.pdf',width=6,height=6)
boxplot(data$mesure~data$temps)
dev.off()
```

Bonnes pratiques

Fichier 'sample_data.csv' :

```
Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Mauvaise idée (ne pas mélanger données et méta-données)

Fichier de commandes **R** :

```
data=read.csv('sample_data.csv',skip=3)
for (time_point in c(0,5,10,15))
{
if (length(data$mesure[data$temps==time_point])<=2)
{
print("Problème : pas assez de points expérimentaux au temps :")
print(time_point)
}
}
pdf('Mon_graphique.pdf',width=6,height=6)
boxplot(data$mesure~data$temps)
dev.off()
```

Bonnes pratiques

Fichier 'sample_data.csv' :

```
Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Mauvaise idée (ne pas mélanger données et méta-données)

Fichier de commandes **R** :

```
data=read.csv('sample_data.csv',skip=3)
for (time_point in c(0,5,10,15))
{
if (length(data$mesure[data$temps==time_point])<=2)
{
print("Problème : pas assez de points expérimentaux au temps :")
print(time_point)
}
}
pdf('Mon_graphique.pdf',width=6,height=6)
boxplot(data$mesure~data$temps)
dev.off()
```

Propre à ce fichier !

Bonnes pratiques

Fichier 'sample_data.csv' :

```
Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Mauvaise idée (ne pas mélanger données et méta-données)

Fichier de commandes **R** :

```
data=read.csv('sample_data.csv',skip=3)
for (time_point in c(0,5,10,15))
{
if (length(data$mesure[data$temps==time_point])<=2)
{
print("Problème : pas assez de points expérimentaux au temps :")
print(time_point)
}
}
pdf('Mon_graphique.pdf',width=6,height=6)
boxplot(data$mesure~data$temps)
dev.off()
```

Propre à ce fichier !

Propre à cette expérience !

Bonnes pratiques

Fichier 'sample_data.csv' :

```
Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Mauvaise idée (ne pas mélanger données et méta-données)

Fichier de commandes **R** :

```
data=read.csv('sample_data.csv',skip=3)
for (time_point in c(0,5,10,15))
{
if (length(data$mesure[data$temps==time_point])<=2)
{
print("Problème : pas assez de points expérimentaux au temps :")
print(time_point)
}
}
pdf('Mon_graphique.pdf',width=6,height=6)
boxplot(data$mesure~data$temps)
dev.off()
```

Propre à ce fichier !

Propre à cette expérience !

Choix arbitraire, amené à changer (le régler par une variable)

Bonnes pratiques

Fichier 'sample_data.csv' :

```
Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Mauvaise idée (ne pas mélanger données et méta-données)

Fichier de commandes **R** :

```
data=read.csv('sample_data.csv',skip=3)
for (time_point in c(0,5,10,15))
{
if (length(data$mesure[data$temps==time_point])<=2)
{
print("Problème : pas assez de points expérimentaux au temps :")
print(time_point)
}
}
pdf('Mon_graphique.pdf',width=6,height=6)
boxplot(data$mesure~data$temps)
dev.off()
```

Propre à ce fichier !

Propre à cette expérience !

Choix arbitraire, amené à changer (le régler par une variable)

Nom de fichier constant, sera écrasé à la prochaine exécution du programme sur d'autres données

Bonnes pratiques

Fichier 'sample_data.csv' :

```
Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Mauvaise idée (ne pas mélanger données et méta-données)

Fichier de commandes **R** :

```
data=read.csv('sample_data.csv',skip=3)
for (time_point in c(0,5,10,15))
{
if (length(data$mesure[data$temps==time_point])<=2)
{
print("Problème : pas assez de points expérimentaux au temps :")
print(time_point)
}
}
pdf('Mon_graphique.pdf',width=6,height=6)
boxplot(data$mesure~data$temps)
dev.off()
```

Propre à ce fichier !

Propre à cette expérience !

Choix arbitraire, amené à changer (le régler par une variable)

Pourraient être plus générales

Nom de fichier constant, sera écrasé à la prochaine exécution du programme sur d'autres données

Bonnes pratiques

Corriger le fichier d'*input* :

Bonnes pratiques

Corriger le fichier d'*input* :

- $1^{ère}$ possibilité : supprimer les méta-données, et les transférer dans un fichier à part.

Bonnes pratiques

Corriger le fichier d'*input* :

- $1^{ère}$ possibilité : supprimer les méta-données, et les transférer dans un fichier à part.

```
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Bonnes pratiques

Corriger le fichier d'*input* :

- $1^{ère}$ possibilité : supprimer les méta-données, et les transférer dans un fichier à part.

```
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

```
data=read.csv('sample_data.csv')
for (time_point in c(0,5,10,15))
{
if (length(data$mesure[data$temps==time_point])<=2)
{
print("Problème : pas assez de points expérimentaux au temps :")
```

Bonnes pratiques

Corriger le fichier d'*input* :

- $1^{ère}$ possibilité : supprimer les méta-données, et les transférer dans un fichier à part.
- $2^{ème}$ possibilité : commenter les méta-données (caractère « # » en début de ligne), et lire le fichier avec *read.table()*.

Bonnes pratiques

Corriger le fichier d'*input* :

- $1^{ère}$ possibilité : supprimer les méta-données, et les transférer dans un fichier à part.
- $2^{ème}$ possibilité : commenter les méta-données (caractère « # » en début de ligne), et lire le fichier avec *read.table()*.

```
#Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
#Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
#Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

Bonnes pratiques

Corriger le fichier d'*input* :

- $1^{ère}$ possibilité : supprimer les méta-données, et les transférer dans un fichier à part.
- $2^{ème}$ possibilité : commenter les méta-données (caractère « # » en début de ligne), et lire le fichier avec *read.table()*.

```
#Expérience réalisée le 10/11/2022 (sur les échantillons du 08/11/2022)
#Problème sur le 2ème réplicat de la mesure à t=10 min (tube renversé)
#Mesure réalisée sur le Tecan de la salle 006
temps,mesure
0,12.1
0,15.2
0,13.4
5,20.4
5,18.0
5,22.3
10,25.3
10,NA
10,28.9
15,30.1
15,28.3
15,32.7
```

```
data=read.table('sample_data.csv',sep=',',header=TRUE)
for (time_point in c(0,5,10,15))
{
if (length(data$mesure[data$temps==time_point])<=2)
{
print("Problème : pas assez de points expérimentaux au temps :")
```

Bonnes pratiques

Réduire la dépendance du script au format de ce fichier particulier de données :

Bonnes pratiques

Réduire la dépendance du script au format de ce fichier particulier de données :

```
> unique(data$temps)
[1]  0  5 10 15
> for (time_point in unique(data$temps))
+ {
```

Bonnes pratiques

Réduire la dépendance du script au format de ce fichier particulier de données :

```
> unique(data$temps)
[1]  0  5 10 15
> for (time_point in unique(data$temps))
+ {
```

Ne pas coder en dur la valeur du seuil (le définir par une variable) :

Bonnes pratiques

Réduire la dépendance du script au format de ce fichier particulier de données :

```
> unique(data$temps)
[1]  0  5 10 15
> for (time_point in unique(data$temps))
+ {
```

Ne pas coder en dur la valeur du seuil (le définir par une variable) :

```
> cutoff=2
> for (time_point in unique(data$temps))
+ {
+ if (length(data$mesure[data$temps==time_point])<=cutoff)
+ {
```

Bonnes pratiques

Donner au fichier de sortie un nom indexé sur celui du fichier d'entrée :

Bonnes pratiques

Donner au fichier de sortie un nom indexé sur celui du fichier d'entrée :

```
> input='sample_data.csv'
> data=read.table(input,sep=',',header=TRUE)
> cutoff=2
> for (time_point in unique(data$temps))
+ {
+ if (length(data$mesure[data$temps==time_point])<=cutoff)
+ {
+ print("Problème : pas assez de points expérimentaux au temps :")
+ print(time_point)
+ }
+ }
> output=paste('Graphique_de_',sub('.csv$','',input),'.pdf',sep='')
> pdf(output,width=6,height=6)
> boxplot(data$mesure~data$temps)
> dev.off()
null device
          1
>
```

Bonnes pratiques

Éventuellement : rendre le script robuste à un renommage des colonnes (mais en le rendant sensible à l'ordre des colonnes) :

Bonnes pratiques

Éventuellement : rendre le script robuste à un renommage des colonnes (mais en le rendant sensible à l'ordre des colonnes) :

```
> colnames(data)
[1] "temps"  "mesure"
>
```

Bonnes pratiques

Éventuellement : rendre le script robuste à un renommage des colonnes (mais en le rendant sensible à l'ordre des colonnes) :

```
> titre_1=colnames(data)[1]
> titre_1
[1] "temps"
> data[,titre_1]
 [1]  0  0  0  5  5  5 10 10 10 15 15 15
```

Bonnes pratiques

Éventuellement : rendre le script robuste à un renommage des colonnes (mais en le rendant sensible à l'ordre des colonnes) :

```
> input='sample_data.csv'
> data=read.table(input,sep=',',header=TRUE)
> cutoff=2
> titre_1=colnames(data)[1]
> titre_2=colnames(data)[2]
> for (time_point in unique(data[,titre_1]))
+ {
+ if (length(data[,titre_2][data[,titre_1]==time_point])<=cutoff)
+ {
+ print("Problème : pas assez de points expérimentaux au temps :")
+ print(time_point)
+ }
+ }
> output=paste('Graphique_de_',sub('.csv$','',input),'.pdf',sep='')
> pdf(output,width=6,height=6)
> boxplot(data[,titre_2]~data[,titre_1],xlab=titre_1,ylab=titre_2)
> dev.off()
null device
          1
```

Exercices d'application

Exercices d'application

Exercice 1 : Écrire une fonction *mean2* qui calcule la moyenne arithmétique (comme le fait la fonction prédéfinie *mean*) et vérifier, sur quelques échantillons aléatoires (générés par *rnorm*, *runif*, *rpois* ou autre) qu'elles donnent bien le même résultat.

Exercices d'application

Exercice 1 : Écrire une fonction *mean2* qui calcule la moyenne arithmétique (comme le fait la fonction prédéfinie *mean*) et vérifier, sur quelques échantillons aléatoires (générés par *rnorm*, *runif*, *rpois* ou autre) qu'elles donnent bien le même résultat.

Exercice 2 : Écrire un programme qui, à partir indifféremment des fichiers 'Data1.csv' et 'Data2.csv' fournis, calcule pour chaque région le nombre minimal de départements représentés dans le fichier, et la moyenne de leur score « Mesure ».

Exercices d'application

Exercice 3 : À partir des données de la consommation en France depuis janvier 2009 (https://www.insee.fr/fr/statistiques/6654433#tableau-conso-biens-g3-fr) : copier ces données dans un tableau, à exporter au format CSV, puis écrire un script qui génère un fichier PDF avec la courbe de l'évolution de la consommation pour chacun des quatre types de produit (Habillement-textile, Matériels de transport, Équipement du logement, Autres biens fabriqués). La commande *as.Date()* pourra être employée pour convertir les chaînes de caractères en objets de classe « date ».

Exercices d'application

Exercice 3 : À partir des données de la consommation en France depuis janvier 2009 (https://www.insee.fr/fr/statistiques/6654433#tableau-conso-biens-g3-fr) : copier ces données dans un tableau, à exporter au format CSV, puis écrire un script qui génère un fichier PDF avec la courbe de l'évolution de la consommation pour chacun des quatre types de produit (Habillement-textile, Matériels de transport, Équipement du logement, Autres biens fabriqués). La commande *as.Date()* pourra être employée pour convertir les chaînes de caractères en objets de classe « date ».

Exercice 4 : Utiliser ce même script pour tracer l'évolution de la consommation des trois catégories du tableau suivant sur la page de l'INSEE (Biens fabriqués, Énergie, Alimentaire). Adapter le script au besoin, s'il ne permettait pas de travailler sur ce nouveau tableau.