Séance 1

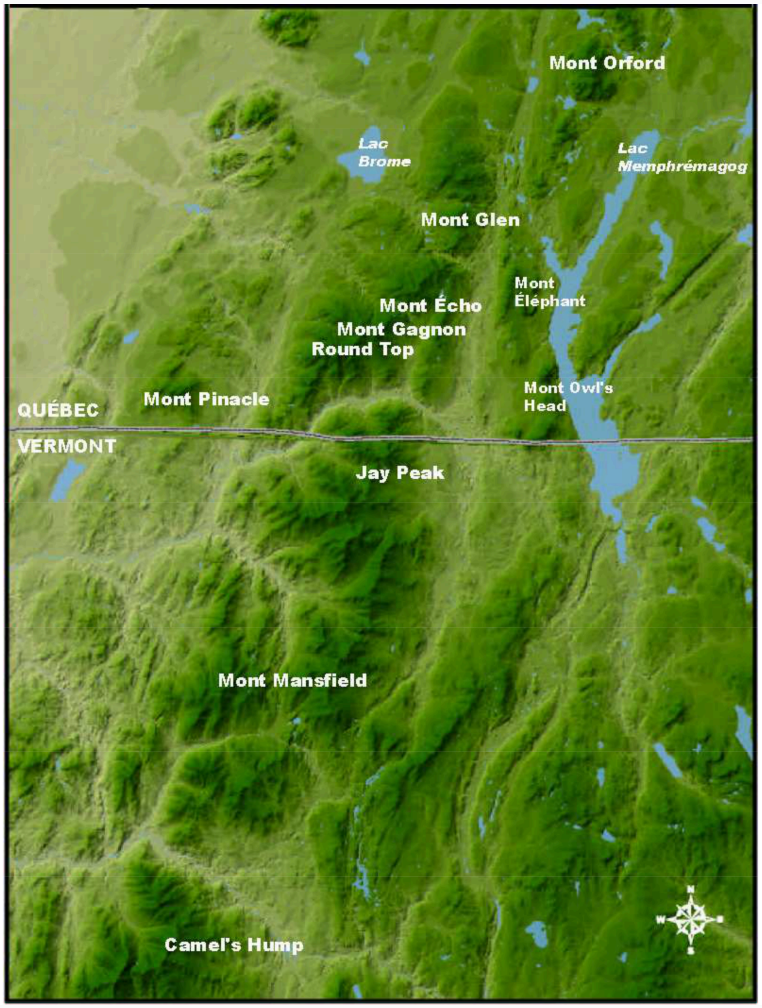
Introduction à la programmation scientifique

BIO109 - Dominique Gravel

Laboratoire d'écologie intégrative | Integrative Ecology Lab [IE]

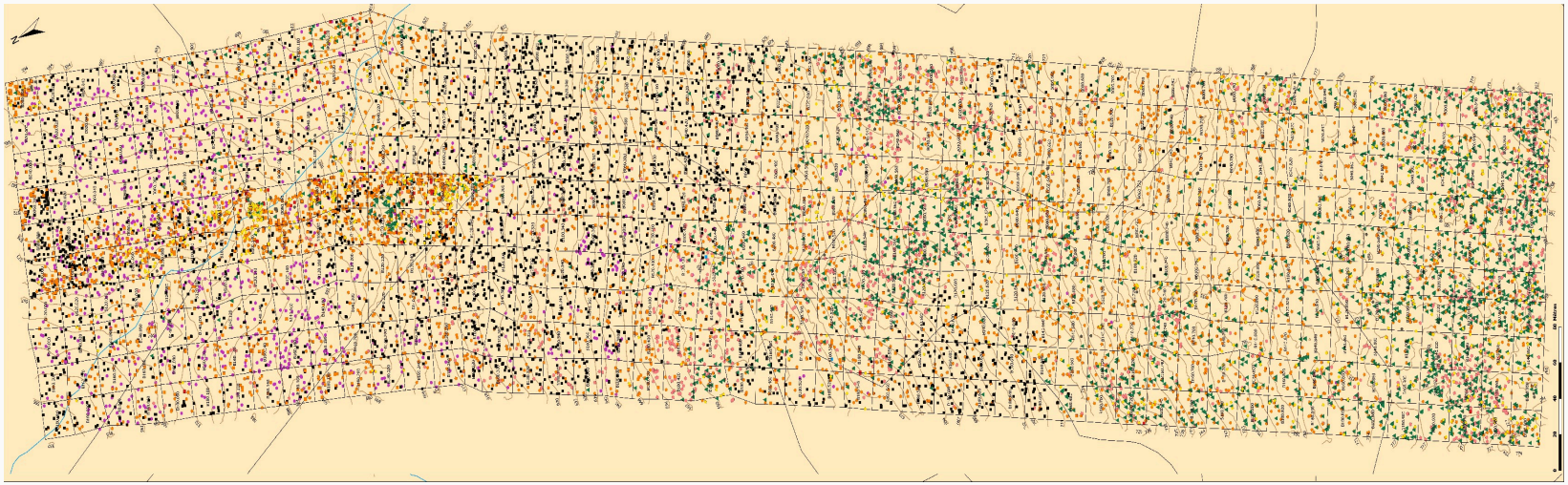
Introduction

Introduction



Introduction

Introduction



Projet de session

Question de recherche

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)
```

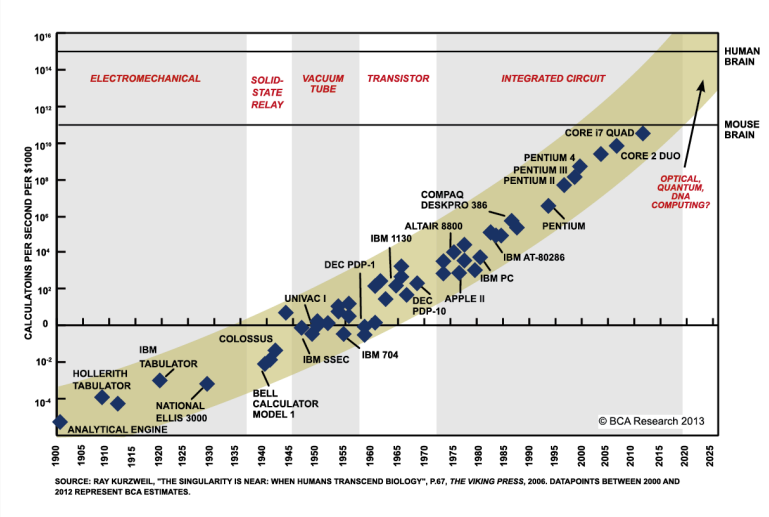
```
##   id_bor borx bory arbre  esp multi mort dhp
## 1    0-0    0    0 34501 acpe  FAUX FAUX  82
## 2    0-0    0    0 34502 acpe  VRAI FAUX  26
## 3    0-0    0    0 34502 acpe  VRAI FAUX  98
## 4    0-0    0    0 34503 acpe  FAUX FAUX  73
## 5    0-0    0    0 34504 acpe  FAUX VRAI  28
## 6    0-0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

La programmation en science

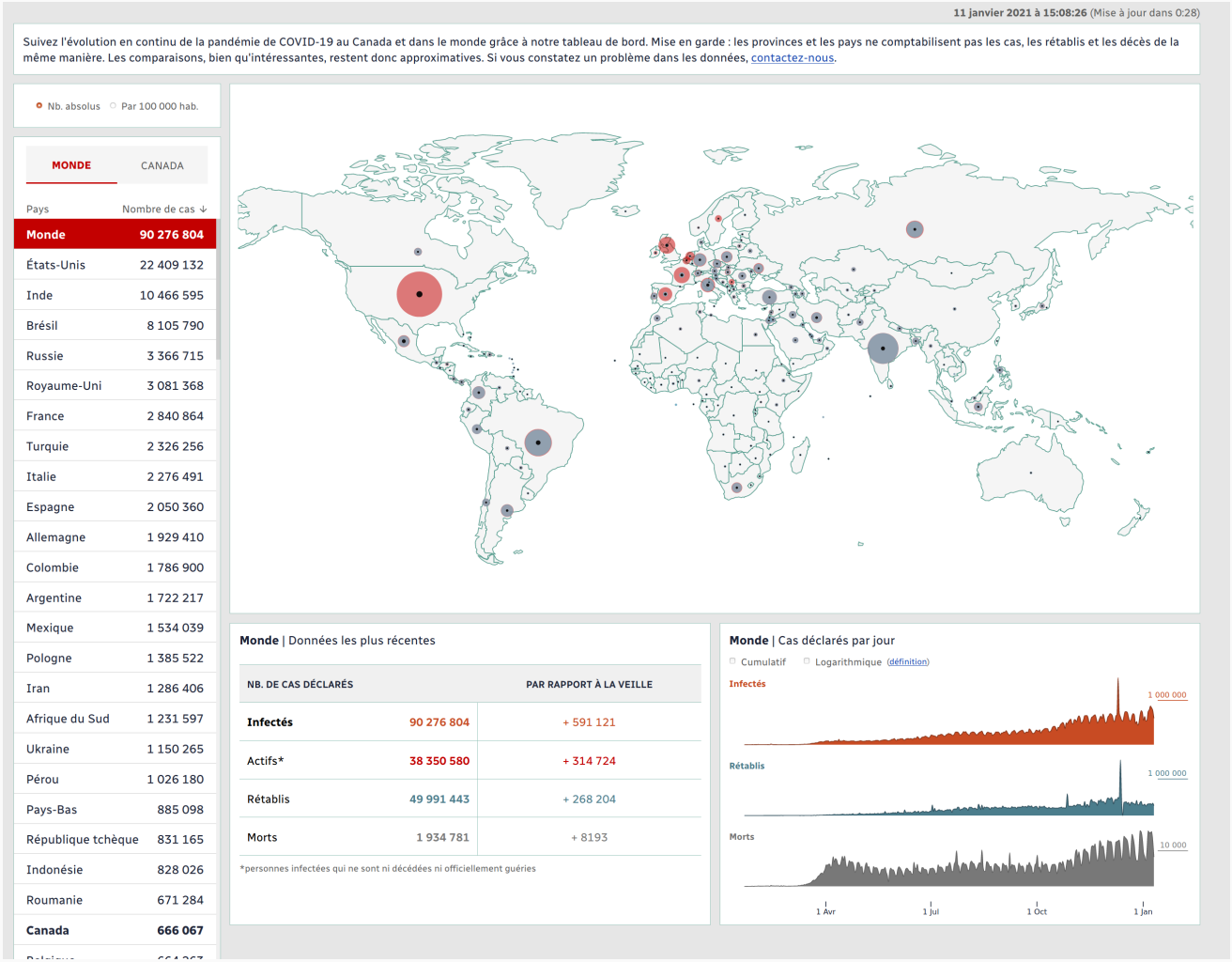
Avantages

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

Capacité de calcul



La programmation et les sciences des données



11 janvier 2021 à 15:08:26 (Mise à jour dans 0:28)

Suivez l'évolution en continu de la pandémie de COVID-19 au Canada et dans le monde grâce à notre tableau de bord. Mise en garde : les provinces et les pays ne comptabilisent pas les cas, les rétablis et les décès de la même manière. Les comparaisons, bien qu'intéressantes, restent donc approximatives. Si vous constatez un problème dans les données, contactez-nous.

Nb. absolus Par 100 000 hab.

MONDE CANADA

Pays	Nombre de cas ↓
Monde	**90 276 804**
États-Unis	22 409 132
Inde	10 466 595
Brésil	8 105 790
Russie	3 366 715
Royaume-Uni	3 081 368
France	2 840 864
Turquie	2 326 256
Italie	2 276 491
Espagne	2 050 360
Allemagne	1 929 410
Colombie	1 786 900
Argentine	1 722 217
Mexique	1 534 039
Pologne	1 385 522
Iran	1 286 406
Afrique du Sud	1 231 597
Ukraine	1 150 265
Pérou	1 026 180
Pays-Bas	885 098
République tchèque	831 165
Indonésie	828 026
Roumanie	671 284
Canada	**666 067**

Monde | Données les plus récentes

NB. DE CAS DÉCLARÉS		PAR RAPPORT À LA VEILLE
Infectés	90 276 804	+ 591 121
Actifs*	38 350 580	+ 314 724
Rétablis	49 991 443	+ 268 204
Morts	1 934 781	+ 8193

*personnes infectées qui ne sont ni décédées ni officiellement guéries

Monde | Cas déclarés par jour

Cumulatif Logarithmique (définition)

Infectés 1 000 000

Rétablis 1 000 000

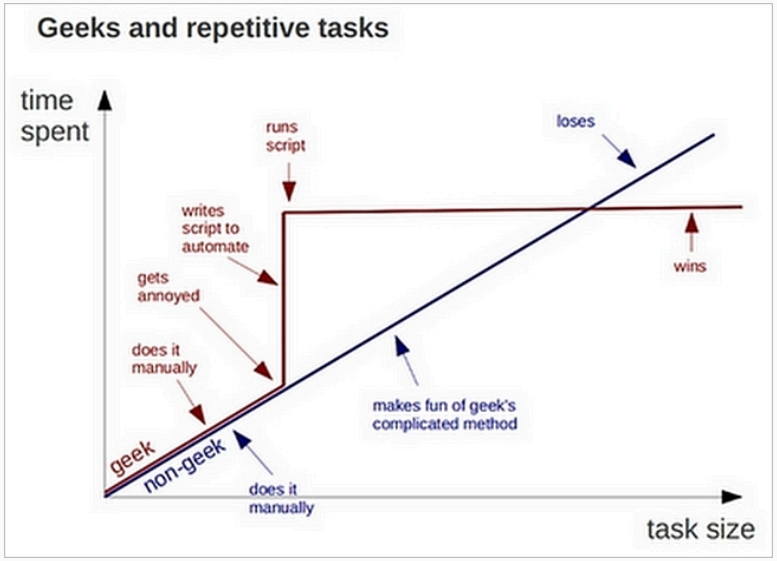
Morts 10 000

1 Avr 1 Jul 1 Oct 1 Jan

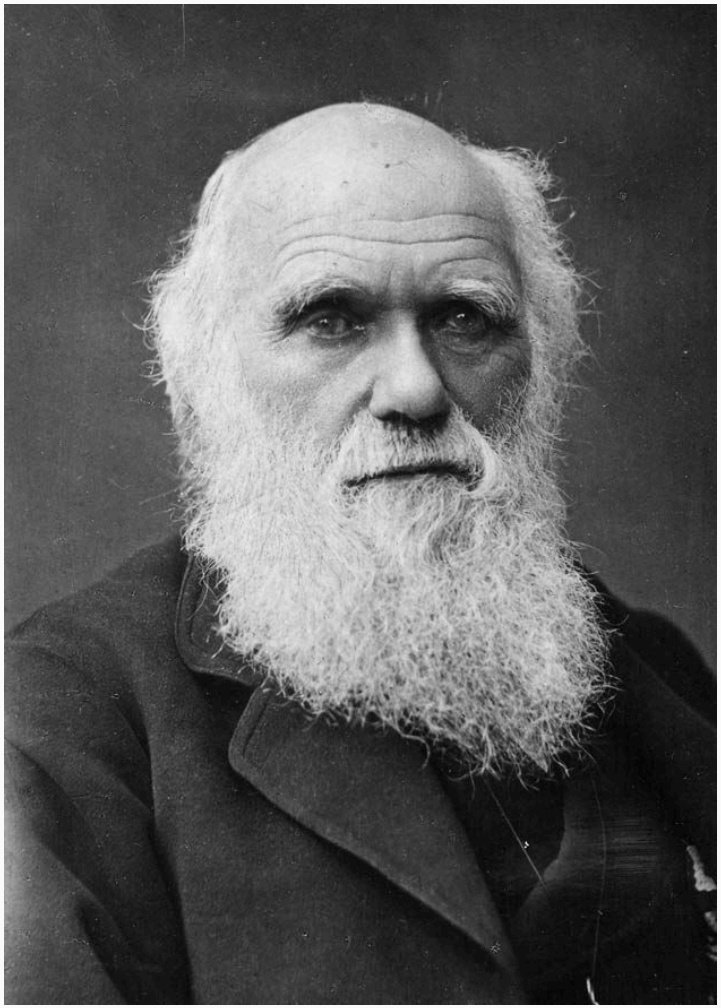
La programmation en science

Inconvénients

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



L'archétype de la biologie



La biologie 2.0

La programmation est un outil indispensable au travail quotidien, elle permet:

- Tâches **répétitives et/ou complexes** (p.ex. nettoyer, explorer et traiter de larges quantités de données)
- Faire des **analyses statistiques avancées** (p. ex. tests par permutations, statistiques bayésiennes)
- Créer des visualisations et des outils interactifs pour **communiquer et sensibiliser** efficacement
- Résoudre des **problèmes complexes** (p.ex. le suivi de la biodiversité, la modélisation de la dynamique des populations, la projection d'aires de distribution)

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. Conceptualiser un problème au moyen de pseudo-code;
2. Manipuler des données;
3. Rédiger des fonctions;
4. Programmer des algorithmes afin de réaliser des tâches complexes, incluant des boucles et des énoncés conditionnels;
5. Réaliser des 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 sont 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 concluront 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

Instructions

- Présence en classe
- Un ordinateur portable
- Matériel du cours : https://github.com/EcoNumUdS/BIO109 et Moodle
- 3h de cours - 6h de travail personnel
- Lire les présentations avant le cours
- Faire les exercices
- Ressources additionnelles : R Cheat Sheet, R pour les débutants, exercies supplémentaires et matériel sur Moodle.

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 sur Moodle sous forme de script *avant le début* de la séance suivante. **Les points sont attribués pour la participation.** Les solutions seront présentées au cours suivant.

- 4 exercies en classe (20%)
- Évaluation terminale : réalisation d'un projet de programmation **en équipe de 2** (80%)
  - Pseudo-code (20%) + programme (60%) à remettre le **18 février à 23h59**

Les évaluations sont à remettre sur Moodle.

Évaluation finale

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.

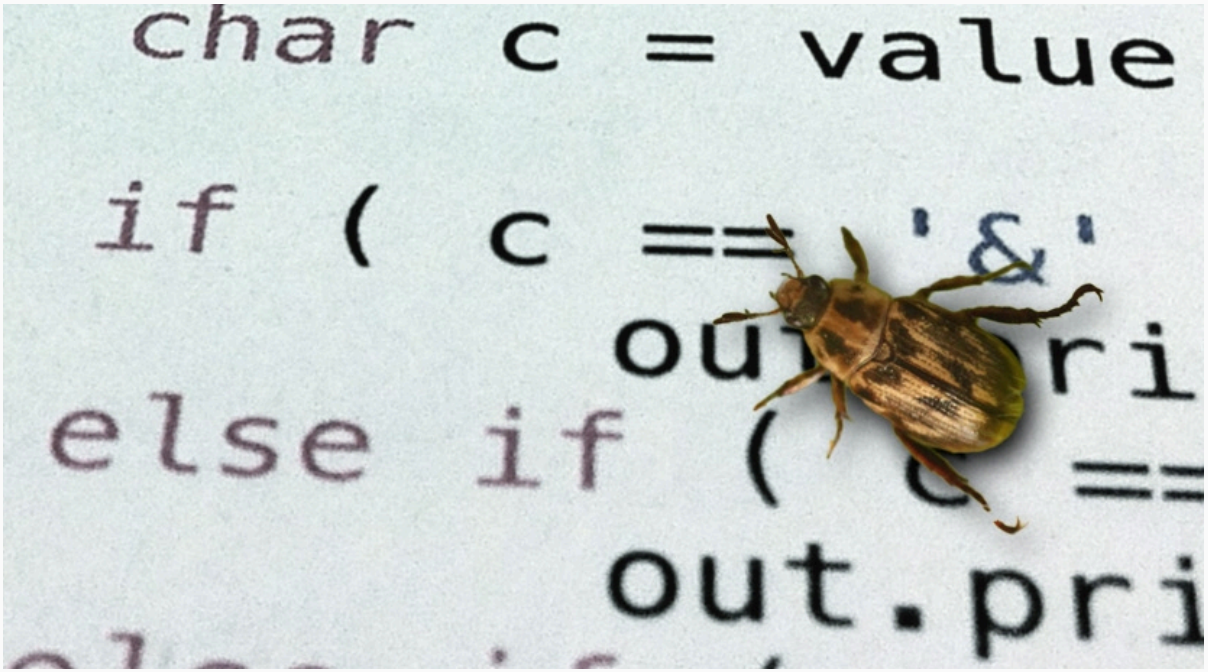
Les étudiants devront remettre le script nécessaire à la réalisation du projet.

Les langages de programmation

Choix d'un langage : performance

	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

Choix d'un langage: 'débugging'



Choix d'un langage : Statistiques et visualisation

Pour ce cours, nous utiliserons R :

- Langage de programmation open-source spécialisé en statistiques et en visualisation de données
- De nombreuses bibliothèques et packages disponibles pour effectuer diverses analyses statistiques
- Interface utilisateur RStudio pour faciliter l'utilisation de R
- Grand nombre de ressources en ligne et communauté active d'utilisateurs de R
- Création de sites Web et d'applications pour sensibiliser aux problèmes environnementaux et promouvoir des solutions durables.

> Langage utilisé dans le cadre de : biométrie, statistiques, méthodes en écologie computationnelle

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.*

- Les principes de base seront les mêmes, d'un langage de programmation à l'autre.

- Les éléments qui suivent résument les principales notions que vous aurez à maîtriser dans le cours. Il faut maîtriser ces notions *avant* de commencer la programmation sur R.

ATTENTION !!

Le pseudo-code n'est pas du code. Il ne peut pas être exécuté. Il sert à décrire les étapes d'un algorithme.

Il est important de bien comprendre la différence entre les deux.

Exemple

```
PROGRAM HELLOWORLD
  FOR t IN 1:100
    PRINT "Hello World"
  END FOR
```

Le programme `HELLOWORLD` imprime à l'écran le message "Hello World" 100 fois.

Exemple

```
PROGRAM HELLOWORLD
  FOR t IN 1:100 <------------- Opération itérative
    PRINT "Hello World" <------ Le programme affiche le message à l'écran
  END FOR <------------------- Fin de la boucle
```

Le programme `HELLOWORLD` imprime à l'écran le message "Hello World" 100 fois.

Exemple

```
PROGRAM DEMO
  DEFINE lambda = 1.1
  SET n_t = 1
  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 croitre une population à un taux λ et affiche à l'utilisateur si la population est éteinte ($n_t < 1$) ou vivante ($n_t > 1$).

Exemple

```
PROGRAM DEMO
  DEFINE lambda = 1.1 <-------- lambda est une constante
  SET n_t = 1  <--------------- n_t est une variable
  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 éxécution
    ELSE
      CONTINUE <--------------- Le programme continue son éxécution
    END IF
  END FOR
```

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

Les concepts de base en programmation

1. Le stockage de l'information
2. Les opérations
    - Séquentielles
    - Itératives (`FOR`, `WHILE`)
    - Décisionnelles (`IF`, `IFELSE`)
3. Les fonctions

Avant-propos

Avant de décrire chacune des opérations d'un algorithme, certaines instructions sont à connaitre pour écrire du pseudo-code :

- `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 arrête son éxécution
- `CONTINUE` : le programme continue son éxécution
- `DEFINE` : définition d'une constante, d'un paramètre
- `SET` : initialise un objet

Assignation des objets

Lorsque l'on crée un objet, il est initialement vide et il faut lui attribuer une valeur :

```
a ← 1
b ← 2
```

On peut faire des opérations mathématiques sur ces objets :

```
a * b
```

Indexation des objets

Plus souvent, on travaille avec des tableaux de données :

```
mat = [1 2 3
       4 5 6]
```

Cet objet a deux dimensions, la première représentée par "i" et l'autre par "j". Sur R par exemple, on accède à la position définie par le couple i,j de la manière suivante :

```
mat[i,j]
```

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 ait 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

```
a = "toi"
b = 2
c = -3
d = 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 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.

Exercice

```
PROGRAM AIRE_CERCLE
  DEFINE pi ← 3.1416
  DEFINE d ← {1, 2, 3, 4, 5}
  DEFINE index ← 2
  CALCUL aire ← pi*(d[index]/2)*(d[index]/2)
  WRITE aire
```

Quel est le résultat de ce calcul ?

Les opérations itératives

La boucle est une séquence de base utilisée par tous les langages de programmation.

Elle permet de répéter une opération un grand nombre de fois automatiquement.

La boucle est définie par un **point de départ**, un **point de d'arrivée** et la **séquence des opérations** à réaliser à chaque itération de la boucle.

Un indice permet de suivre la position dans la boucle.

Les opérations itératives

Exemple avec `FOR`: Croissance exponentielle

```
PROGRAM DEMO
  DEFINE lambda = 1.1
  SET n_t = 1
  FOR etape IN 1:100
    n_t = n_t * lambda
  END FOR
  WRITE n_t
```

La population va croître pendant 100 pas de temps.

Exercice

```
PROGRAM FACTORIELLE
  DEFINE depart = 1
  DEFINE fin = 5
  SET x = 1
  FOR etape in depart:fin
    x = x * etape
  END FOR
  WRITE x
```

Quelle valeur prendra x au terme de l'exécution du programme ?

Les opérations décisionnelles

Souvent un programme repose sur la réalisation d'une décision basée sur le résultat d'un calcul ou bien un paramètre défini par l'utilisateur.

Les décisions sont binaires (oui/non, vrai/faux), mais plusieurs choix peuvent se combiner.

Les opérations décisionnelles

Exemple avec `IF`: quelques tests sur λ

```
PROGRAM DEMO
  READ lambda
  IF lambda > 1
    PRINT "La population est croissante"
  ELSE lambda < 1
    PRINT "La population est décroissante"
  ENDIF
```

Note : on a un problème si lamba = 1 ....

Les fonctions

Elles permettent de répéter rapidement des opérations, sous des conditions définies lorsqu'on les appelle. Ces conditions sont nommées des "arguments".

```
FONCTION AIRE_CERCLE(rayon, pi)
  {
    aire = pi*rayon*rayon
    RETURN aire
  }
```

Les fonctions

Le résultat de la fonction changera si on change les arguments.

```
AIRE_CERCLE(rayon = 2, pi = 3.1416)
```

Ne donnera pas le même résultat que

```
AIRE_CERCLE(rayon = 1, pi = 3.1416)
```

Exemple bonus

Calculez la moyenne de la série de valeurs (vecteur) suivante

```
series = [1, 9, NA, 21, 3, NA, 7]
```

ATTENTION !! Il y a des valeurs manquantes (`NA`) dans la série de chiffres. Il faut les exclure du calcul de la moyenne.

Exemple bonus

```
PROGRAM MOYENNE
  serie = [1, 9, NA, 21, 3, NA, 7]
  n = 7
  somme = 0
  FOR i IN 1:n
    IF serie[i] == NA
      somme = somme
    ELSE
      somme = somme + serie[i]
    END IF
  END FOR
  moyenne = somme / n
  WRITE moyenne
```

Exemple bonus

```
PROGRAM MOYENNE
  SET serie = [1, 9, NA, 21, 3, NA, 7]
  SET n = 0
  SET somme = 0
  FOR i IN 1:7 <------------- Opération itérative
    IF serie[i] == NA <------ Opération décisionnelle
      somme = somme  <------- Si NA
    ELSE <------------------- Si pas NA
      somme = somme + serie[i]
      n = n + 1
    END IF
  END FOR
  moyenne = somme/n <-------- Le programme calcule la moyenne
  WRITE moyenne <------------ Le programme affiche la moyenne
```

Exemple bonus

ATTENTION !!

Le pseudo-code n'est pas du code. Il ne peut pas être exécuté. Il sert à décrire les étapes d'un algorithme.

Il est important de bien comprendre la différence entre les deux.

Les règles du `pseudo-code`

À 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 (la fonction utilisée)
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 `FOR`, `IF`, etc.)
6. Délimitez les étapes en formant des blocs d'instructions par l'utilisation de l'indentation.

Le pseudo-code est général, peu importe le langage de programmation utilisé.

Les bonnes pratiques en programmation scientifique

Les 10 commandements de la programmation

> **1.** Commenter le code pour que d'autres puissent le lire, le comprendre et le partager

Les 10 commandements de la programmation

> **1.** Commenter le code pour que d'autres puissent le lire, le comprendre et le partager
>
> **2.** Découper un script en blocs cohérents

Les 10 commandements de la programmation

> **1.** Commenter le code pour que d'autres puissent le lire, le comprendre et le partager
>
> **2.** Découper un script en blocs cohérents
>
> **3.** Une seule opération par ligne

Les 10 commandements de la programmation

> **1.** Commenter le code pour que d'autres puissent le lire, le comprendre et le partager
>
> **2.** Découper un script en blocs cohérents
>
> **3.** Une seule opération par ligne
>
> **4.** S'assurer qu'une instruction fonctionne avant de passer à l'étape suivante

Les 10 commandements de la programmation

1. Commenter le code pour que d'autres puissent le lire, le comprendre et le partager

2. Découper un script en blocs cohérents

3. Une seule opération par ligne

4. S'assurer qu'une instruction fonctionne avant de passer à l'étape suivante

5. Nommer adéquatement ses variables

Les 10 commandements de la programmation

1. Commenter le code pour que d'autres puissent le lire, le comprendre et le partager

2. Découper un script en blocs cohérents

3. Une seule opération par ligne

4. S'assurer qu'une instruction fonctionne avant de passer à l'étape suivante

5. Nommer adéquatement ses variables

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

Les 10 commandements de la programmation

1. Commenter le code pour que d'autres puissent le lire, le comprendre et le partager

2. Découper un script en blocs cohérents

3. Une seule opération par ligne

4. S'assurer qu'une instruction fonctionne avant de passer à l'étape suivante

5. Nommer adéquatement ses variables

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

7. La vie est trop courte, optimiser son code

Les 10 commandements de la programmation

1. Commenter le code pour que d'autres puissent le lire, le comprendre et le partager

2. Découper un script en blocs cohérents

3. Une seule opération par ligne

4. S'assurer qu'une instruction fonctionne avant de passer à l'étape suivante

5. Nommer adéquatement ses variables

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

7. La vie est trop courte, optimiser son code

8. S'assurer que le code soit reproductible par autrui

Les 10 commandements de la programmation

> **9.** Contraindre les tirages au sort afin de le répéter

Les 10 commandements de la programmation

> **9.** Contraindre les tirages au sort afin de le répéter
>
> **10.** Utiliser le contrôle de version afin de pouvoir revenir dans le passé

Installation de R et de RStudio

À faire pour la **prochaine séance**

- Guide d'installation : https://rstudio-education.github.io/hopr/starting.html

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 maniaque vous demande d'écrire un programme qui ordonne les 5 lettres

Prenez le temps de distinguer les étapes que vous réalisez lorsque vous triez les lettres. Vous devez les écrire sous forme de pseudo-code.

Note 1 : cet exercice reviendra au cours 4, où vous programmerez cette fonction.

Note 2 : vous pouvez assumer que l'ordinateur sait que 'A' vient avant 'B'.