Programmation, Fonctions et Listes

Rappels de programmation

- **Traitement conditionnel** : exécution d'une ou plusieurs commande lorsqu'une condition est vraie.
- **Syntaxe** :

```
if(condition){
 # commandes à exécuter
}
if(condition){
  # commandes à exécuter si la condition est vraie
} else {
  # commandes à exécuter si la condition est fausse
}
ifelse(condition, valeur_si_vraie, valeur_si_fausse)
```

où *condition* est une variable **booléenne**, c'est-à-dire à valeurs *TRUE/FALSE* ou *1/0*.

Rappels de programmation (2)

- **Conditions typiques** :

```
var1 == valeur1
var1 %in% c(valeur1, valeur2, ...)
var2 <= num1
var2 >= num2
var2 <= num1 && var2 >= num2
```

- où
  - *var1* est une variable qualitative ou discrète, et *valeur1*, *valeur2*, ... sont des modalités de *var1*,
  - *var2* est une variable quantitative, et *num1*, *num2* sont des valeurs seuil.
- **Note** : la fonction *ifelse()* fonctionne également sur un vecteur de conditions.

Rappels de programmation (3)

- **Traitement itératif** : commandes à exécuter suivant les valeurs itératives d'une variable (*for()*), ou jusqu'à ce qu'une condition soit satisfaite (*while()*).
- **Syntaxe** :

```
for(var in sequence){
  # commandes à exécuter
}
```

où *var* est une variable prenant ses valeurs dans le vecteur *sequence*.

```
while(condition){
  # commandes à exécuter
}
```

où *condition* est une variable de type booléenne.
Attention : s'assurer que les commandes permettent de satisfaire *condition* au bout d'un nombre **fini** d'itérations, sous peine de tomber dans une boucle infinie.

TP : Exercice 1

- Ecrire deux boucle permettant de calculer respectivement
  - $\sum_{i=1}^{10000} 1/i$,
  - $\sum_{i \in \{1,\ldots,10000 \mid i \text{ pair}\}} 1/i$.
- Réécrire les mêmes boucles, mais en partant de 10000 jusqu'à 1.
- Ajouter dans les boucles précédentes une condition permettant d'enlever les valeurs $i \in \{198, 2067, 532, 8934\}$ des calculs des sommes.

Fonctions

- **Procédure** : calculs à partir d'une ou plusieurs valeurs d'entrée, sans retour de résultat

```
preocedure = function(entree){
  # Commandes à exécuter, fonctions de la valeur d'entree
}
```

- **Fonction** : calculs à partir d'une ou plusieurs valeurs d'entrée, avec retour de résultat

```
fonction = function(entree){
  # Commandes à exécuter, fonctions de la valeur d'entree
  # valeur = valeur calculée par la fonction
  valeur
}
```

- Les procédures ou fonctions sont stockées dans des objets de type *function*, sous le nom donné (ici *procedure* et *fonction*)
- *entree* peut être une suite de valeurs.

Fonctions (2)

- Appel des fonctions :

```
procedure(valeurs entree)
fonction(valeurs entree)
var = fonction(valeurs entree)
```

- Ajout de paramètres : on peut ajouter des paramètres, dont des valeurs par défaut peuvent être indiquées. Typiquement, ces paramètres permettent de définir des conditions qui changent ou ajoutent des calculs dans la fonction.

```
fonction = function(entree, parametres){}
```

ajout de paramètres sans valeur par défaut,

```
fonction = function(entree, parametres = valeur){}
```

ajout de paramètres avec valeur par défaut.

TP : Exercice 2

- Reprendre les deux boucles de l'exercice 1.
- Créer à partir de ces boucles deux fonctions permettant de calculer les sommes suivantes :
  - $\sum_{i=1}^{n} 1/i$,
  - $\sum_{i \in \{1,\ldots,10000 \mid i \ pair\}} 1/i$,

où n sera la valeur d'entrée de ces deux fonctions.

- Créer une fonction unique, avec paramètre, permettant de calculer l'une des deux sommes précédentes suivant la valeur du paramètre.
- Tester la fonction sur plusieurs valeurs de n, avec des paramétres différents.

Listes

- **liste** : Objet permettant de stocker dans une même variable (de type *list()*) des éléments de types différents (tables, vecteurs, listes, etc. . .).
- Liste vide :

```
list()
```

- Exemple : liste composée de 3 éléments de type différents

```
list(1:10, head(LETTERS), head(mtcars))
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
## [1] "A" "B" "C" "D" "E" "F"
##
## [[3]]
##                    mpg cyl disp  hp drat    wt  qsec vs
```

Listes (2)

- Exemple : liste **nommée**

```
l = list(v = 1:10,
     lettres = head(LETTERS),
     modeles = head(mtcars)
     )
l
```

```
## $v
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $lettres
## [1] "A" "B" "C" "D" "E" "F"
##
## $modeles
##                   mpg cyl disp  hp drat    wt  qsec vs
## Mazda RX4        21.0   6  160 110 3.90 2.620 16.46  0
## Mazda RX4 Wag    21.0   6  160 110 3.90 2.875 17.02  0
## Datsun 710       22.8   4  108  93 3.85 2.320 18.61  1
```

Listes (3)

Appel dans la liste

- d'un élément

```
l[[1]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
l$v # si liste nommée
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

- d'une liste

```
l[1:2]
```

```
## $v
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $lettres
```

TP : Exercice 3

- Créer une fonction prenant en entrée la valeur de n et donnant en sortie une liste nommée retournant les 2 valeurs suivantes :
  - n,
  - le vecteur

$$u = \left(\sum_{i=1}^{n} 1/i, \sum_{i \in \{1,\ldots,10000 \mid i \text{ pair}\}} 1/i, \sum_{i \in \{1,\ldots,10000 \mid i \text{ impair}\}} 1/i \right).$$

- Tester la fonction sur plusieurs valeurs de n.

Fonctions spécifiques

- Création de tables résumés à partir d'une *fonction* : si *table* est une table au format *data.frame* ou *tibble*
  - **table %>% summarise(fonction1(var1), fonction2(var2))** : résumé de *table* suivant les calculs de fonctions sur des variables,
  - **table %>% summarise_all(fonction)** : résumé de *table* suivant le calcul de *fonction* sur toutes les variables,
  - **table %>% summarise_at(vars, fonction)** : résumé de *table* suivant les calculs de *fonction* sur la liste de variables *vars*,
  - **table %>% summarise_if(condition, fonction)** : résumé de *table* suivant les calculs de *fonction* sur les variables vérifiant *condition*.
- Pour la gestion de listes, voir les fonctions **map()** et de type **map_xxx()** de la librairie *purrr*, disponible dans tidyverse.

Fonctions spécifiques (2)

- Exemple : table *mtcars* résumée par les moyennes sur les variables

```
library(dplyr)
mtcars %>% summarise_all(mean)
```

```
##         mpg    cyl     disp        hp     drat       wt    
## 1 20.09062 6.1875 230.7219 146.6875 3.596563 3.21725 17.
##      gear   carb
## 1 3.6875 2.8125
```

Fonctions spécifiques (3)

- Exemple : table *mtcars* résumée par les moyennes sur les variables *mpg*, *hp*, *wt*

```
mtcars %>% summarise_at(c("mpg", "hp", "wt"), mean)
```

```
##        mpg       hp      wt
## 1 20.09062 146.6875 3.21725
```

- Exemple : table *starwars* résumée par les moyennes des variables numériques

```
starwars %>% summarise_if(is.numeric, mean, na.rm = TRUE)
```

```
## # A tibble: 1 x 3
##   height  mass birth_year
##    <dbl> <dbl>      <dbl>
## 1   174.  97.3       87.6
```

TP : Exercice 4 avec dplyr

Résumer la table *iris*

- par les moyennes et médianes des variables numériques,
- par les moyennes par espèce des variables numériques.