CSC014

Apprentissage des langages R et Python pour la manipulation de données

Séance 1 : Introduction générale

Maxime Woringer et Chloé Mimeau

13 février 2018

Présentation des enseignants □ ▷

Responsable national : Avner Bar-Hen
avner@cnam.fr

Enseignant pour la partie R : Maxime Woringer
maxime@woringer.fr

Enseignante pour la partie Python : Chloé Mimeau
chloe.mimeau@cnam.fr

Modalités de l'UE CSC014 ▹

15 séances hebdomadaires de 4h, du 13 février au 12 juin 2018 :

- 1 séance d'introduction générale à la programmation
- 14 séances qui aborderont 7 thématiques .
  $\longrightarrow$ Chaque thématique sera traitée en 2 séances : 1 en R et 1 en Python

Modalités de l'UE CSC014 ▹

15 séances hebdomadaires de 4h, du 13 février au 12 juin 2018 :

- 1 séance d'introduction générale à la programmation
- 14 séances qui aborderont 7 thématiques .
  $\longrightarrow$ Chaque thématique sera traitée en 2 séances : 1 en R et 1 en Python

Les 7 thématiques seront :

- Notions de base en programmation
- Structures de données
- Manipulation de données
- Visualisation graphique
- Reporting (présentation de résultats)
- Création de pages web interactives
- Fonctionnalités avancées

Modalités de l'UE CSC014 ▹

Exemples et démonstrations :

- En R
  - Une application web aboutie
  - Un exemple plus simple à faire en moins de 10 lignes
- En Python :
  - Une application web aboutie
  - Des exemples plus simples (1 et 2) à faire en quelques lignes

Modalités de l'UE CSC014

Évaluation finale

Mode d'évaluation : Évaluation sur projet, à rendre à la fin du semestre en R **ou** en Python

Objectif : Mettre en œuvre l'ensemble des notions vues au cours du semestre sur un jeu de données au choix.

Modalités de l'UE CSC014

Contrôle continu

2 devoirs maison à rendre au cours du semestre :

- Devoir 1 : à rendre le 20 mars 2018
  Objectif : maîtriser les bases de la programmation et l'environnement
- Devoir 2 : à rendre le 15 mai 2018
  Objectif : maîtriser les fonctionnalités avancées pour la manipulation de données

Modalités de l'UE CSC014

Moodle

Tous les échanges possibles de documents, d'informations, etc... se feront via la page Moodle du cours CSC014 (accessible à partir de vos identifiants, obtenus après inscription).

- L'ensemble des **supports de cours** y seront déposés (transparents, notices d'explication, fichiers de données de travail, codes ...)
- Toutes les **annonces** qui pourront vous être destinées au long du semestre seront faites à partir de cette page Moodle
- Les **projets** finaux seront à déposer dans l'espace qui sera crée à cet effet sur la page Moodle

Installation des langages R et Python sur ordinateurs personnels ▹

Installation de R :
Suivre les instructions données sur la page Moodle concernant l'installation de R et Rstudio

Installation de la distribution Anaconda (pour Python) :
Suivre les instructions données sur la page Moodle concernant l'installation d'Anaconda

Présentation du JupyterHub du Cnam ▹

Que vous ayez ou non installé R ou Python sur votre ordinateur personnel, il vous sera toujours possible de programmer dans ces deux langages grâce au **JupyterHub** du Cnam.

JupyterHub est un Hub multi-utilisateurs installé au Cnam permettant d'utiliser ce que l'on appelle des **notebook** ou **carnets numériques Jupyter**. Ces carnets numériques se présentent sous la forme de cellules dans lesquelles il est possible d'implémenter du code.

Sur le JupyterHub du Cnam, il est possible de créer des carnets numériques en R et en Python.

Ce Hub est accessible partout avec n'importe quel type de support équipé d'une connexion internet (ordinateur, téléphone, tablette, ...) à partir des **identifiants personnels** qui vous sont distribués dans ce cours. Pour cela aller sur :

https://jhub.cnam.fr/

Qu'est-ce qu'un langage de programmation ?

source : www.openclassroom.fr

Votre ordinateur est une machine bizarre. On ne peut s'adresser à lui qu'en lui envoyant des 0 et des 1. Ainsi, si je traduis "Fais le calcul 3 + 5" en langage informatique, ça pourrait donner quelque chose comme :

00101101100100110100111110

Ce que vous voyez là, c'est le langage informatique de votre ordinateur, appelé **langage binaire**. Votre ordinateur ne connaît que ce langage-là et, comme vous pouvez le constater, c'est absolument incompréhensible.

Voilà donc le vrai problème : *Comment parler à l'ordinateur plus simplement qu'en binaire avec des 0 et des 1 ?*

Votre ordinateur ne parle pas l'anglais et encore moins le français. Pourtant, il est inconcevable d'écrire un programme en langage binaire.

Qu'est-ce qu'un langage de programmation ?

Les langages compilés

L'idée que les informaticiens ont eue, c'est d'inventer de nouveaux langages qui seraient ensuite traduits en binaire pour l'ordinateur.

Le plus dur à faire, c'est de réaliser le programme qui fait la "traduction". Heureusement, ce programme a déjà été écrit par des informaticiens et nous n'avons pas à le refaire.

On va au contraire s'en servir pour écrire des phrases comme : "Fais le calcul 3 + 5" qui seront traduites par le programme de "traduction" en quelque chose comme : "00101101100100110100111110".

Qu'est-ce qu'un langage de programmation ?

Les langages compilés

Dans la première case, le "langage simplifié" est appelé en fait **langage de haut niveau**. Plus un langage est haut niveau, plus il est proche de votre vraie langue.

Un autre mot de vocabulaire à retenir est **code source**. Ce qu'on appelle le code source, c'est le code de votre programme écrit dans un langage de haut niveau dans un **fichier**.

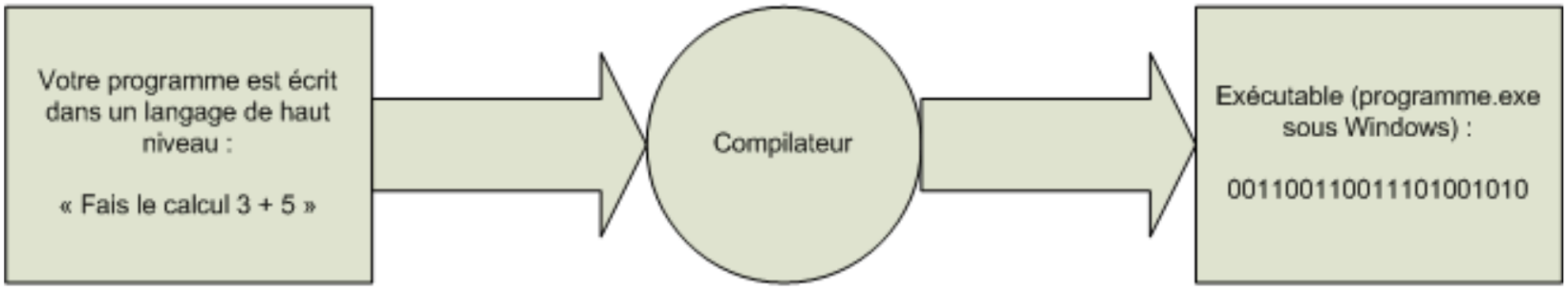
Ensuite, le "programme de traduction" qui traduit notre langage de haut niveau en binaire a un nom : on l'appelle le **compilateur**. La traduction, elle, s'appelle la compilation.

Enfin, le programme binaire créé par le compilateur est appelé l'**exécutable**.

Qu'est-ce qu'un langage de programmation ?

Les langages compilés

Reprenons notre schéma précédent, et utilisons cette fois le vocabulaire défini ci-dessus:



Exemple de langages compilés : C, C++, Fortran

Qu'est-ce qu'un langage de programmation ?

Les langages interprétés

Dans le cadre de ce cours on utilisera les langages **R et Python**, qui eux sont des langages de programmation dits **interprétés**.

Contrairement au langage compilé, le langage interprété est en quelque sorte traduit au fur et à mesure de l'exécution par ce que l'on appelle l'**interpreteur**.

Les langages interprétés sont des langages **de plus haut niveau** que les langages compilés. En pratique, cela signifie qu'ils sont plus accessibles: en temps que programmeur, ce type de langage vous paraîtra plus simple à apprivoiser pour créer votre code source. En revanche, par définition, un langage compilé sera plus rapide à l'exécution qu'un langage interprété. C'est comme si un discours est traduit à l'avance : les gens le lisent plus vite que si le discours est traduit au fur et à mesure par l'interprète.

Présentation du langage R

Histoire : Le projet R naît en 1993 comme un projet de recherche de Ross Ihaka et Robert Gentleman (basé sur le langage R développé par John Chambers). Environ 20 développeurs aujourd'hui.

Atouts :

- langage libre
- solide communauté dans le monde des données (fort soutien en ligne, beaucoup de librairies publiées)
- outil très adapté à l'analyse statistique, la production de graphiques ou de rapports, dans un but exploratoire
- s'intègre bien avec d'autres langages informatiques comme C++, Java et C
- ne nécessite pas d'expérience en programmation

Présentation du langage Python ▹

Histoire : Langage créé en 1989 par Guido Van Rossum, qui, fan de la série TV Monty Python's Flying Circus, décide de baptiser ce projet Python. Environ 250 développeurs aujourd'hui.

Atouts :

- langage libre
- facilité de prise en main, syntaxe simple, codes compacts et grande lisibilité
- solide communauté internationale qui ne fait que croître
- s'intègre bien avec d'autres langages informatiques
- langage adapté à l'accès aux bases de données relationnelles et aux développements liés à Internet comme les navigateurs Web, les agents intelligents et les moteurs de recherche
- élu "meilleur langage 2017" par l'IEEE, la plus grande association mondiale de professionnels techniques (https://www.developpez.com)

Instructions ▹

Les instructions sont les commandes permettant de définir le comportement d'un programme. Les instructions fondamentales sont :

- la **déclaration** et l'**affectation** c'est-à-dire donner une valeur ou le résultat d'une expression à une variable, dont on choisit le nom. Exemple : `x = 1` : une valeur (toujours située à droite du signe =), ici 1, est affectée à une variable (toujours située à gauche du signe =), ici x. La variable x occupe donc l'espace nécessaire dans la mémoire de la machine pour pouvoir stocker la valeur 1.
- la **conditionnelle**, c'est-à-dire exécuter une suite d'instructions si une expression est vérifiée, et éventuellement une autre suite d'instructions si cette expression n'est pas vérifiée (typiquement: boucle `if`),
- les **itérations**, c'est-à-dire répéter une suite d'instructions tant qu'une certaine condition d'arrêt n'est pas vérifiée (typiquement: boucle `for` ou boucle `while`).

En général, on écrit une instruction par ligne par souci de clarté.

Les types de données ▹

Les types de données simples

Dans la plupart des langages de programmation il est possible d'attribuer arbitrairement aux variables des **types** pour déterminer la nature des données qu'elle peut contenir et la manière dont elle sont enregistrées et traitées par le système.

Les types de données ▹

Les types de données simples

Les 4 types de données simples (ou élémentaires) dont nous nous servirons sont :

- l'**entier** (*integer* en anglais) : c'est le type numérique le plus simple : il stocke une valeur qui est un entier relatif.
- le **nombre à virgule flottante** (*float* en anglais) : il stocke une valeur qui est un nombre réel.
- le **caractère** : ce type est prévu pour stocker des lettres.
- le **booléen** : un booléen ne peut prendre que deux valeurs : `True` (=1) ou `False` (=0). Ce type est utilisé lorsqu'une expression logique est testée.

Les types de données ▹

Les types de données simples

R

Pour chaque type simple, on affecte une valeur à la variable x. On pourra afficher la valeur de x en faisant `x` et on pourra connaître le type de x en faisant `class(x)`

- entier :
  `x = 5`
- nombre à virgule flottante :
  `x = 5.0`
- caractère :
  `x = "t"`
- booléen :
  `x = 0<3`
  `x = 0>3`

Python

Pour chaque type simple, on affecte une valeur à la variable x. On pourra afficher la valeur de x en faisant `x` et on pourra connaître le type de x en faisant `type(x)`

- entier :
  `x = 5`
- nombre à virgule flottante :
  `x = 5.0`
- caractère :
  `x = "t"`
- booléen :
  `x = 0<3`
  `x = 0>3`

Les tests conditionnels ▹

Premier test conditionnel : Si ... Alors

On souhaite très souvent que les programmes aient un comportement dynamique, voire interactif, c'est-à-dire que leur exécution se fasse en fonction d'événements intervenant au cours de l'exécution. Ceci est possible par les tests conditionnels :

Syntaxe :

```
Si (expression) Alors
    instructions
```

Ici `expression` désigne quelque chose que l'on peut évaluer comme étant vrai ou faux : c'est donc un booléen.

Les tests conditionnels ▹

Premier test conditionnel : Si ... Alors

R

```
x = 0
y = 0
if (x > 5) {
     x = 1
     y = 2 }
print(x)
print(y)
```

Note : en R les instructions à l'intérieur d'un `if` sont écrites entre crochets {}. L'indentation après le `if` n'est pas obligatoire mais fortement recommandée !

Python

```
x = 0
y = 0
if (x > 5) :
     x = 1
     y = 2
print(x)
print(y)
```

Note : en Python il y a toujours "**:**" après un `if` et il est **obligatoire d'indenter** les instructions à l'intérieur d'un `if`.

Les tests conditionnels ▹

Premier test conditionnel : Si ... Alors

R

```
x = 0
y = 0
if (x > 5) {
     x = 1
     y = 2 }
print(x)
print(y)
```

Note : en R les instructions à l'intérieur d'un `if` sont écrites entre crochets {}. L'indentation après le `if` n'est pas obligatoire mais fortement recommandée !

Python

```
x = 0
y = 0
if (x > 5) :
     x = 1
     y = 2
print(x)
print(y)
```

Note : en Python il y a toujours "**:**" après un `if` et il est **obligatoire d'indenter** les instructions à l'intérieur d'un `if`.

Nous voyons ici un premier exemple de **non-séquentialité du code** puisque les instructions d'affectations `x = 1` et `y = 2` seront sautées, et par conséquent non exécutées. Dans ce cas, les variables x et y restent égales à 0.

Les tests conditionnels ▹

Deuxième test conditionnel : Si ... Alors ... Sinon

De façon assez naturelle, on peut rajouter un Sinon :

R

```
x = 0
y = 0
if (x > 5) {
      x = 1
      y = 2
} else {
      x = 5
      y = 7
}
print(x)
print(y)
```

Python

```
x = 0
y = 0
if (x > 5) :
      x = 1
      y = 2
else :
      x = 5
      y = 7
print(x)
print(y)
```

Les tests conditionnels ▹

Troisième test conditionnel : Si ... Alors ... Sinon Si ... Alors ... Sinon

Enfin, il est possible d'avoir plus de 2 conditions dans un test conditionnel :

R

```
x = 0
y = 0
if (x > 5) {
      x = 1
      y = 2
} else if (x ==2) {
      x = 5
      y = 7
} else {
      x = 10
      y = 11
}
print(x)
print(y)
```

Python

```
x = 0
y = 0
if (x == 1) :
      x = 1
      y = 2
elif (x == 2) :
      x = 5
      y = 7
else :
      x = 10
      y = 11
print(x)
print(y)
```

Les opérateurs ▹

Opérateurs élémentaires (R et python) :

- somme de a et b : $a + b$
- soustraction de a par b : $a - b$
- produit de a par b : $a * b$
- division de a par b : a / b
- a à la puissance b : $a ** b$

R

- division entière de a par b : $a \%/\% b$
- a modulo b : $a \%\% b$

Python

- division entière de a par b : $a // b$
- a modulo b : $a \% b$

Les opérateurs ▹

Opérateurs de comparaison (en R et Python) :

- a égal à b : $a == b$
- a différent de b : $a != b$
- a strictement supérieur à b | supérieur ou égal : $a > b$ | $a >= b$
- a strictement inférieur à b | inférieur ou égal : $a < b$ | $a <= b$

Opérateurs logiques :

R

- ET logique : bool1 & bool2
- OU logique : bool1 | bool2
- négation : ! bool1

Python

- ET logique: bool1 and bool2
- OU logique : bool1 or bool2
- négation : not bool1

Les boucles

Types de boucles

Les boucles permettent à l'ordinateur de répéter un même bloc d'instructions plusieurs fois. Il existe deux types de boucle : les boucles `while` et les boucles `for`.

Boucle `for`: le nombre de répétitions à faire est connu à l'avance.

Boucle `while`: le nombre de répétitions n'est pas connu à l'avance.

Note: on peut toujours réexprimer une boucle `for` comme une boucle `while`, mais l'inverse n'est pas toujours vrai.

Les boucles

La boucle `while`

Syntaxe :

```
tant que  (condition vérifiée)  faire:
    instructions
    mettre à jour condition
```

Exemple : afficher tous les multiples de 7 inférieurs à 100.

R

```
multiple = 0
while (multiple < 100) {
  print(multiple)
  multiple = multiple + 7
}
```

Python

```
multiple = 0
while multiple < 100:
    print(multiple)
    multiple = multiple + 7
```

Les boucles

La boucle `for`

En théorie, la boucle while permet de réaliser toutes les boucles que l'on veut. Toutefois, il est dans certains cas utile d'avoir un autre système de boucle plus "condensé", plus rapide à écrire.

Syntaxe :

```
pour "compteur" allant de "start" à "stop" par pas de "step" faire:
    instructions
```

Cela signifie que la variable "compteur" va prendre toutes les valeurs entières entre "start" et "stop" par pas de "step" et que pour chacune de ces valeurs les instructions écrites dans la boucle `for` seront exécutées.
$\longrightarrow$ En pratique cela signifie qu'à la fin de chaque itération la valeur de "compteur" est automatiquement **incrémentée** de "step" (compteur = compteur + step).

Les boucles

La boucle `for`

R

```
for (k in seq(1,10,by=2)) {
     print(k) }

for (k in seq(10)) {
     print(k) }

for (k in seq(10)) {
     print("Salut les pros de R") }
```

Note : comme pour la conditionnelle `if`, les instructions présentes à l'intérieur de la boucle `for` sont entre crochets {}.
Le triplet "start, stop, step" est défini par la commande "seq". **Par défaut** "start" est initialisé à 1 et "step" vaut 1.

Python

```
for k in range(1,10,2) :
     print(k)

for k in range(10) :
     print(k)

for k in range(10) :
     print("Salut les pros de Python")
```

Note : comme pour le `if`, il faut mettre ":" et indenter après le `for`. **Par défaut** la fonction `range` initialise toujours les valeurs entières d'une boucle à 0 et avance par pas de 1. Pour la valeur finale, Python s'arrête à la valeur donnée **moins 1**

Les fonctions

Les fonctions sont des séries d'instructions **réutilisables**. Elles permettent d'éviter la redondance du code et de le rendre plus facile à lire. Elles peuvent accepter des paramètres.

Le travail avec une fonction se fait en deux temps:

1. La **déclaration de la fonction**, où l'on spécifie les instructions qui seront exécutées à chaque appel (à faire une seule fois, en général au début du code)
2. L'**appel de la fonction** à chaque fois qu'on en a besoin.

Pour déclarer une fonction, on fournit les informations suivantes :

- Un nom (qui permettra de l'appeler dans le code)
- Une série de paramètres (ou arguments, qui pourront être spécifiés par l'utilisateur)
- Une série d'instructions à exécuter à chaque appel de la fonction

Les fonctions

Elle se déclare de la manière suivante :

Syntaxe:

```
définir la fonction nommée "afficher_le_menu",
  qui prend pour argument(s) (jour_de_la_semaine) et qui fait :
    instructions
```

et elle s'appelle en tapant simplement :

```
afficher_le_menu(argument)
```

Note : les variables définies au sein d'une fonction sont appelées **variables locales**, elles n'existent pas en dehors de la fonction.

Les fonctions

R

```
## Déclaration
afficher_menu = function(jour) {
  if (jour=="lundi") {
    print("Pizza")
  } else if (jour=="mardi") {
    print("Pâtes")
  } else if (jour=="mercredi") {
    ...
  } else {
    print("Jour non reconnu.")
  }
}

## Appel
afficher_menu("lundi")
>>> Pizza
```

Python

```
## Declaration
def afficher_menu(jour):
    if jour=="lundi":
        print("Pizza")
    elif jour=="mardi":
        print("Pâtes")
    elif jour=="mercredi":
        ...
    else:
        print("Jour non reconnu.")

## Appel
afficher_menu("mardi")
>>> Pâtes
```

Les bonnes habitudes en programmation

- **Structurer son code** : importations au début, variables, puis fonctions, puis instructions. Modulariser en créant des fonctions réutilisables. Être explicite et cohérent dans le nom des variables, fonctions, etc.
- **Commenter son code (avec le #)** : décrire ce que fait chaque fonction (pas comment elle le fait), chaque bloc de code (pas chaque ligne)
- **S'assurer que son code fonctionne**: écrire des "tests" (petites fonctions qui vérifient que la méthode fonctionne sur des données où le résultat est connu)

À l'aide ! ▫

Quelques méthodes pour résoudre des bugs ou obtenir de l'aide.

Ne pas paniquer : "Déboguer" fait partie du travail de programmation, il est important d'attaquer les problèmes avec méthode.

1. Lire (et comprendre) le message d'erreur (si il y en a un). Le vrai message d'erreur correspond à la dernière ligne du bloc d'erreur (il faut donc aller tout en bas des erreurs pour comprendre le vrai problème). La ligne du code qui cause l'erreur est alors indiquée. Parfois, l'erreur vient de la ligne du dessus dans le code et le message d'erreur indique surtout où l'erreur survient.
2. Savoir avec quoi l'on travaille :
   - quel est le **type** des objets utilisés dans la ligne pose problème,
   - quel type est attendu par les fonctions utilisées ?
   - par exemple : diviser un nombre par une chaine de caractères ne peut pas fonctionner
3. Apprendre à chercher et lire la documentation technique

À l'aide ! ◻

Les messages d'erreur.

```
total = 1+a
>>> NameError   Traceback (most recent call last)
>>> <ipython-input-3-1b746bb122bc> in <module>()
>>> ----> 1 b = 1+a
>>>
>>> NameError: name 'a' is not defined

total = 1+"a"
>>> TypeError    Traceback (most recent call last)
>>> <ipython-input-5-400c44fdc315> in <module>()
>>> ----> 1 total = 1+"a"
>>>
>>> TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

À l'aide ! ◻

Les messages d'erreur.

```
k=4
if k==4:
    print ("Chouette"
    k=k+1

>>> File "<ipython-input-9-20a763b9c892>", line 4
>>>   k=k+1
>>>    ^
>>> SyntaxError: invalid syntax
```

À l'aide ! ▫

La documentation.

Documentation de référence: décrit le comportement (entrées et sorties) pour chaque fonction. Elle est accessible depuis l'interpréteur. Elle indique précisément comment utiliser chaque fonction, souvent avec des **exemples**.

- En `R`: commande `?`, par exemple : `?sum`
- En `Python`: fonction `help`, par exemple : `help(sum)`

Exemple en `R`: `?sum`

```
Sum of Vector Elements
Description: 'sum' returns the sum of all the values present in its arguments.
Usage: sum(..., na.rm = FALSE)
Arguments:
      ...: numeric or complex or logical vectors.
   na.rm: logical.  Should missing values (including 'NaN') be removed? [...]
Value: The sum. [...]
Examples: sum(1:5) ## Pass a vector to sum, and it will add the elements together.
```

À l'aide ! ▫

La documentation.

Dans Jupyter: quand le curseur se trouve sur un des arguments d'une fonction, la commande `Maj + Tab` affiche la documentation de la fonction.

Sur internet: la même documentation est aussi accessible, ainsi que des tutoriaux parfois plus simples d'accès que la documentation.

Pour un module externe: trouver la documentation du module, en général sur son site internet.