

## SOLUTION TP n° 1

**Solution 1.** On définit trois vecteurs  $x$ ,  $y$  et  $z$  par les commandes R suivantes :

$x = c(1, 3, 5, 7, 9)$

$y = c(2, 3, 5, 7, 11, 13)$

$z = c(9, 3, 2, 5, 9, 2, 3, 9, 1)$

Reproduire et comprendre les résultats des commandes suivantes :

$x + 2$  renvoie : [1] 3 5 7 9 11

$y * 3$  renvoie : [1] 6 9 15 21 33 39

$length(x)$  renvoie : [1] 5

On obtient ainsi le nombre d'éléments dans  $x$ .

$x + y$  renvoie : [1] 3 6 10 14 20 14

$sum(x > 5)$  renvoie : [1] 2

Ainsi, on a le nombre d'éléments de  $x$  dont les valeurs sont  $> 5$  (soient 7 et 9).

$sum(x[x > 5])$  renvoie : [1] 16

On a ainsi la somme des valeurs des éléments de  $x$  supérieurs à 5 (soit  $7 + 9$ ),

$sum(x > 5 \mid x < 3)$  renvoie : [1] 3

C'est le nombre d'éléments de  $x$  dont les valeurs sont  $> 5$  et  $< 3$  (soient 1, 7 et 9).

$y[3]$  renvoie : [1] 5

On a alors la valeur du 3-ème élément de  $y$ .

$y[-3]$  renvoie : [1] 2 3 7 11 13

Ainsi, on a le vecteur  $y$  privé du 3-ème élément.

$y[x]$  renvoie : [1] 2 5 11 NA NA

On obtient alors un vecteur composé du 1-er élément de  $y$ , donc 2, puis du 3-ème élément de  $y$ , donc 5, puis du 5-ème élément de  $y$ , donc 11, puis du 7-ème élément de  $y$ , donc rien, d'où le NA = Non Available, et enfin, du 9-ème élément de  $y$ , donc rien, d'où le NA.

$(y > 7)$  renvoie : [1] FALSE FALSE FALSE FALSE TRUE TRUE

On a ainsi un vecteur logique dont les éléments sont TRUE si la valeur de l'élément de  $y$  associé est  $> 7$ , et FALSE sinon.

$y[y > 7]$  renvoie : [1] 11 13

On a alors un vecteur contenant les éléments de  $y$  dont les valeurs sont  $> 7$ .

$sort(z)$  renvoie : [1] 1 2 2 3 3 5 9 9 9

On obtient ainsi un vecteur dont les éléments sont les valeurs ordonnées par ordre croissant des valeurs des éléments de  $z$ .

$sort(z, dec = TRUE)$  renvoie : [1] 9 9 9 5 3 3 2 2 1

On obtient ainsi un vecteur dont les éléments sont les valeurs ordonnées par ordre décroissant des valeurs des éléments de  $z$ .

$rev(z)$  renvoie : [1] 1 9 3 2 9 5 2 3 9

On a un vecteur qui réarrange les éléments du vecteur  $z$  en sens inverse.

$order(z)$  renvoie : [1] 9 3 6 2 7 4 1 5 8

On obtient ainsi le vecteur des rangs de classement des valeurs des éléments de  $z$  par ordre croissant (la plus petite valeur est en 9-ème position, la 2-ème plus petite est en 3-ème position...).

$unique(z)$  renvoie : [1] 9 3 2 5 1

On a ainsi un vecteur dont les éléments sont ceux de  $z$  privé des doublons.

$duplicated(z)$  renvoie : [1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE FALSE

On a ainsi un vecteur logique dont les éléments sont `TRUE` si la valeur de l'élément `z` commence à être répété (de gauche à droite),

`table(z)` renvoie : `[1] 1 2 2 1 3` Ainsi, on obtient un vecteur dont les valeurs des éléments précisent le nombre de fois qu'apparaissent les chiffres : 1, 2, 3, 5 et 9.

`rep(z, 3)` renvoie : `[1] 9 3 2 5 9 2 3 9 1 9 3 2 5 9 2 3 9 1 9 3 2 5 9 2 3 9 1`

On a ainsi un vecteur qui concatène 3 fois le vecteur `z`.

**Solution 2.** Créer deux vecteurs de dimensions quelconques. Créer un vecteur en insérant le second vecteur entre les 2-ème et 3-ème éléments du premier vecteur :

```
x = 1:10
y = rep(0, 3)
z = c(x[1:2], y, x[3:length(x)])
```

**Solution 3.**

1. Créer les vecteurs suivants :

(a) `y0` constitué de la suite des entiers de 0 à 10 par pas de 2 :

```
y0 = seq(0, 10, 2)
```

(b) `y1` constitué de tous les entiers pairs entre 1 et 18 :

```
y1 = seq(2, 18, 2)
```

(c) `y2` constitué de 20 fois de suite la valeur 4 :

```
y2 = rep(4, 20)
```

(d) `y3` constitué de 20 nombres entre 0 et 10 :

```
y3 = seq(0, 10, length.out = 20)
```

2. Extraire de `y3` :

(a) le troisième élément:

```
y3[3]
```

(b) tous les éléments sauf le troisième :

```
y3[-3]
```

3. Comparer les commandes suivantes :

```
matrix(y3, nrow = 2)
```

```
matrix(y3, byrow = TRUE)
```

4. Construire une matrice `A` comportant quatre lignes et trois colonnes remplies par lignes successives avec les éléments du vecteur `1:12` :

```
A = matrix(1:12, nrow = 4, ncol = 3, byrow = TRUE)
```

`A`

5. Construire une matrice `B` comportant quatre lignes et trois colonnes remplies par colonnes successives avec les éléments du vecteur `1:12` :

```
B = matrix(1:12, nrow = 4, ncol = 3, byrow = FALSE)
```

`B`

6. Extraire l'élément situé en deuxième lignes et troisième colonne de  $A$  :

```
A[2, 3]
```

7. Extraire la première colonne de  $A$  :

$A[, 1]$  (on prend toutes les lignes, et seulement la première colonne),  
puis la deuxième ligne de  $A$  :

```
A[2, ]
```

8. Construire une matrice  $C$  constituée des lignes 1 et 4 de  $A$  :

```
C = A[c(1, 4), ]
```

```
C
```

**Solution 4.** Construire une matrice comportant 9 lignes et 9 colonnes avec des 0 sur la diagonale et des 1 partout ailleurs (on pourra utiliser la commande `diag`):

```
M = matrix(1, 9, 9) - diag(9)
```

```
M
```

**Solution 5.**

1. Créer un vecteur  $x = (x_1, \dots, x_{11})$  contenant les réels compris entre 0 et 1 par pas de 0.1 :

```
x = seq(0, 1, 0.1)
```

2. Afficher la longueur de  $x$  :

```
length(x) renvoie : [1] 11.
```

3. En utilisant les opérations vectorielles, créer un vecteur  $y = 4x(1 - x)$  :

```
y = 4 * x * (1 - x)
```

4. Tracer la courbe rejoignant les points  $(x_1, y_1), \dots, (x_{11}, y_{11})$  avec la commande `plot` :

```
plot(x, y, type = "l", col = "red")
```

5. Calculer le maximum des  $y_1, \dots, y_{11}$  :

```
max(y) renvoie : [1] 1.
```

6. En quel point le maximum est-il atteint ?

```
x[which.max(y)] renvoie : [1] 0.5 (ou x[y == max(y)]).
```

7. Tracer la courbe de la fonction  $f(x) = 4x^2(1 - x)$ ,  $x \in [-2, 1]$ , en rouge :

```
curve( 4 * x^2 * (1 - x), -2, 1, col = "red")
```

ou

```
x = seq(-2, 1, 0.1)
```

```
y = 4 * x^2 * (1 - x)
```

```
plot(x, y, type = "l", col = "red")
```

**Solution 6.** Utiliser R pour donner les valeurs numériques attendues :

- Combien y-a-t-il de carrés (4 cartes de même valeur) dans un jeu de 32 cartes ?

$$32 / 4 = 8$$

- Combien d'anagrammes peut-on faire avec le mot "dinosaur" ?

Comme les 9 lettres du mot "dinosaur" sont différentes, le nombre d'anagrammes est égal au nombre de permutations des 9 éléments de l'ensemble  $E = \{d, i, n, o, s, a, u, r, e\}$ . Par conséquent, le nombre d'anagrammes est  $9!$ , lequel se calcule avec la commande `factorial(9)` qui renvoie : [1] 362880.

- Combien de chances a-t-on de gagner le super jackpot à l'euromillion ? (donc d'avoir 5 bons numéros parmi 49, et 2 bons numéros étoilés parmi 10) :

Pour les 5 numéros parmi 49, il y a  $\binom{49}{5}$  possibilités et, pour les 2 numéros parmi 10,  $\binom{10}{2}$  possibilités. Donc le nombre total de possibilités est  $\binom{49}{5} \times \binom{10}{2}$ , lequel se calcule avec la commande

```
choose(49, 5) * choose(10, 2)
```

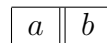
qui renvoie : [1] 85809780

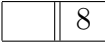

Ainsi, la probabilité de gagner est  $1/85809780 = 1.165368e - 08$ .

Ou alors, directement: `1/(choose(49, 5) * choose(10, 2))` ou avec la fonction `factorial` :

```
1/(factorial(49) / (factorial(5) * factorial(49 - 5)) * factorial(10) / (factorial(2) * factorial(10 - 2))),
```

- Chaque pièce d'un nouveau jeu de domino est de la forme :



avec  $(a, b) \in \{0, \dots, 9\}^2$  en sachant qu'un domino reste le même si on le tourne à 180 degrés (par exemple,   $=$   (c'est un, et un seul domino)). Déterminer le nombre de pièces différentes que contient un jeu complet de dominos.

Le nombre de dominos ayant des nombres de points différents est égal au nombre de combinaisons de 2 éléments parmi 10, donc  $\binom{10}{2}$ . Il y a 10 dominos ayant les deux mêmes nombres de points. Par conséquent, le nombre de dominos différents est  $\binom{10}{2} + 10$ , lequel se calcule avec la commande `choose(10, 2) + 10` qui renvoie : [1] 55.

**Solution 7.** On souhaite calculer avec R les 100 premiers termes de suite de Fibonacci :

$$u_{n+2} = u_{n+1} + u_n,$$

avec  $u_0 = 0$ .

- Créer un script dans le menu "fichier -> Nouveau script", le nommer "fib.R" : OK,
- Sur la première ligne: mettre en commentaire (la ligne commence par `#`) le nom du programme, par exemple `# suite de Fibonacci` : OK,

3. Créer un vecteur  $u$  de taille 100 ne contenant que des 1 :

```
u = rep(1, 100)
```

4. En utilisant la boucle `for`, assigner à  $u_{n+2}$  la valeur  $u_{n+1} + u_n$  :

```
for (n in 1:(length(u) - 2)) {  
  u[n+2] = u[n+1] + u[n]  
}
```

5. En utilisant la commande `plot` représenter la suite  $(u_n)$  sur un graphique :

```
plot(0:(length(u) - 1), u)
```