



# Séance 1: Introduction à la programmation scientifique

<https://economuds.github.io/BIO109/cours1/>

Dominique Gravel  
Laboratoire d'écologie intégrative



# Séance 1

---

- ✓ Ces diapositives sont disponibles en version HTML (Web) et en **PDF**.
- ✓ L'ensemble du matériel de cours est disponible sur la page du portail **moodle**.

## Introduction



# Introduction

---



# Introduction

---



# Introduction

---



## Introduction

---



## Introduction

---



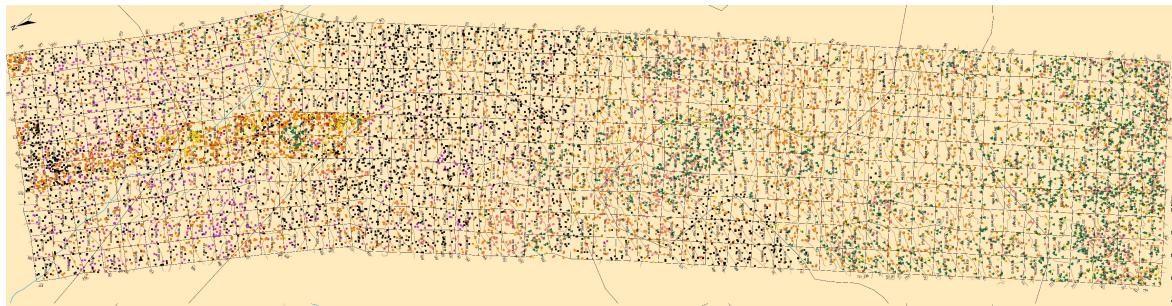
## Introduction

---



# Introduction

---



## Question de recherche (et projet de session)

---

À quelle vitesse se réalisera la migration de l'érable à sucre, et des espèces associées, au sein de la sapinière de montagne du massif des Montagnes vertes ?

## Le type de données

```
arbres <- read.table(file = './donnees/arbres.txt', header = TRUE, sep=";")  
head(arbres)
```

```
##   id_bor borx bory arbre esp multi mort dhp  
## 1    0-0     0    0 34501 acpe FAUX FAUX  82  
## 2    0-0     0    0 34502 acpe VRAI FAUX  26  
## 3    0-0     0    0 34502 acpe VRAI FAUX  98  
## 4    0-0     0    0 34503 acpe FAUX FAUX  73  
## 5    0-0     0    0 34504 acpe FAUX VRAI  28  
## 6    0-0     0    0 34506 fagr FAUX FAUX  26
```

## Exercice 1

---

Ouvrir le fichier **arbres** avec Excel et calculer le nombre d'individus de chaque espèce pour le quadrat 1.

## Exercice 1: solution sur R

```
arbres <- read.table(file = './donnees/arbres.txt', header = TRUE, sep=";")  
quadrats <- table(arbres$id_bor,arbres$esp)  
head(quadrats)
```

```
##  
##      abba acpe acsa beal bepa fagr piru  
## 0-0     1   55   11    7    0   92    0  
## 0-100   0    5    4    3    0    6    0  
## 0-120   2    7   12    4    1    7    0  
## 0-140   4    5    4    8    1    2    1  
## 0-160   2    2   11    8    1    6    1  
## 0-180   5    3    9    7    0    3    1
```

## Exercice 2

---

Ouvrir le fichier **quadrats** avec Excel et calculer la corrélation entre toutes les paires d'espèces.

Petit truc: sur Excel, la fonction pour calculer une corrélation est:

```
=COEFFICIENT.CORRELATION(données_1; données_2)
```

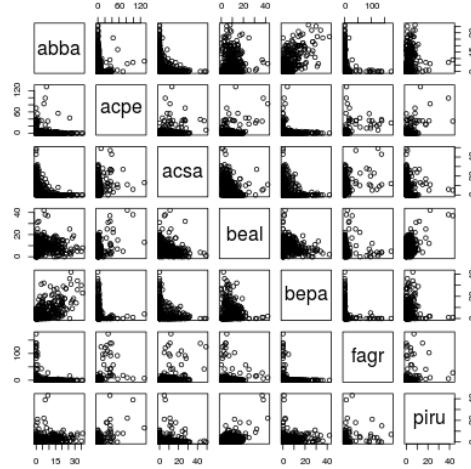
## Exercice 2: solution sur R

```
quadrats <- read.table(file = './donnees/quadrats.txt', header = TRUE, sep= ";")  
cor(quadrats)
```

```
##          abba      acpe      acsa      beal      bepa      fagr      piru  
## abba  1.000000 -0.061892 -0.487308  0.182668  0.665776 -0.180726  0.274651  
## acpe -0.061892  1.000000  0.223190  0.266397 -0.111782  0.449700  0.550620  
## acsa -0.487308  0.223190  1.000000 -0.370743 -0.329024  0.333552 -0.144015  
## beal  0.182668  0.266397 -0.370743  1.000000 -0.030187 -0.109972  0.504235  
## bepa  0.665776 -0.111782 -0.329024 -0.030187  1.000000 -0.153591  0.133875  
## fagr -0.180726  0.449700  0.333552 -0.109972 -0.153591  1.000000  0.192932  
## piru  0.274651  0.550620 -0.144015  0.504235  0.133875  0.192932  1.000000
```

## Exercice 2: visualisation sur R

```
plot(quadrats)
```



## Objectif général

---

Au terme de ce cours, l'étudiant sera en mesure de conceptualiser un problème qui requiert de la programmation scientifique et de réaliser des tâches courantes de programmation.

## Objectifs spécifiques

---

1. Charger des données et exporter des résultats d'analyses au moyen du logiciel R;
2. Conceptualiser un problème au moyen de pseudo-code;
3. Manipuler des données;
4. Rédiger des fonctions;
5. Programmer des algorithmes afin de réaliser des tâches complexes, incluant des boucles et des énoncés conditionnels;
6. Réaliser des simulations de Monte Carlo;

## Contenu

---

1. Introduction et bonnes pratiques de programmation
2. Interagir avec R
3. Les fonctions
4. Algorithmique I: boucles et conditions
5. Algorithmique II: simulations de Monte Carlo

## Ce que le cours n'est pas ...

---

1. Des recettes
2. Un catalogue de fonctions R
3. Un cours de statistiques

## Approche

---

Les connaissances requises pour la programmation scientifique sont minimales, l'apprentissage porte davantage sur l'acquisition de compétences et le développement de capacités à la résolution de problèmes.

Les séances seront constituées de courtes leçons magistrales sur des notions de bases de programmation, entre-coupées d'exercices spécifiques destinés à pratiquer les éléments enseignés. Les séances se conclueront sur la réalisation d'un exercice intégrateur à compléter à la maison.

L'ensemble du matériel du cours sera disponible sur un dépôt Git à l'adresse :

**<https://github.com/EcoNumUdS/BIO109.git>**

## Évaluation

---

L'évaluation porte sur la participation (20%) et sur un travail de session (80%). Un exercice simple sera présenté à la fin des séances 1-4 et **chaque étudiant** devra remettre la solution de l'exercice sous forme de script avant le début de la séance suivante. Les exercices peuvent être réalisés en groupe, mais **chaque étudiant** devra remettre sa propre copie, personnalisée. **Les points sont attribués pour la participation.**

L'évaluation finale portera sur la réalisation d'un projet de programmation **en équipe de 4** à remettre deux semaines après la fin du dernier cours, soit au plus tard le **19 février 2019 à 16:00**. La pénalité sera de 10% par jour de retard. Le rapport final sera évalué à partir de

1. le pseudo-code pour le projet de programmation,
2. le respect des bonnes pratiques de programmation
3. la réussite de l'exercice demandé.

Les étudiants devront remettre le script nécessaire à la réalisation du projet.

## **La place de la programmation en écologie**

---

## Hier

---

La dynamique d'une population:

$$\frac{dN}{dt} = rN(1 - \frac{N}{K})$$

Qui donne pour solution à l'équilibre:

$$N^* = K$$

$N$  = Taille d'une population

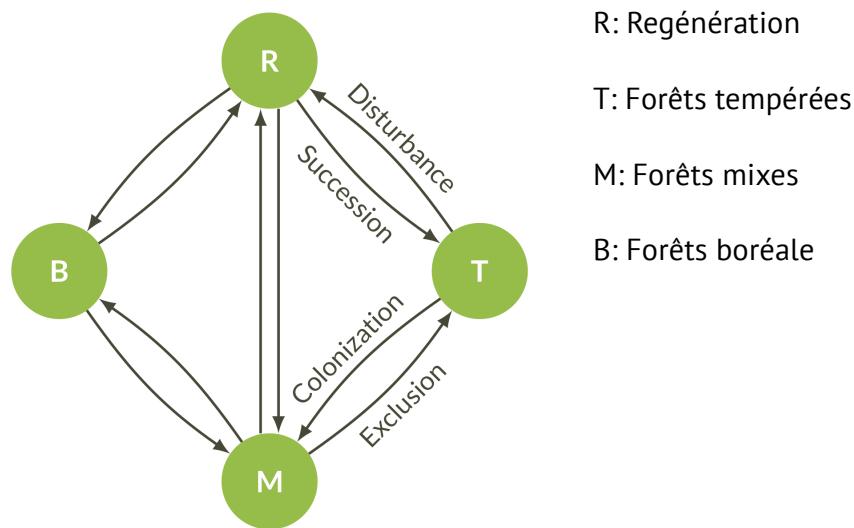
$t$  = temps

$r$  = taux de croissance

$K$  = capacité de support de l'environnement

## Aujourd'hui

---



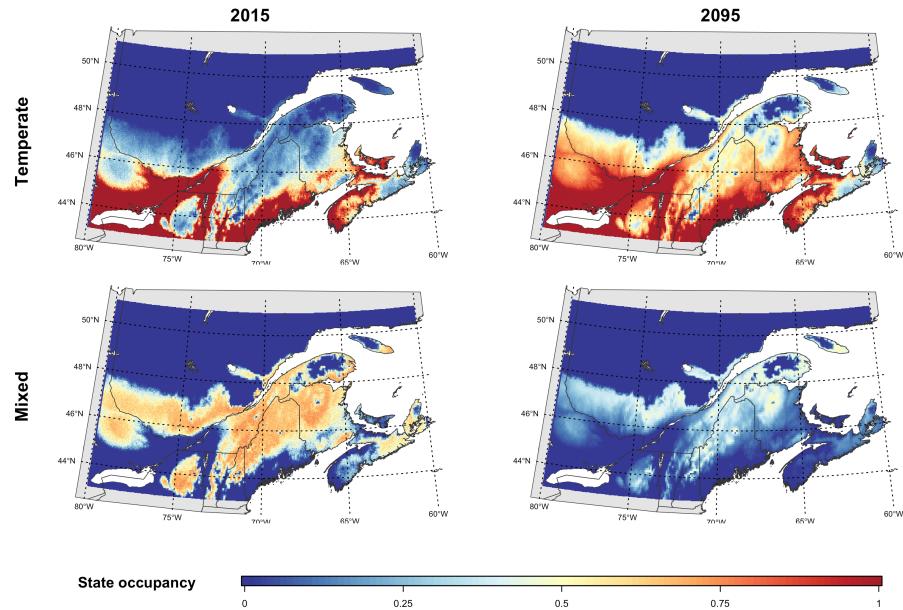
R: Regénération

T: Forêts tempérées

M: Forêts mixtes

B: Forêts boréale

# Aujourd'hui



## Et demain, la modélisation de la biosphère?

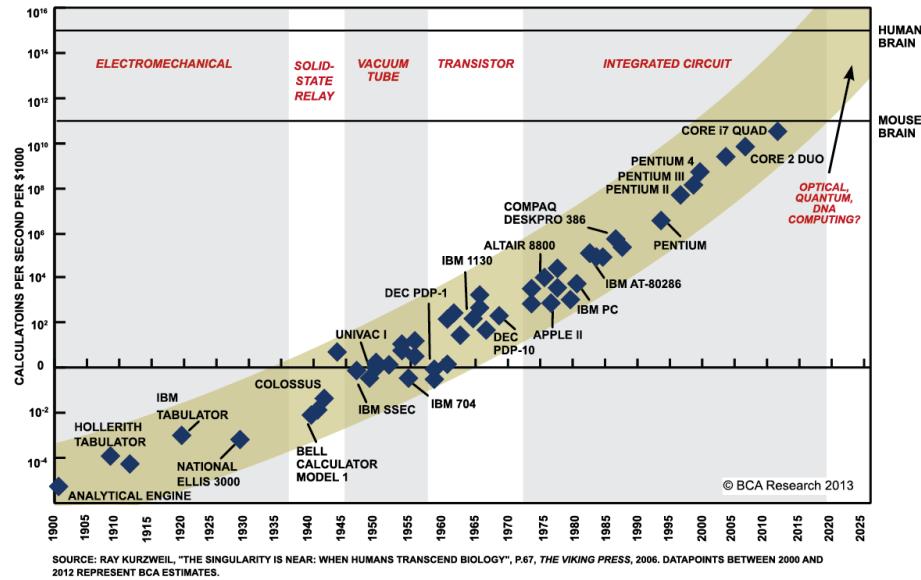


### Time to model all life on Earth

To help transform our understanding of the biosphere, ecologists — like climate scientists — should simulate whole ecosystems, argue **Drew Purves** and colleagues.

# Progression de la puissance de calcul

---



## Utilisation en science au quotidien

---

La programmation est outil indispensable au biologiste 2.0, elle permet:

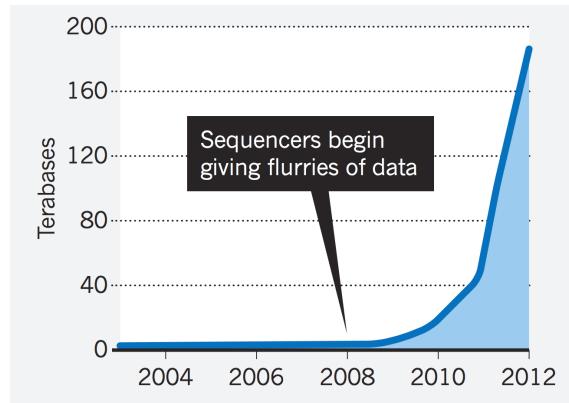
- ✓ Tâches répétitives et/ou complexes (p.ex. Nettoyage des données, Simulations stochastiques)
- ✓ Visualisation et exploration des données
- ✓ Analyses statistiques avancées (p. ex. tests par permutations, statistiques bayésiennes)

# La programmation en science

## Avantages

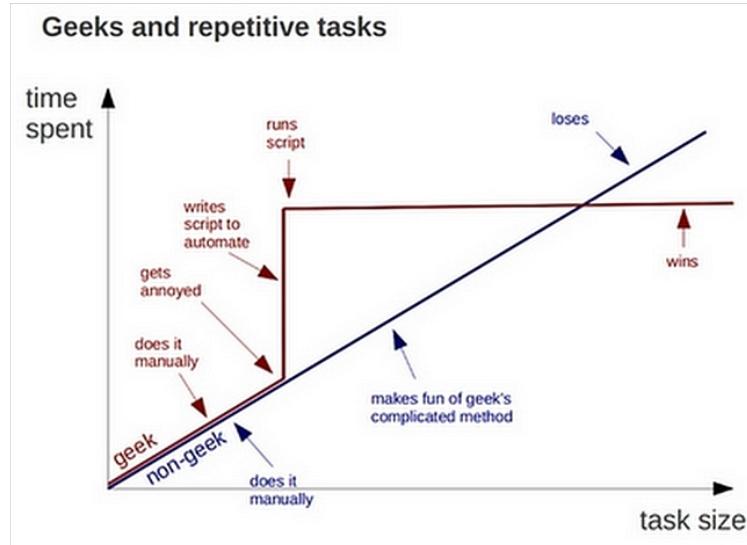
- ✓ Gain de temps
- ✓ Limiter les erreurs
- ✓ Formaliser les opérations
- ✓ Archiver, reproduire et partager
- ✓ Tâches intensives (e.g. en génomique)

Augmentation du volume de données génomiques



NATURE 2013

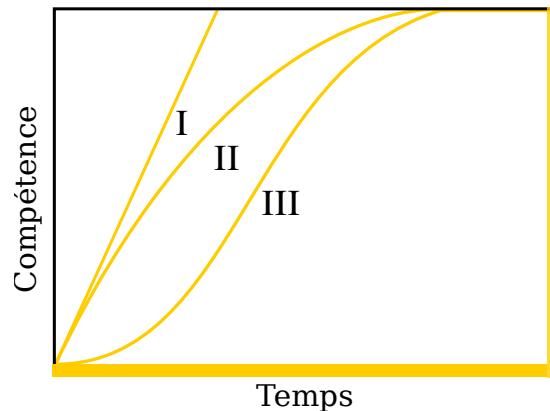
# La programmation en science



## La programmation en science

### Inconvénients

- ✓ L'erreur est avant tout humaine, avant d'être informatique
- ✓ La courbe apprentissage peut être difficile



## Les langages de programmation

---

## Deux grandes familles de langages

---

1. Les langages compilés
2. Les langages interprétés

## 1. Les langages compilés

```
for(i in 1:n){  
    print "Hello world"  
}
```

Code Source

Compilateur

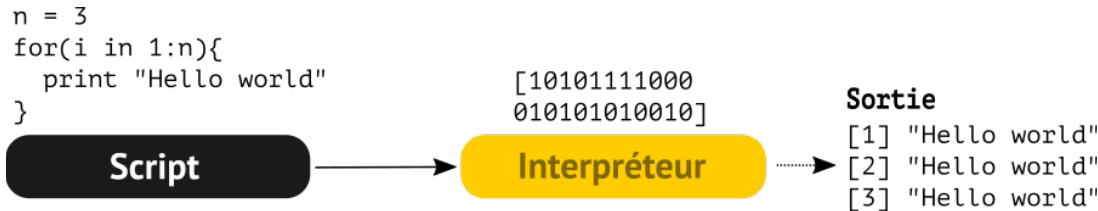
Entrée  
n=3

Exécutable

[10101111000  
010101010010]

Sortie  
[1] "Hello world"  
[2] "Hello world"  
[3] "Hello world"

## 2. Les langages interprétés



# La performance: un critère pour le choix d'un langage

---

	Fortran gcc 5.1.1	Julia 0.4.0	Python 3.4.3	R 3.2.2	Matlab R2015b	Octave 4.0.0	Mathematica 10.2.0	JavaScript V8 3.28.71.19	Go go1.5	LuajIT gsl-shell 2.3.1	Java 1.8.0_45
fib	0.70	2.11	77.76	533.52	26.89	9324.35	118.53	3.36	1.86	1.71	1.21
parse_int	5.05	1.45	17.02	45.73	802.52	9581.44	15.02	6.06	1.20	5.77	3.35
quicksort	1.31	1.15	32.89	264.54	4.92	1866.01	43.23	2.70	1.29	2.03	2.60
mandel	0.81	0.79	15.32	53.16	7.58	451.81	5.13	0.66	1.11	0.67	1.35
pi_sum	1.00	1.00	21.99	9.56	1.00	299.31	1.69	1.01	1.00	1.00	1.00
rand_mat_stat	1.45	1.66	17.93	14.56	14.52	30.93	5.95	2.30	2.96	3.27	3.92
rand_mat_mul	3.48	1.02	1.14	1.57	1.12	1.12	1.30	15.07	1.42	1.16	2.36

## Un autre critère est le 'débugging'

```
char c = value
if ( c == '&' )
    out.println("ampersand");
else if ( c == '=' )
    out.println("assignment operator");
else if ( c == ';' )
    out.println("semicolon");
else if ( c == ',' )
    out.println("comma");
else if ( c == '.' )
    out.println("period");
else if ( c == '-' )
    out.println("minus");
else if ( c == '+' )
    out.println("plus");
else if ( c == '*' )
    out.println("asterisk");
else if ( c == '/' )
    out.println("slash");
else if ( c == '%' )
    out.println("percent");
else if ( c == '#' )
    out.println("hash");
else if ( c == '\'' )
    out.println("single quote");
else if ( c == '\"' )
    out.println("double quote");
else if ( c == '\t' )
    out.println("tab");
else if ( c == '\n' )
    out.println("newline");
else if ( c == '\r' )
    out.println("carriage return");
else
    out.println("unknown character: " + c);
```



## Et en écologie?

---

## Le Pseudo-Code

---

## Le **Pseudo-Code** et ses algorithmes

---

### Définitions

- ✓ En programmation, le **pseudo-code** est une façon de formuler un **algorithme** sans référence à un langage de programmation en particulier.
- ✓ Un **algorithme** est une suite d'actions qui sont réalisées dans un ordre précis par l'ordinateur. C'est une séquence d'étapes dans la résolution d'un problème.

## Le Pseudo-Code

### Exemple

```
PROGRAM DEMO
FOR t IN 1:100
    n_t = n_t * lambda
    PRINT n_t
    IF n_t < 1
        BREAK
    ELSE
        CONTINUE
    END IF
END FOR
```

Le programme **DEMO** fait croître une population à un taux  $\lambda$  et affiche à l'utilisateur si la population est éteinte ( $n_t < 1$ ) ou vivante ( $n_t > 1$ ).

## Le Pseudo-Code

### Exemple

```
PROGRAM DEMO
FOR t IN 1:100 <----- Opération itérative
    n_t = n_t * lambda
    PRINT n_t <----- Le programme affiche la valeur à l'écran
    IF n_t < 1 <----- Opération décisionnelle
        BREAK <----- Le programme arrête son exécution
    ELSE
        CONTINUE <----- Le programme continue son exécution
    END IF
END FOR
```

Le programme **DEMO** fait croître une population à un taux  $\lambda$  et affiche à l'utilisateur si la population est éteinte ( $n_t < 1$ ) ou vivante ( $n_t > 1$ ).

## Les structures de base d'un algorithme

---

On retrouve 3 familles d'opérations:

1. Les opérations séquentielles
2. Les opérations itératives (**FOR**, **WHILE**)
3. Les opérations décisionnelles (**IF**, **IFELSE**)

## Avant-propos

---

Avant de décrire chacune des opérations d'un algorithme, certaines instructions sont communes:

- ✓ **READ**: Le programme lit un fichier
- ✓ **WRITE**: Le programme écrit un fichier
- ✓ **PRINT**: Le programme écrit un message à l'écran pour l'utilisateur
- ✓ **BREAK**: Le programme stop son exécution
- ✓ **CONTINUE**: Le programme continue son exécution

## 1. Les opérations séquentielles

---

Exemple: Calculer l'aire d'un rectangle

```
PROGRAM REC_AIRE
READ hauteur
READ largeur
WRITE hauteur * largeur
```

Chaque opération est effectuée l'une après l'autre dans un ordre déterminé.

## 2. Les opérations itératives

---

Exemple avec **FOR**: Croissance exponentielle

```
PROGRAM DEMO
  FOR t IN 1:100
    n_t = n_t * lambda
  END FOR
```

La population va croître pendant 100 pas de temps.

## 2. Les opérations itératives

---

Exemple avec **WHILE**: Croissance avec capacité de support (K)

```
PROGRAM DEMO
  WHILE n_t < K
    n_t = n_t * lambda
  END WHILE
```

### 3. Les opérations décisionnelles

Exemple avec **IF**: quelques tests sur  $\lambda$

```
PROGRAM DEMO
IF lambda > 0
    PRINT "La population est croissante"
ELSE lambda < 0
    PRINT "La population est décroissante"
ENDIF
```

- ✓ Et si le taux de croissance est nul?

### 3. Les opérations décisionnelles

```
PROGRAM DEMO
IF lambda > 0
    PRINT "La population est croissante"
IF ELSE lambda < 0
    PRINT "La population est décroissante"
ELSE
    PRINT "Absence de croissance"
ENDIF
```

Avec la clause **ELSE**, la croissance est nulle

## Les types d'objets

---

Les objets en programmation sont définis en fonction de leur dimensionalité.

### Dimensionalité

**Dimension 0** : Valeur unique

**Dimension 1** : Vecteur

**Dimension 2** : Matrice

**Dimension 3** : ...

Bien qu'il n'y ai pas de limite à la dimension d'un objet en programmation, pour le cours nous nous limiterons à des objets en deux dimensions (c.à.d Matrice)

## Dimension 0

---

Ces objets ne contiennent qu'une seule information

### Exemple

```
bobo = "toi"  
coco = 2  
dodo = -3  
fofo = 456457.678
```

## Dimension 1

---

Ces objets contiennent un série d'information. Chaque valeur a une position dans le vecteur, laquelle peut être accédée.

### Exemple

```
lettre = ["A" "R" "C" "D" "A"]
lettre[3]
# "C"
```

## Dimension 2

---

Ayant deux dimensions, ces objets présentent les données sous forme de matrices et ont des lignes et des colonnes. Pour accéder à une valeur dans une matrice il faut donner la position de la **ligne** en premier suivit de la position de la **colonne**.

### Exemple

```
lettreTab = ["A" "R" "C"  
           "D" "A" "T"  
           "R" "A" "Q"]  
  
lettreTab[2, 1]  
# "D"
```

## Les règles du **pseudo-code**

---

### A garder en mémoire

1. N'écrivez qu'une seule instruction par ligne de pseudo-code.
2. Écrivez en lettres majuscules le verbe de chaque opération principale.
3. Soyez explicite en nommant les opérations et les variables.
4. Soyez le plus détaillé possible (c.à.d les plus petites étapes possibles)
5. Utilisez des structures de langages de programmation connues (c.à.d **WHILE**, **FOR**, **IF** etc.)
6. Délimitez les étapes en formant des blocs d'instructions par l'utilisation de l'indentation.

**Ces règles sont générales, peu importe le langage de programmation utilisé.**

## Les bonnes pratiques en programmation scientifique

---

## Les 10 commandements de la programmation

---

“ 1. Tu commenteras ton code pour que d'autres puissent le lire, le comprendre et le partager ”

## Les 10 commandements de la programmation

---

|| " 2. Il faut prendre soin de l'environnement et nettoyer ses déchets ,"

## Les 10 commandements de la programmation

---

|| “ 3. Ton script sera dur à avaler. Mieux vaut le découper ,”

## Les 10 commandements de la programmation

---

|| " 4. Plusieurs chiens s'appellent Fido, le tiens tu sauras le nommer ,,"

## Les 10 commandements de la programmation

---

|| " 5. Des pas de bébés permettent aussi d'avancer ,,"

## Les 10 commandements de la programmation

---

“ 6. Un bon programmeur est paresseux. Les opérations répétées doivent être définies sous forme de fonctions ”

## Les 10 commandements de la programmation

---

|| " 7. La vie est trop courte, ton code sera optimisé ,,"

## Les 10 commandements de la programmation

---

|| “ 8. Et un jour tu disparaîtras, alors assure toi que ton code soit reproductible ,”

## Les 10 commandements de la programmation

---

|| " 9. En tout puissant que tu es, le tirage au sort tu pourras répéter ,,"

## Les 10 commandements de la programmation

---

|| “ 10. Et dans le passé tu souhaiteras voyager, utilise le contrôle de versions ”

## Google R Style Rules

---

- ✓ Noms de **fichier**: se termine par .R
- ✓ **Identifiants**: variable.nom (or VariableNom), FonctionNom
- ✓ Longueur de **ligne**: maximum 80 caractères
- ✓ **Indentation**: deux espaces, pas de tabulations
- ✓ **Espacement**: placer des espaces autour des opérateurs binaires
- ✓ **Accolades { }**: s'ouvre sur la même ligne, se ferme sur une ligne indépendante (sauf pour **else**)
- ✓ **else** : Entourer **else** avec des accolades (**}else{**)
- ✓ **Affectation**: utiliser **<-**, pas **=**
- ✓ **Commentaire**: tous les commentaires sont précédés par **#** et suivit d'un espace
- ✓ **Fonction**: doivent avoir une section de commentaires

## Exercice de la semaine

---

## Une situation qui peut arriver tous les jours

---

1. On jette en face de vous 5 lettres d'un jeu de scrabble
2. Un maniac vous demande d'écrire un programme permettant d'ordonner les 5 lettres

Prenez le temps de distinguer les étapes que vous réalisez lorsque vous triez les lettres.  
Essayez de les décrire sous forme de pseudo-code.

**Note:** cet exercice reviendra au cours 4, où vous programmerez cette fonction.