Traitement de Données

Clé Moodle : 27170 Mot de passe : TraitDon

Marc LEGEAY (marc.legeay@univ-angers.fr)

Année universitaire 2023-2024

Licence 3 Informatique - Université d'Angers

FACULTÉ
DES SCIENCES
UNIVERSITÉ D'ANGERS

Organisation du cours

- Cours
  - 6h40 de CM (5 séances)
  - 12h de TP
- Évaluation
  1. 1h20 en fin de période (questions de cours et TP) [2/3 de la note]
  2. des rendus de TP [1/3 de la note]

Chapitre 1

Introduction

Big Data

Définition par les « 3V » :

- Variété : Des données variées ...
- Vitesse : ... collectées de plus en plus rapidement ...
- Volume : ... dans des volumes importants.

Auxquels on pourrait rajouter :

- Valeur : Que devons-nous stocker ? Où se trouve la valeur dans les données ?
- Véracité : Les données collectées sont-elles vraies ?

Big Data - Quelques chiffres

- Environ 2,5 Exaoctets (10^{18}) de données sont créés chaque jour
- Environ 97 Zettaoctets (10^{21}) auraient été générés en 2022
- Environ 1,7 Megaoctets (10^{6}) de données est généré par seconde par un utilisateur d'internet

Temps nécessaire pour télécharger les données avec une fibre 2 Gbits/s :

- 1,7 Mo $\Rightarrow$ 0,0068 s
- 2,5 Eo $\Rightarrow$ 1 $\times 10^{10}$ s $\approx$ 700 ans
- 97 Zo $\Rightarrow$ 3,88 $\times 10^{14}$ s $\approx$ 12 294 978 ans

Data Science

- Science des données : comment extraire de l'information de données brutes ?
- Inter-disciplinaire :
  - inférence de données (logique, apprentissage)
  - mathématiques, statistiques
  - intelligence artificielle
  - traitement automatique du langage naturel (Natural Language Processing)

Data Science - Fonctionnement

1. Acquisition des données
2. Stockage des données (Data Warehouse)
3. Interrogation des données (Data Mining)
4. Analyse (Data Analysis)
5. Visualisation (Data Vizualisation)

Traitement de Données

Clé Moodle : 27170 Mot de passe : TraitDon

Marc LEGEAY (marc.legeay@univ-angers.fr)

Année universitaire 2023-2024

Licence 3 Informatique - Université d'Angers

FACULTÉ
DES SCIENCES
UNIVERSITÉ D'ANGERS

Chapitre 2

Données

Encodage des données

Sources de données

Traitement des données

Encodage des données

Encodage des données

Le *codage des caractères* est la manière dont un caractère est transformé.

Historique des encodages

- Signaux maritimes, Morse
- Codage par séquence de bits
- 1971 : ASCII, 128 signes codés sur 7 bits
- 1986 : ISO 8859-1 (latin1), 191 signes codés sur 8 bits
- 1996 : UTF-8, plusieurs millions de signes codés sur 1 à 4 octets

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[END OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

American Standard Code for Information Interchange

- Codé sur 1 octet (7 bits, le 8ème bit est toujours 0)
- Permet de représenter des caractères visibles (lettres, chiffres, symboles) du langage anglais
- Permet de représenter des caractères non-visibles (null, sauf à la ligne, retour chariot, ...)

American Standard Code for Information Interchange

- Codé sur 1 octet (7 bits, le 8ème bit est toujours 0)
- Permet de représenter des caractères visibles (lettres, chiffres, symboles) du langage anglais
- Permet de représenter des caractères non-visibles (null, sauf à la ligne, retour chariot, ...)

Exemple

0x48 0x65 0x6C 0x6C 0x6F 0x20 0x57 0x6F 0x72 0x6C 0x64 0x21

Hello World !

Latin-1

ISO-8859-1																
	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Latin-1

- Codé sur 1 octet (8 bits)
- Englobe la table ASCII
- Rajoute les caractères accentués des langages d'Europe Occidentale (latins et germaniques)

Latin-1

- Codé sur 1 octet (8 bits)
- Englobe la table ASCII
- Rajoute les caractères accentués des langages d'Europe Occidentale (latins et germaniques)

Exemple

0x55 0x6E 0x69 0x76 0x65 0x72 0x73 0x69 0x74 0xE9

Université

UTF-8

- Codé sur 1 à 4 octets, les derniers bits (bits de poids forts) indiquent le codage :
  - `0` : 1 seul octet (rétro-compatibilité avec ASCII)
  - `110` : 2 octets ; `1110` : 3 octets ; `11110` : 4 octets

  ⇒ Dans le cas où le bit de poids fort est différent de 0 : la taille de la séquence de 1 (terminée par un 0, non compris dans la séquence) indique le nombre d'octets utilisés. Tous les autres octets démarreront par la séquence `10`
- Tous les caractères sont associés à un point de code.

  En UTF-8 les points de code vont de U+0000 à U+D7FF et de U+E000 à U+10FFFF

Exemples

	Point de code	Décimal	Représentation binaire
A	U+0041	65	`01000001` (0x41)
é	U+00E9	233	`11000011 10101001` (0xC3 0xA9)

Byte Order Mark

L'*Indicateur d'Ordre d'Octets* (BOM en anglais) est utile en UTF-16 et UTF-32, mais pas obligatoire en UTF-8.

Il est parfois utilisé en UTF-8 pour ne pas le confondre avec du latin-1.

- C'est un caractère invisible pour UTF
- C'est une suite de caractères visibles pour latin-1 : ï»¿
- Attention à la lecture du fichier à bien ignorer le BOM s'il est présent

Compatibilité

- Un fichier encodé en ASCII peut être ouvert en latin-1 ou UTF-8
- Un fichier encodé en latin-1, n'utilisant que des caractères ASCII, sera lisible en ASCII et UTF-8 ; s'il utilise des caractères accentués : aucune compatibilité
- Un fichier encodé en UTF-8, n'utilisant que des caractères ASCII, sera lisible en ASCII et latin-1 ; s'il utilise d'autres caractères : aucune compatibilité.

Exemple chaîne UTF-8

0x55 0x6E 0x69 0x76 0x65 0x72
0x73 0x69 0x74 0xC3 0xA9

Impossible en ASCII (0xC3)

UniversitÃ© (latin-1)

Université (UTF-8)

Exemple chaîne latin-1

0x55 0x6E 0x69 0x76 0x65 0x72
0x73 0x69 0x74 0xE9

Impossible en ASCII (0xE9)

Université (latin-1)

Universit (UTF-8)

Détection de l'encodage

Problème

L'encodage d'un fichier n'est pas indiqué dans les formats textes.

- Utilisation du BOM si présent
- Tous les bits de poids forts à 0 : ASCII
- Détection des octets en lisant l'ensemble du fichier
- Pour les scripts :
  - Donner l'encodage en entrée de la commande
  - Donner l'encodage au début du fichier

Attention

Un fichier ne pourra être lu qu'avec un seul encodage, mais écrit avec plusieurs.

Sources de données

Fichiers textes

- Texte brut
- CSV (Comma Separated Value)
- TSV (Tabulation Separated Value)
- XML (eXtended Markup Language)
- JSON (JavaScript Object Notation)
- YAML (Yet Another Markup Language)
- TOML (Tom's Obvious Minimal Language)

- Et tous les autres formats spécifiques ...
- FastA

Attention

L'encodage a toute son importance pour les fichiers textes !

Fichiers binaires

- XLSX (Excel) / ODS (Calc)
- ZIP / RAR / TAR.GZ / 7Z (Archives compressées)
- SQLITE (Base de données)

- Et tous les autres formats spécifiques ...
- DAT (Data)

Encodage

Théoriquement l'encodage est géré directement par l'application qui lit les données et pose moins de problèmes.

Bases de données

SGBD connus :

- Oracle
- PostgreSQL
- Microsoft SQL Server
- MySQL / MariaDB
- SQLite

Attention

L'encodage dans une base de données peut être définie **par champ** !
Il est aussi défini globalement.

Traitement des données

Types de données

- Données quantitatives $\Rightarrow$ Statistiques
- Données qualitatives $\Rightarrow$ Statistiques, TALN
- Texte brut $\Rightarrow$ TALN, Statistiques

Données quantitatives

Les données quantitatives sont des données de quantité, finie ou infinie, représentée par une valeur numérique. Par exemple, la taille d'une personne.

Opérations possibles (entre autres) :

- Comptage
- Somme
- Moyenne
- Médiane

Données qualitatives

Les données qualitatives sont des données qui qualifient le sujet. Il existe deux types de données qualitatives : les qualitatives *ordinales* et *non-ordinales*.

Une donnée qualitative ordinale est une qualité dont les valeurs peuvent être ordonnées. Par exemple, le stade d'une maladie.

Une donnée qualitative non-ordinale est une qualité dont les valeurs ne peuvent être ordonnées. Par exemple, la catégorie socio-professionnelle d'une personne.

Données qualitatives - Opérations

Opérations possibles (entre autres) :

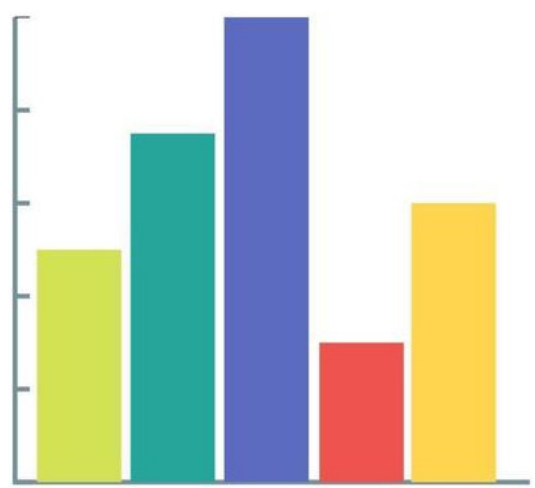
- Comptage
- Somme cumulée (si ordinale)
- Médiane
- Fréquence
- Fréquence cumulée (si ordinale)

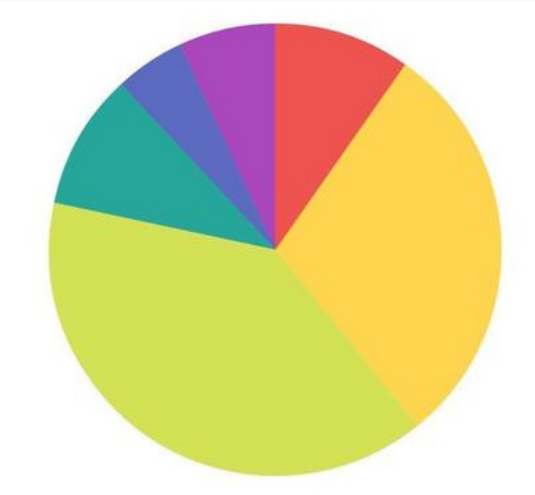
Opérations impossibles (entre autres) :

- Somme
- Moyenne

Affichage des résultats

- Données quantitatives :
  - Tableau de données
  - Nuages de points, lignes
  - Histogrammes
  - Boîtes à moustaches
  - Toiles
- Données qualitatives non-ordonales :
  - Tableau de données, trié par effectif
  - Diagramme à barres
  - Diagramme circulaire
- Données qualitatives ordonales :
  - Tableau de données, trié par ordre
  - Diagramme à barres
  - Diagramme circulaire





Texte brut

Pour exploiter du texte brut il existe différentes méthodes :

- On connaît la donnée à extraire, et on utilise des méthodes de recherche
- On ne connaît pas la donnée à extraire, et on l'extrait avec des méthodes de *Traitement Automatique du Langage Naturel* (TALN)

Le TALN s'intéresse à :

- Syntaxe
- Sémantique
- Reconnaissance (vocale et graphique)
- Extraction

Exemples de TALN

- Syntaxe : Identifier les lemmes.
  *Tu **mets** sur la table de très bons **mets**!*
- Sémantique : Traduction automatique.
- Reconnaissance : Identifier les mots dans un homophone.
  « Aidé, j'adhère au quai ; lâche et rond je m'ébats.
  Et déjà, des roquets lâchés rongent mes bas. »

  — Vers holorime d'Alphonse Allais.
- Extraction : Fouille de texte (text-mining)

Traitement de Données

Clé Moodle : 27170 Mot de passe : TraitDon

Marc LEGEAY (marc.legeay@univ-angers.fr)

Année universitaire 2023-2024

Licence 3 Informatique - Université d'Angers

FACULTÉ
DES SCIENCES
UNIVERSITÉ D'ANGERS

Chapitre 3

Langages

Bash

Aho, Weinberger et Kernighan

R

Python

Bash

Bash

Aho, Weinberger et Kernighan

R

Python

Bourne-Again shell

Les commandes shell sont utiles pour avoir un *aperçu* des données.

Les commandes shell peuvent être utilisées pour pré-traiter les données : les formater à volonté.

Répétabilité : Afin d'effectuer une tâche répétitive, autant créer des scripts. En plus, cela vous permettra de garder une trace des traitements effectués sur les données.

Commandes utiles au traitement de fichiers

`file`	Permet de connaître l'encodage d'un fichier
`iconv`	Permet de convertir un fichier
`cat`	Permet de concaténer des fichiers
`cut`	Permet d'extraire/réarranger des colonnes
`grep`	Permet d'extraire des lignes
`tr`	Permet de remplacer/supprimer un caractère
`sed`	Permet de chercher/remplacer des caractères
`awk`	Permet de traiter un fichier texte

Encodage de fichiers

La commande **`file`** va détecter l'encodage d'un fichier.

La commande **`iconv`** va lire un fichier à partir d'un encodage en entrée, et le convertir pour l'encodage de sortie défini.

Exemple dans un terminal en UTF-8

```
user@host:~$ echo "Université" > original.txt
user@host:~$ file original.txt
original.txt: UTF-8 Unicode text
user@host:~$ iconv -f utf8 -t latin1 -o latin1.txt ori
ginal.txt
user@host:~$ file latin1.txt
latin1.txt: ISO-8859 text
user@host:~$ cat latin1.txt
Universit?
```

Concaténation de fichier

La commande `cat` permet de voir le contenu d'un fichier, mais sert originalement à concaténer deux fichiers.

Exemple dans un terminal en UTF8

```
user@host:~$ printf '\xEF\xBB\xBFA\n' > utf8_bom.txt
user@host:~$ printf '\xEF\xBB\xBFB\n' > utf8_bom2.txt
user@host:~$ file utf8*
utf8_bom2.txt: UTF-8 Unicode (with BOM) text
utf8_bom.txt:  UTF-8 Unicode (with BOM) text
user@host:~$ cat utf8* > utf8_bom_cat.txt
user@host:~$ file utf8_bom_cat.txt
utf8_bom_cat.txt: UTF-8 Unicode (with BOM) text
user@host:~$ cat utf8_bom_cat.txt
B
A
```

Extraire des colonnes

La commande `cut` permet de découper des lignes en colonnes, puis de les extraire dans un ordre donné.

Exemple : Extraire l'âge des personnes

```
user@host:~$ cat data_fr.csv
Prénom;Nom;Âge
Hector;Gounelle;22
Jérôme;Bessette;33
Marcel;Baschet;26
user@host:~$ cut -d\; -f3 data_fr.csv
Âge
22
33
26
```

Extraire des lignes

La commande `grep` permet d'extraire des lignes d'un fichier.

Quelques options utiles :

- `-i` ignore la casse
- `-n` affiche les numéros de lignes
- `-c` donne le nombre de lignes qui correspondent
- `-v` inverse le pattern
- `-l` liste les fichiers qui ont au moins une correspondance

Exemple : Supprimer l'en-tête du fichier CSV

```
user@host:~$ grep -iv "âge" data_fr.csv
Hector;Gounelle;22
Jérôme;Bessette;33
Marcel;Baschet;26
```

Remplacer des caractères - `tr`

La commande `tr` permet de « traduire » (remplacer) un caractère par un autre. Il est possible également de remplacer plusieurs caractères par un autre.

Exemple : CSV français vers CSV anglais

```
user@host:~$ cat data_fr.csv
Prénom;Nom;Âge
Hector;Gounelle;22
Jérôme;Bessette;33
Marcel;Baschet;26
user@host:~$ tr ';' ',' < data_fr.csv
Prénom,Nom,Âge
Hector,Gounelle,22
Jérôme,Bessette,33
Marcel,Baschet,26
```

Remplacer des caractères - `tr`

La commande **`tr`** permet de « traduire » (remplacer) un caractère par un autre. Il est possible également de remplacer plusieurs caractères par un autre.

Exemple : minuscules vers majuscules

```
user@host:~$ cat data_fr.csv
Prénom;Nom;Âge
Hector;Gounelle;22
Jérôme;Bessette;33
Marcel;Baschet;26
user@host:~$ tr 'a-z' 'A-Z' < data_fr.csv
PRéNOM;NOM;ÂGE
HECTOR;GOUNELLE;22
JéRôME;BESSETTE;33
MARCEL;BASCHET;26
```

Remplacer des caractères - `sed`

La commande `sed` permet d'éditer (modifier) un flux. La plupart du temps elle est utiliser pour substituer des caractères (commande `s/cherche/remplace`), mais il existe plusieurs autres commandes, et il est possible d'écrire des script `sed`.

Commandes	Explications
`a \`	Ajouter du texte après la ligne
`i \`	Ajouter du texte avant la ligne
`d`	Supprime la ligne
`s`	Substitution

Remplacer des caractères - sed

Exemple : script.sed

```
# Ajouter Jean Dupont 33 ans, après la 1ère ligne
1 a \
Jean;Dupont;33

# Supprimer les lignes 3 à la fin
3,$ d
```

```
u@h:~$ cat data_fr.csv
Prénom;Nom;Âge
Hector;Gounelle;22
Jérôme;Bessette;33
Marcel;Baschet;26
Roch;Bossuet;5
```

```
u@h:~$ sed -f script.sed data
_fr.csv
Prénom;Nom;Âge
Jean;Dupont;33
Hector;Gounelle;22
```

Traitement de Données

Clé Moodle : 27170 Mot de passe : TraitDon

Marc LEGEAY (marc.legeay@univ-angers.fr)

Année universitaire 2023-2024

Licence 3 Informatique - Université d'Angers

FACULTÉ
DES SCIENCES
UNIVERSITÉ D'ANGERS

Chapitre 3

Langages

Bash

Aho, Weinberger et Kernighan

R

Python

Aho, Weinberger et Kernighan

Bash

Aho, Weinberger et Kernighan

R

Python

AWK

AWK est un langage qui a été développé par Alfred Aho, Peter Weinberger et Brian Kernighan, d'où le nom du langage.

C'est un langage de script qui permet de manipuler des flux textuels.

AWK découpe le flux en enregistrements (*records*) et champs (*fields*). Un enregistrement est, de base, une ligne. Un champ est, de base, un mot.

Les séparateurs d'enregistrement et de champs peuvent être modifiés.

Condition - action

C'est un langage *condition - action*. Il permet de faire des actions lorsque les conditions sont réunies. La structure est simple :

Structure de base d'un script AWK

```
condition1 {
  action1.1
  action1.2 ; action 1.3
  ...
  action1.n
}

...
```

Conditions

- Une expression booléenne
  - Utilisation des opérateurs booléens (`&&`, `||`, `!`, `==`, `<=`, ...)
  - Utilisation des opérateurs arithmétiques (`+`, `-`, `/`, `*`, `%`)
- Filtrage via une expression booléenne
  - Syntaxe : `variable ~ / regex /`
  - Vrai si la variable `variable` correspond avec l'expression régulière `regex`
  - Absence de `variable ~` : vrai si la ligne correspond avec l'expression régulière
  - Expression régulière :
    - `*` : répétition 0, 1 ou plusieurs fois
    - `?` : répétition 0 ou 1 fois
    - `+` : répétition 1 ou plusieurs fois
    - `[]` : un caractère parmi
    - `^` : début de chaîne
    - `$` : fin de chaîne
    - `.` : n'importe quel caractère

Conditions (suite)

2 mots-clefs permettent de définir une condition particulière :

- `BEGIN` : Permet de définir des actions à réaliser avant le traitement des enregistrements
- `END` : Permet de définir des actions à réaliser après le traitement des enregistrements

Déclenchement des conditions

Lorsqu'une condition est vérifiée, elle ne stoppe pas l'exécution du script (toutes les conditions vérifiées par l'enregistrement seront donc activées).

Les actions sont exécutées dans l'ordre du script.

Actions

Les actions sont définies par du code (semblable au C).

Une instruction par ligne, ou alors les instructions doivent être séparées par un point-virgule.

Possibilité de faire des conditions (`if`, `else`) ou des boucles (`while`, `do ... while`, `for(;;)` et `for(v in t)`)

Utilisation de fonctions prédéfinies :

- `print` et `printf` pour l'affichage
- `length(s)` pour obtenir la taille de la chaîne `s`
- `substr` pour extraire une sous-chaîne
- fonctions mathématiques (`cos`, `abs`, ...)

Variables

Les variables en AWK sont **globales**.

Les variables en AWK ne sont pas typées.

ATTENTION : Valeur par défaut

Les variables n'ont pas besoin d'être déclarées en amont.

Par défaut, une variable aura la valeur 0 ou "".

Attention au fautes de frappes !

Les tableaux peuvent être associatifs.

Bonne pratique

Initialiser les variables dans un bloc `BEGIN`.

Variables système

`NR` - Number Record	Numéro de l'enregistrement global
`FNR` - File Number Record	Numéro de l'enregistrement local au fichier
`NF` - Number of Fields	Nombre total de champs
`FS` - Field Separator	Séparateur de champs (défaut : espace et tabulation)
`RS` - Record Separator	Séparateur d'enregistrements (défaut : nouvelle ligne)
`FILENAME`	Nom du fichier
`ARGC`	Nombre d'arguments passés à la commande
`ARGV`	Tableau des arguments passés à la commande
`$i` - `i` $\in$ [0;`NF`]	valeur du `i`-ème champs. `$0` est toute la ligne

Commandes

- `gawk 'programme' fichiers`
  Exemple : `gawk 'FNR==1 {print FILENAME}' *.txt`
- `gawk -f script fichiers`
  Exemple : `gawk -f afficheNomFichiers.awk *.txt`
- Il est possible de donner le séparateur de champ à la commande avec l'option `-F separateur`

Exemples

userShells1.awk

```
 {
    # On affiche le nom de l'utilisateur (1er champ)
    # et son shell (7ème champ)
    print $1, $7
}
```

```
gawk -F: -f userShells1.awk /etc/passwd
```

Exemples

userShells1.awk

```
 {
    # On affiche le nom de l'utilisateur (1er champ)
    # et son shell (7ème champ)
    print $1, $7
}
```

```
gawk -F: -f userShells1.awk /etc/passwd
```

Question bonus

Quelle autre commande permet d'obtenir le même résultat ?

Exemples

userShells1.awk

```
 {
    # On affiche le nom de l'utilisateur (1er champ)
    # et son shell (7ème champ)
    print $1, $7
}
```

```
gawk -F: -f userShells1.awk /etc/passwd
```

Question bonus

Quelle autre commande permet d'obtenir le même résultat ?

```
cut -d: -f1,7 --output-delimiter=' ' /etc/passwd
```

Exemples

userShells2.awk

```
 {
    # On affiche le nom de l'utilisateur (1er champ)
    # et son shell (7ème champ)
    print $1, $7
}
BEGIN {
    # On s'assure que le séparateur de champs est :
    FS=':'
}
```

```
gawk -f userShells2.awk /etc/passwd
```

Exemples

functions.awk

```
/^\s*function \w/ {
    nom[nb] = $2 ; fic[nb] = FILENAME
    ligne[nb] = FNR ; nb++
}
END {
    print nb, "fonctions ont été identifiées"
    for(i=0;i<nb;++i) {
        print nom[i], fic[i], ligne[i]
    }
}
```

```
gawk -f functions.awk *.php
```

Traitement de Données

Clé Moodle : 27170 Mot de passe : TraitDon

Marc LEGEAY (marc.legeay@univ-angers.fr)

Année universitaire 2023-2024

Licence 3 Informatique - Université d'Angers

FACULTÉ
DES SCIENCES
UNIVERSITÉ D'ANGERS

Chapitre 3

Langages

Bash

Aho, Weinberger et Kernighan

R

Python

R

Bash

Aho, Weinberger et Kernighan

R

Python

R

- Langage et environnement pour des traitements statistiques
- Fait parti du GNU Project
- Implémentation/Amélioration du langage S

- Permet de faire du calcul vectoriel et matriciel
- Permet de générer des graphiques
- Peut être étendu par des packages

- Implémente le paradigme objet
- Implémente le paradigme fonctionnel

Exécutions

Pour lancer un environnement R, il faut simplement utiliser la commande : `R`

Cela chargera automatiquement le dernier environnement utilisé (`.RData` et `.Rhistory`).

Lorsqu'on quitte un environnement R (commande `q()`), on nous demande si on souhaite sauvegarder l'environnement.

R est également un langage de script, on peut donc simplement exécuter un script R :

```
Rscript script.r
```

Dans une console R, pour exécuter un script R il suffit d'utiliser la commande :

```
source("script.r")
```

Environnement

- Espace de stockage des variables et fonctions
- Interactif
- `ls()` liste toutes les variables et fonctions de l'environnement
- `ls.str()` liste toutes les variables et fonctions de l'environnement avec leur types et valeurs
- `rm()` supprime une variable ou fonction de l'environnement

Nom des objets (variables, fonctions)

Les noms d'objets peuvent comporter des lettres, chiffres, points et underscores.

Le nom doit commencer par une lettre.

Les noms sont sensibles à la casse.

Aide

- RDocumentation (`https://www.rdocumentation.org/`)
- Depuis l'environnement :
  - `?commande`
    (par exemple `?ls` pour l'aide sur la commande `ls`)
  - `help(commande)`
    (par exemple `help(ls)` pour l'aide sur la commande `ls`)
  - `help("commande")`
    (par exemple `help("ls")` pour l'aide sur la commande `ls`)
- `fct` (sans parenthèses) affiche le code de la fonction `fct`
  (par exemple `ls` affiche le code de la fonction `ls`)

Affectation

L'affectation d'une variable se fait avec l'opérateur `<-` ou `->`.

`variable <- expression` ou `expression -> variable`

L'opérateur `=` peut également être utilisé, il sera alors l'équivalent de l'opérateur `<-`. L'opérateur `=` est utilisé principalement dans d'autres cas.

Exemples

```
a <- 3 + 2

3 * 5 -> b

cube = 2**3
```

Fonctions

Les fonctions sont des objets R spécifiques, ils sont définis par le mot-clef `function`.

Les fonctions sont toutes **anonymes**. Afin de pouvoir les appeler, il faut les affecter à une variable.

Exemple

```
bonjour <- function() { print("bonjour") }
```

Séparateur d'instructions

Vu que R est un langage qui est utilisé dans une console, les instructions sont séparées par des *sauts de lignes*.

Si on souhaite mettre plusieurs instructions sur une même ligne, il faut les séparer par un *point-virgule*.

Arguments de fonction

Les arguments de fonctions sont nommés mais non typés.

Ils peuvent avoir une valeur par défaut : `nom=valeur`

Ils doivent être appelés dans l'ordre de leur déclaration, ou bien ils peuvent être nommés :

Exemple

```
test <- function(a, b=2, c) {
    print(a, b, c)
}
test("a", "b", "c") # Affiche : "a" "b" "c"
test(c=3, a=1) # Affiche : 1 2 3
```

Valeur de fonction

La valeur d'une fonction est celle de la dernière ligne de la fonction.

Exemple

```
double <- function(v) {
    2*v
}
double(3) # 6
```

On peut aussi utiliser le mot-clef `return` pour retourner une valeur.

Exemple

```
double2 <- function(v) {
    return (2*v)
}
double2(3) # 6
```

Affichage

Pour afficher la valeur d'un objet, il suffit de taper le nom de cet objet. Par défaut, R fera appel à la fonction `print()` pour afficher l'objet.

Dans une fonction, le simple appel à un objet ne l'affichera pas. Pour forcer l'affichage d'une valeur, on peut utiliser directement la fonction `print()`.

Affichage de `print()`

`print()` indexe le premier élément de la ligne affichée.

Exemple :

```
> vecteur <- c("bonjour", "tout", "le", "monde")
> vecteur
[1] "bonjour" "tout"
[3] "le" "monde"
```

Objet

Tous les objets ont un *mode* et une *taille*.

Le *mode* (`mode()`) d'un objet définit s'il contient une valeur numérique, caractère, numérique complexe, logique, fonction ou liste.

La *taille* (`length()`) d'un objet représente le nombre de valeurs stockées dans l'objet. Attention, une chaîne de caractères est une valeur et non plusieurs. Pour connaître la taille d'une chaîne de caractères, il faut utiliser la fonction `nchar()`.

Valeurs spécifiques :

- `NA` - Not Assigned. Valeur non assignée.
- `NaN` - Not a Number. Valeur non-numérique.
- `TRUE` / `FALSE`
- `-Inf` / `Inf`

Objets spéciaux

- `vector` - Un tableau à 1 dimension
- `factor` - Un facteur est une catégorie (donnée quantitative)
- `array` - Un tableau à plusieurs dimensions
- `matrix` - Un tableau à 2 dimensions
- `data.frame` - Un tableau de données (peut contenir des données de modes différents)
- `list` - Une liste d'objets qui peut contenir tout objet

Paquets

Par défaut, le paquet `base` est chargé. Il contient l'ensemble des fonctions et définition de classes nécessaires au bon fonctionnement de R.

- `install.packages()` permet d'installer un paquet sur votre ordinateur
- `library()` permet de charger un paquet installé
- `installed.packages()` liste tous les paquets installés
- `require()` permet de charger un paquet installé, retourne un booléen

Les paquets permettent de définir de nouvelles classes et de nouvelles fonctions.

Paquets

Avec `installed.packages()`

```
if(!("superPaquet" %in% installed.packages())) {
    install.packages("superPaquet")
}
library("superPaquet")
```

Avec `require()`

```
if(!require("superPaquet")) {
    install.packages("superPaquet")
    library("superPaquet")
}
```

Lecture/Écriture de fichiers texte

- Fonctions
  - `read.table()` / `read.csv()` / `read.csv2()`
  - `write.table()` / `write.csv()` / `write.csv2()`
- Arguments
  - `header` - le fichier contient-il une première ligne d'en-tête ?
  - `sep` - séparateur de champs
  - `quote` - caractères utilisés pour définir une chaîne (lecture) / entourer les chaînes de guillemets doubles ? (écriture)
  - `dec` - séparateur de décimales
  - `as.is` - les chaînes sont-elles des facteurs (`FALSE`) ou des chaînes (`TRUE`) ?
  - `comment.char` - caractère de commentaires

Lecture/Écriture de fichiers Excel

- R ne peut pas lire les fichiers Excel dans le paquet `base`
- Plusieurs paquets existent pour le faire :
  - `gdata` avec `read.xls()` (ne supporte pas XLSX)
  - `openxlsx` avec `read.xlsx()` (ne supporte pas XLS)
  - `readxl` avec `read_excel()`
  - `xlsx` avec `read.xlsx()` (ne supporte pas XLS)

Sauvegarde/Chargement/Redirection

- `save(..., file=fic.RData)` : sauvegarde les objets en arguments dans le fichier `fic.RData`
- `load(fic.RData)` : charge les objets contenus dans le fichier `fic.RData`
- `sink(fic.txt)` permet de rediriger la sortie de la console vers le fichier `fic.txt`
- `sink()` permet de rediriger la sortie de la console vers la console (arrête une précédente redirection)

Générer des données connues

- Séquence : `debut:fin` (exemples : `1:10` génère la suite de 1 à 10 ; `3:-2` génère la suite de 3 à -2)
- Séquence : `seq(d,f,p)` génère une séquence de `d` à `f` avec un pas de `p`
- Combiner : `c()` combine tous les arguments en un vecteur (exemple : `c(3,1,5)`)
- Lire au clavier : `scan()`. Ne lit que des numériques.
- Répéter : `rep(v, t)` crée un vecteur de taille `t` contenant la valeur `v`

Générer des données aléatoires

Il existe plusieurs fonctions de générations aléatoires de données, selon la loi de probabilité.

Elles sont toutes nommées `rprob(n, p1, p2, ...)` où `prob` est le nom de la loi de probabilité, `n` la taille de l'échantillon et $\texttt{p}_i$ les paramètres de la loi.

Quelques exemples :

Normale	`rnorm(n, mean=0, sd=1)`
Poisson	`rpois(n, lambda)`
Student	`rt(n, df)`
χ^2	`rchisq(n, df)`
Binomiale	`rbinom(n, size, prob)`
Uniforme	`runif(n, min=0, max=1)`

R vectoriel

R est un langage **vectoriel**, c'est-à-dire qu'il implémente les opérateurs sur les vecteurs.

Console R

```
> 1:3*2
[1] 2 4 6
> 1:3*1:6
[1] 1 4 9 4 10 18
```

Explications :

- `1:3` crée le vecteur `1 2 3` auquel est appliqué la multiplication
- `1:6` crée le vecteur `1 2 3 4 5 6`. Le vecteur `1:3` n'a pas la bonne taille pour appliquer la multiplication. Mais sa taille est un multiple de `1:6` et donc est répété : `1 2 3 1 2 3`.

Manipulation des données

Les vecteurs, matrices et tableaux sont indicés de 1 à n et l'accès se fait via les crochets `[]`. Les matrices et tableaux sont indicés dans l'ordre de leur dimension (`[dim1, dim2, ... ]`).

Les tableaux de données (`data.frame`) sont indicés de 1 à n. Les colonnes sont nommées et peuvent être accédées grâce au nom. `df[3, "age"]` permet ainsi d'accéder à la troisième ligne de la colonne "age" du tableau de données `df`.

Les listes sont indicées et l'accès se fait via les double-crochets `[[]]`. `maListe[[2]]` est ainsi le deuxième élément de la liste.

Les listes peuvent être nommées, l'accès se fera ainsi grâce au dollar `$` et le nom de l'élément : `maListe$monElement`.

On peut donner un vecteur d'indices pour accéder à plusieurs données (sauf pour les listes).

Exemples de manipulations - Vecteurs, Matrices

```
> x <- 5:14
> y <- 1:5
```

Exemples de manipulations - Vecteurs, Matrices

```
> x <- 5:14
> y <- 1:5
> x[3]
```

Exemples de manipulations - Vecteurs, Matrices

```
> x <- 5:14
> y <- 1:5
> x[3]
[1] 7
> x[1:3]
```

Exemples de manipulations - Vecteurs, Matrices

```
> x <- 5:14
> y <- 1:5
> x[3]
[1] 7
> x[1:3]
[1] 5 6 7
> y[-1]
```

Exemples de manipulations - Vecteurs, Matrices

```
> x <- 5:14
> y <- 1:5
> x[3]
[1] 7
> x[1:3]
[1] 5 6 7
> y[-1]
[1] 2 3 4 5
> matrix(x, nrow=2)[1,]
```

Exemples de manipulations - Vecteurs, Matrices

```
> x <- 5:14
> y <- 1:5
> x[3]
[1] 7
> x[1:3]
[1] 5 6 7
> y[-1]
[1] 2 3 4 5
> matrix(x, nrow=2)[1,]
[1] 5 7 9 11 13
> matrix(x, nrow=2)[,3]
```

Exemples de manipulations - Vecteurs, Matrices

```
> x <- 5:14
> y <- 1:5
> x[3]
[1] 7
> x[1:3]
[1] 5 6 7
> y[-1]
[1] 2 3 4 5
> matrix(x, nrow=2)[1,]
[1] 5 7 9 11 13
> matrix(x, nrow=2)[,3]
[1] 9 10
> matrix(x, nrow=2)[1,3]
```

Exemples de manipulations - Vecteurs, Matrices

```
> x <- 5:14
> y <- 1:5
> x[3]
[1] 7
> x[1:3]
[1] 5 6 7
> y[-1]
[1] 2 3 4 5
> matrix(x, nrow=2)[1,]
[1] 5 7 9 11 13
> matrix(x, nrow=2)[,3]
[1] 9 10
> matrix(x, nrow=2)[1,3]
[1] 9
```

Exemples de manipulations - Tableaux de données

```
> data.frame(x, y)
    x y
1   5 1
2   6 2
3   7 3
4   8 4
5   9 5
6  10 1
7  11 2
8  12 3
9  13 4
10 14 5
> data.frame(x, y)[6:10,"y"]
```

Exemples de manipulations - Tableaux de données

```
> data.frame(x, y)
    x y
1   5 1
2   6 2
3   7 3
4   8 4
5   9 5
6  10 1
7  11 2
8  12 3
9  13 4
10 14 5
> data.frame(x, y)[6:10,"y"]
[1] 1 2 3 4 5
```

Exemples de manipulations - Listes

```
> list(x, y)
[[1]]
[1]  5  6  7  8  9 10 11 12 13 14
[[2]]
[1] 1 2 3 4 5
> list(x, y)[[1]][3]
```

Exemples de manipulations - Listes

```
> list(x, y)
[[1]]
[1]  5  6  7  8  9 10 11 12 13 14
[[2]]
[1] 1 2 3 4 5
> list(x, y)[[1]][3]
[1] 7
> list(X=x, Y=y)
$X
 [1]  5  6  7  8  9 10 11 12 13 14
$Y
[1] 1 2 3 4 5
> list(X=x, Y=y)$X[3]
```

Exemples de manipulations - Listes

```
> list(x, y)
[[1]]
[1]  5  6  7  8  9 10 11 12 13 14
[[2]]
[1] 1 2 3 4 5
> list(x, y)[[1]][3]
[1] 7
> list(X=x, Y=y)
$X
 [1]  5  6  7  8  9 10 11 12 13 14
$Y
[1] 1 2 3 4 5
> list(X=x, Y=y)$X[3]
[1] 7
```

Graphiques

Le paquet `base` de R permet de générer des graphiques avancés.

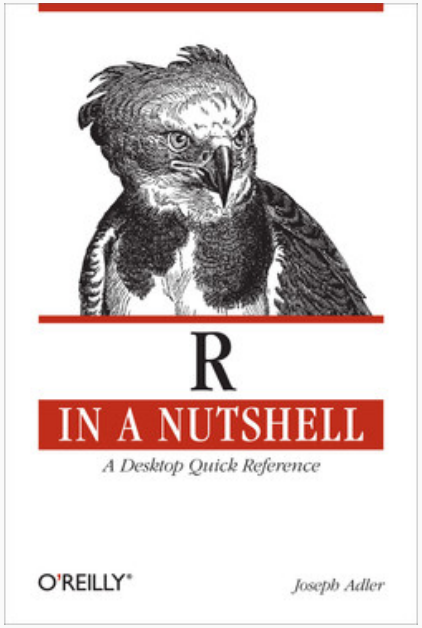
Plusieurs paquets ont été développés par la communauté pour manipuler plus facilement les graphiques ou pour en produire des plus jolis.

Exemples de fonctions graphiques :

- `plot` - nuage de points
- `pie` - graphique circulaire
- `boxplot` - boîtes à moustaches
- `hist` - histogramme de fréquences
- `barplot` - diagramme en barres

Pour aller plus loin ...

- R pour les débutants - Emmanuel Paradis (`https://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf`)
- R in a Nutshell - Jospeh Adler (`https://www.oreilly.com/library/view/r-in-a/9781449377502/`)
- Learning R - Richard Cotton (`https://www.oreilly.com/library/view/learning-r/9781449357160/`)



Traitement de Données

Clé Moodle : 27170 Mot de passe : TraitDon

Marc LEGEAY (marc.legeay@univ-angers.fr)

Année universitaire 2023-2024

Licence 3 Informatique - Université d'Angers

FACULTÉ
DES SCIENCES
UNIVERSITÉ D'ANGERS

Chapitre 3

Langages

Bash

Aho, Weinberger et Kernighan

R

Python

Python

Python

Généralités

Python

- Langage et environnement pour des traitements de données
- Supporte le parallélisme

- Permet de générer des graphiques
- Peut être étendu par des modules

- Implémente le paradigme objet
- Implémente le paradigme fonctionnel

Exécutions

Pour lancer un environnement Python, il faut simplement utiliser la commande : `python`

Chaque environnement Python est indépendant. Il n'est pas possible de sauvegarder/charger un environnement.

Il est possible d'exécuter un script Python directement :

```
python script.py
```

Votre script peut également être un exécutable s'il débute par :

```
#!/usr/bin/python
```

Alors ensuite il pourra être exécuté avec `./script.py`.

Environnement

- `dir()` liste toutes les variables accessibles
- `locals()` renvoie un dictionnaire des variables locales (avec leurs valeurs)
- `globals()` renvoie un dictionnaire des variables globales (avec leurs valeurs)

Aide

- Documentation officielle (`https://docs.python.org/fr/3/contents.html`)
- La fonction `help()`
  - interactive : `help()`
  - aide sur la fonction `f` : `help(f)`
  - aide sur le module `m` : `help('m')`

Indentation

L'indentation en Python est **obligatoire** : elle permet de définir des blocs à l'instar des accolades dans d'autres langages de programmation.

À chaque fois que vous définissez un bloc, le contenu de ce bloc devra être précédé d'un niveau d'indentation supérieur. Le bloc débutera toujours par une instruction suivie de deux-points `:`.

Blocs

```
for key in dictionary:
    val = dictionary[key]
    if val<0:
        print(val)

        print("negatif")
    else:
        print("positif")
```

Indentation

Conseil

Finissez vos blocs par un commentaire !

blocs.py

```
for key in dictionary:
    val = dictionary[key]
    if val<0:
        print(val)

        print("negatif")
    else:
        print("positif")
    #end else (if val<0)
#end for
```

Instructions - particularités

- Conditions : `if`, `elif`, `else`
- Boucles : `while`, `for`
- Boucles sinon : `while ... else`, `for ... else`
  Le bloc `else` n'est exécuté qu'après l'exécution normale de la boucle (une sortie par `break` empêche l'exécution du `else`).

Boucle … sinon

prime.py

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'equals', x, '*', n//x)
            break
    else:
        # loop fell through without finding a factor
        print(n, 'is a prime number')
```

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

Fonctions

Les fonctions en Python sont définies à l'aide du mot-clef **`def`**.

La valeur de retour d'une fonction est définie par le mot-clef **`return`**.

Lorsque la valeur de retour est un n-uplet, il est possible de récupérer la valeur de chaque membre individuellement.

Exemple de retour d'un couple

```
def test():
    return (1,2)
#end test
(a,b) = test() # a=1 ; b=2
```

Arguments de fonctions

Les arguments d'une fonction doivent être appelés dans l'ordre dans lequel ils ont été définis. Il est possible de nommer les arguments pour les appeler, comme en R. De même, il est possible de donner des valeurs par défaut.

- Il est possible de créer une liste d'arguments : `*args`
- Il est possible de créer une liste d'arguments nommés : `**args`

Fonctions

```
def affiche_args(*args):
    for a in args:
        print(a)

def affiche_arg_foo(**args):
    print(args["foo"])

affiche_args('a', 2) # Affiche 'a' et 2
affiche_arg_foo(test=1, foo="World", inutile='') # Affiche "World"
```

Modules

Les **modules** sont des fichiers Python importés dans un script.

monModule.py

```
def helloWorld():
    print('Hello World!')

def uneFonction(n):
    return n
```

- `import monModule` : permet d'importer le module `monModule` avec l'espace de nom `monModule`;
- `from monModule import uneFonction` : permet d'importer la fonction `uneFonction` du module `monModule` dans l'espace de nom courant;
- `from monModule import *` : permet d'importer totalement le module `monModule` dans l'espace de nom courant;

Modules - Renommage

- `import monModule as mod` : permet d'importer le module `monModule` avec l'espace de nom `mod` ;
- `from monModule import uneFonction as identite` : permet d'importer la fonction `uneFonction` du module `monModule` dans l'espace de nom courant, en la renommant `identite`.

Modules - Exemples

import

```
import monModule

monModule.helloWorld()
print(monModule.uneFonction(1))
```

from import

```
from monModule import helloWorld, uneFonction

helloWorld()
print(uneFonction(2))
```

from import *

```
from monModule import *

helloWorld()
print(uneFonction(3))
```

Modules - Exemples

import as

```
import monModule as mod

mod.helloWorld()
print(mod.uneFonction(4))
```

from import as

```
from monModule import helloWorld as hello, uneFonction as identite

hello()
print(identite(5))
```

Exécuter un module

Un **module** est un script Python, il peut donc être exécuté en tant que tel.

Comment savoir si le script est un module ?

La variable `__name__` contient le nom du module.

Si le nom du module est `__main__`, cela signifie que le script est *exécuté*.

Si le nom du module est le nom du fichier (sans l'extension), cela signifie que le script est *importé*.

Tests !

Ce système est très pratique pour tester ses fonctions !

Module exécuté comme script

monModule.py

```
def helloWorld():
    print('Hello World!')
#end helloWorld

def uneFonction(n):
    return n
#end uneFonction

if __name__ == "__main__":
    print("Test uneFonction", uneFonction(1)==1)

    import sys
    if len(sys.argv) == 2:
        print("Test uneFonction", uneFonction(sys.argv[1]) == sys.
            argv[1])
    #end if len(argv)
#end main
```

Paquets

Les **paquets** sont une manière de regrouper et de structurer des modules.

Ils permettent de structurer des modules en créant des sous-modules et des espaces de noms différents.

Ils permettent de regrouper des modules dans un seul et même paquet.

Paquet

```
paquet/
    module.py     # import paquet.module
    sp/
        sous_module.py # import paquet.sp.sous_module
        autre_sous_module.py
```

Gestionnaire de paquets

- `pip` est un gestionnaire de paquets de Python
- PyPI (`https://pypi.org/`) liste les paquets disponibles sur `pip`
- `pip install <paquet>` permet d'installer un paquet
- `pip uninstall <paquet>` permet de désinstaller un paquet
- Les versions sont gérées (on peut installer une version particulière d'un paquet)

Python

Pandas

Pandas

pandas est un paquet Python open source.

C'est un puissant outil dans l'analyse et la manipulation de données.

Il est utilisé dans des algorithmes de *machine learning*.

Il fourni également une bibliothèque graphique qui permet la visualisation des données.

Utilisation de Pandas

Installation :

```
pip install pandas
```

Importation :

```
import pandas
```

Généralement, **pandas** est raccourci en **pd** :

```
import pandas as pd
```

DataFrame

À l'instar de R, **pandas** propose des tableaux de données (dataframes).

Pour illustrer, nous utiliserons le jeu de données des *Iris de Fisher* :

- 3 variétés d'Iris (Setosa, Versicolour, Virginica)
- 50 iris de chaque variétés
- Pour chaque iris on connaît :
  - la longueur du sépal (en cm)
  - la largeur du sépal (en cm)
  - la longueur du pétal (en cm)
  - la largeur du pétal (en cm)
  - sa variété
- Le fichier est au format CSV, avec les en-têtes

Chargement des données

La fonction `read_csv` de **pandas** crée un dataframe à partir du fichier CSV lu.

Chargement et affichage des données Iris

```
>>> df = pd.read_csv("iris.csv")
>>> print(df)
     sepal_length  sepal_width  petal_length  petal_width           class
0             5.1          3.5           1.4          0.2     Iris-setosa
1             4.9          3.0           1.4          0.2     Iris-setosa
2             4.7          3.2           1.3          0.2     Iris-setosa
3             4.6          3.1           1.5          0.2     Iris-setosa
4             5.0          3.6           1.4          0.2     Iris-setosa
..            ...          ...           ...          ...             ...
145           6.7          3.0           5.2          2.3  Iris-virginica
146           6.3          2.5           5.0          1.9  Iris-virginica
147           6.5          3.0           5.2          2.0  Iris-virginica
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]
```

Sauvegarde des données

La méthode `to_csv` de `DataFrame` permet d'écrire dans un fichier.

Sauvegarder dans un fichier

```
>>> df.to_csv("iris2.csv", index=False)
```

Plusieurs paramètres possibles, dont :

- `index` : indique s'il faut inclure les indices de lignes (défaut : True)
- `header` : indique s'il faut inclure l'en-tête (défaut : True)
- `sep` : indique le séparateur à utiliser (défaut : ",")
- `encoding` : précise l'encodage à utiliser
- `decimal` : caractère séparateur de décimales (défaut : ".")
- `quotechar` : caractère de citation (défaut : "")

Manipulation du dataframe (I)

- La taille : `df.shape`

```
(150, 50)
```

- Le début : `df.head()` (La fin : `df.tail()`)

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

- Les colonnes : `df.columns`

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', '
    class'], dtype='object')
```

- Le nom des colonnes : `df.columns.values`

```
['sepal_length' 'sepal_width' 'petal_length' 'petal_width' 'class']
```

Manipulation du dataframe (II)

- Les types des colonnes : `df.dtype`

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
class            object
dtype: object
```

- Des informations : `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   class         150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Manipulation du dataframe (III)

- Description (statistiques) des valeurs numériques :

```
df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

- Description (statistiques) des valeurs 'objet' :

```
df.describe(include=object)
```

	class
count	150
unique	3
top	Iris-setosa
freq	50

Accès aux séries

La *série* est le nom donné par **pandas** aux colonnes d'un dataframe.

L'accès aux séries peut se faire de 3 manières différentes :

- Via un attribut du dataframe
- Via un nom ou une liste de noms de colonnes

Accès via un attribut

```
>>> df.petal_length
0       1.4
1       1.4
2       1.3
3       1.5
4       1.4
       ...
145     5.2
146     5.0
147     5.2
148     5.4
149     5.1
Name: petal_length, Length: 150, dtype: float64
```

Accès aux séries

La *série* est le nom donné par **pandas** aux colonnes d'un dataframe.

L'accès aux séries peut se faire de 3 manières différentes :

- Via un attribut du dataframe
- Via un nom ou une liste de noms de colonnes

Accès via un nom de colonne

```
>>> df["petal_length"]
0      1.4
1      1.4
2      1.3
3      1.5
4      1.4
      ...
145    5.2
146    5.0
147    5.2
148    5.4
149    5.1
Name: petal_length, Length: 150, dtype: float64
```

Accès aux séries

La *série* est le nom donné par **pandas** aux colonnes d'un dataframe.

L'accès aux séries peut se faire de 3 manières différentes :

- Via un attribut du dataframe
- Via un nom ou une liste de noms de colonnes

Accès via une liste de noms de colonnes

```
>>> df[["petal_length", "petal_width"]]
     petal_length  petal_width
0             1.4          0.2
1             1.4          0.2
2             1.3          0.2
3             1.5          0.2
4             1.4          0.2
..            ...          ...
145           5.2          2.3
146           5.0          1.9
147           5.2          2.0
148           5.4          2.3
149           5.1          1.8
[150 rows x 2 columns]
```

Opérations sur les séries

Les séries sont des objets **pandas** sur lesquels on peut appliquer différentes opérations :

Opérations sur les données

```
>>> df[["petal_length", "petal_width"]].mean()
petal_length    3.758667
petal_width     1.198667
dtype: float64
>>> df["class"].value_counts()
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

Manipulation des séries

- Une série est un vecteur, qui peut être accédé via des indices :

Accès aux séries via indice

```
>>> df.petal_length[0]
1.4
```

- On peut également accéder à plusieurs valeurs en indiquant l'indice de départ, et l'indice de fin (exclus) :

Accès aux séries via une plage d'indices

```
>>> df.petal_length[0:4]
0    1.4
1    1.4
2    1.3
3    1.5
Name: petal_length, dtype: float64
```

Tri d’une série

Une série peut être triée :

Tri d’une série

```
>>> df.sepal_width.sort_values().head()
60      2.0
62      2.2
119     2.2
68      2.2
41      2.3
Name: sepal_width, dtype: float64
>>> df.sepal_width.sort_values(ascending=False).head()
15     4.4
33     4.2
32     4.1
14     4.0
16     3.9
Name: sepal_width, dtype: float64
```

Tri d'un dataframe

Un dataframe peut également être trié, il faut indiquer par quelle série le tri doit s'effectuer :

Accès aux séries via une plage d'indices

```
>>> df.sort_values(by="sepal_width").head()
     sepal_length  sepal_width  petal_length  petal_width           class
60            5.0          2.0           3.5          1.0  Iris-versicolor
62            6.0          2.2           4.0          1.0  Iris-versicolor
119           6.0          2.2           5.0          1.5   Iris-virginica
68            6.2          2.2           4.5          1.5  Iris-versicolor
41            4.5          2.3           1.3          0.3      Iris-setosa
>>> df.sort_values(by="sepal_width",ascending=False).head()
    sepal_length  sepal_width  petal_length  petal_width           class
15           5.7          4.4           1.5          0.4  Iris-setosa
33           5.5          4.2           1.4          0.2  Iris-setosa
32           5.2          4.1           1.5          0.1  Iris-setosa
14           5.8          4.0           1.2          0.2  Iris-setosa
16           5.4          3.9           1.3          0.4  Iris-setosa
```

Itérer sur les valeurs

Les séries peuvent être itérées soit par indices, soit par itérateurs :

Itération par indice

```
>>> for i in range(0,len(df.columns)):
...     print(df[df.columns[i]].dtype)
...
float64
float64
float64
float64
object
```

Itération par itérateur

```
>>> for col in df.columns:
...     print(df[col].dtype)
...
float64
float64
float64
float64
object
```

Itération via des call-back

La méthode **apply** permet d'appliquer une fonction sur les colonnes (`axis=0`, par défaut) ou les lignes (`axis=1`).

Itération par call-back

```
>>> def moyenne(x):
...     return x.mean()
...
>>> df[["petal_width", "petal_length"]].apply(moyenne)
petal_width     1.198667
petal_length    3.758667
dtype: float64
>>> df[["petal_width", "petal_length"]].apply(moyenne, axis=1).head()
0    0.80
1    0.80
2    0.75
3    0.85
4    0.80
dtype: float64
```

Itération par lambda fonction

La méthode **apply** peut également prendre en paramètre des lambda fonctions plutôt que des fonctions.

Itération par lamba fonction

```
>>> df[["petal_width", "petal_length"]].apply(lambda x:x.mean())
petal_width     1.198667
petal_length    3.758667
dtype: float64
```

Le paramètre de la lambda fonction sont soit la ligne soit la colonne. Certaines opérations arithmétiques se propagent directement aux éléments :

Itération par lamba fonction

```
>>> df[["petal_width", "petal_length"]].apply(lambda x:-x)[0:2]
   petal_width  petal_length
0         -0.2          -1.4
1         -0.2          -1.4
```

Sélection de colonnes par type

Un dataframe peut être composé de colonnes de types différents. La méthode **`apply`** va s'appliquer à toutes les données, sans distinction.

La méthode `select_dtypes()` permet d'inclure (ou exclure) des colonnes selon leur type.

Sélection des colonnes numériques

```
>>> df.select_dtypes("number").apply(moyenne)
sepal_length    5.843333
sepal_width     3.054000
petal_length    3.758667
petal_width     1.198667
dtype: float64
```

Accès aux valeurs par indices matriciels

L'attribut `iloc` d'un dataframe permet d'accéder aux éléments comme une matrice, avec le ou les indices des lignes et colonnes.

Sélection de la première valeur

```
>>> df.iloc[0,0]
5.1
```

On peut sélectionner des plages d'indices avec `:`.

Sélection de la première et deuxième ligne de toutes les colonnes

```
>>> df.iloc[0:2, :]
   sepal_length  sepal_width  petal_length  petal_width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
```

Accès aux dernières valeurs par indices matriciels

Il est possible d'utiliser des indices négatifs.

L'indice sélectionné correspond alors au résultat de l'addition entre la taille de la dimension et l'indice donné.

Sélection de la dernière valeur

```
>>> df.iloc[-1,-1]
'Iris-virginica'
>>> df.iloc[df.shape[0]-1, df.shape[1]-1]
'Iris-virginica'
```

On peut également sélectionner des plages d'indices négatifs.

Sélection de l'avant-dernière et dernière ligne de toutes les colonnes

```
>>> df.iloc[-2:, :]
     sepal_length  sepal_width  petal_length  petal_width          class
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica
```

Sélection par booléens

L'attribut `loc` de dataframe permet de sélectionner des lignes et colonnes en fonction d'un vecteur de booléens.

Sélection des iris Virginica

```
>>> df.loc[df['class']=='Iris-virginica', :].head()
     sepal_length  sepal_width  petal_length  petal_width           class
100           6.3          3.3           6.0          2.5  Iris-virginica
101           5.8          2.7           5.1          1.9  Iris-virginica
102           7.1          3.0           5.9          2.1  Iris-virginica
103           6.3          2.9           5.6          1.8  Iris-virginica
104           6.5          3.0           5.8          2.2  Iris-virginica
```

La sélection des colonnes peut se faire classiquement.

Sélection des iris Virginica

```
>>> df.loc[df['class']=='Iris-virginica', ["sepal_length","petal_length"
    ]][0:2]
     sepal_length  petal_length
100           6.3           6.0
101           5.8           5.1
```

Opérations booléennes pour la sélection

Pour pouvoir faire des sélections plus complexes, vous pouvez utiliser :

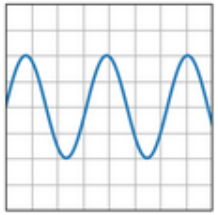
- La méthode `isin()` qui prend en paramètre un tableau de valeurs et retourne vrai pour chaque valeur de la série présente dans le tableau donné en paramètre.
- Les opérateurs sur les booléens `&` (and), `|` (or), `~` (not).
- Les opérateurs sur les valeurs numériques (<, <=, >, ...)

matplotlib

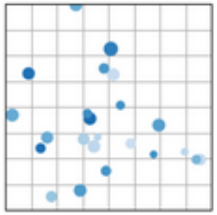
- Ce package Python est le plus utilisé pour réaliser des graphiques
- Il permet d'utiliser les fonctions de MATLAB dans Python
- Il est incorporé dans **pandas**.

```
import matplotlib.pyplot as plt
```

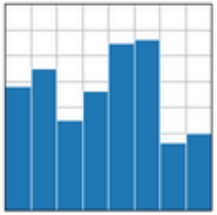
Types de graphiques



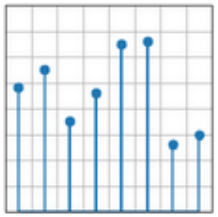
plot(x, y)



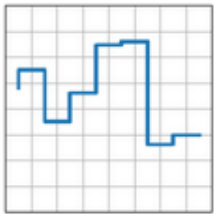
scatter(x, y)



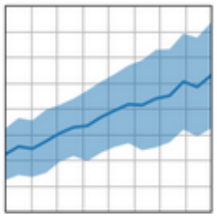
bar(x, height)



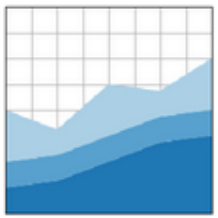
stem(x, y)



step(x, y)



fill_between(x, y1, y2)

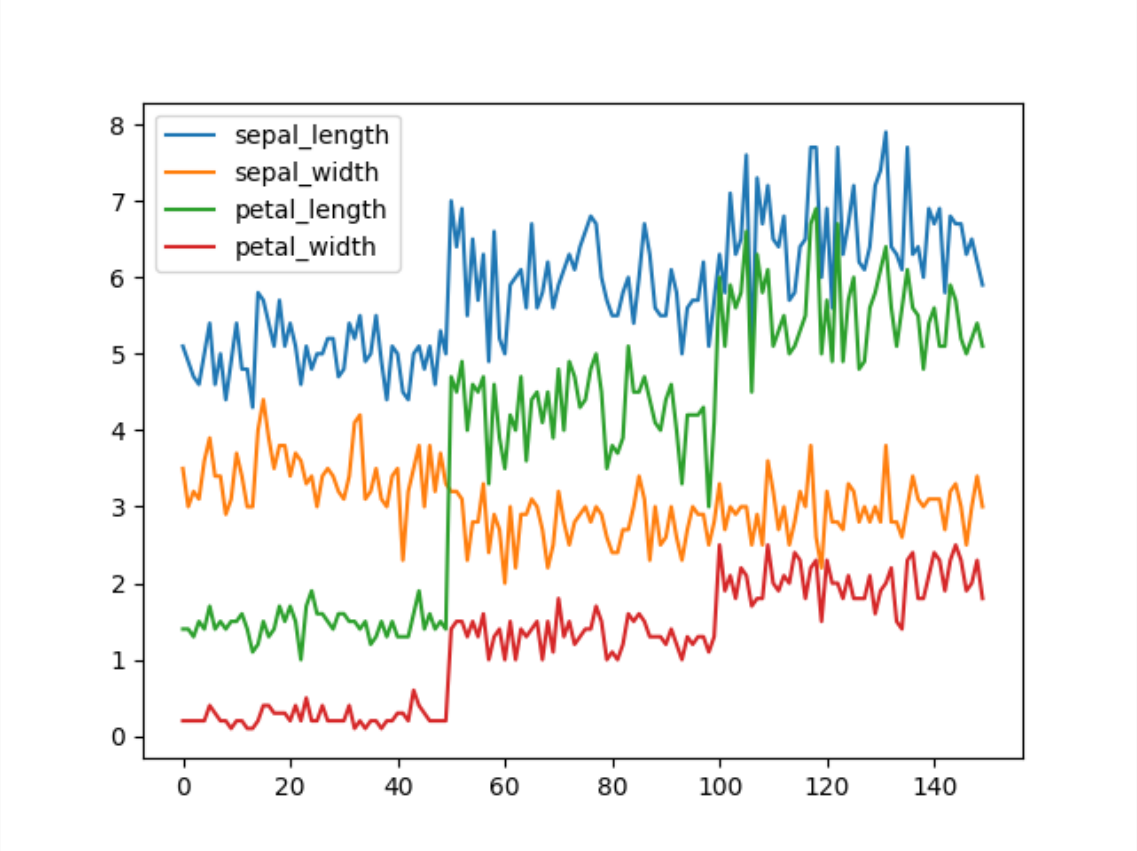


stackplot(x, y)

Inclusion dans pandas (I)

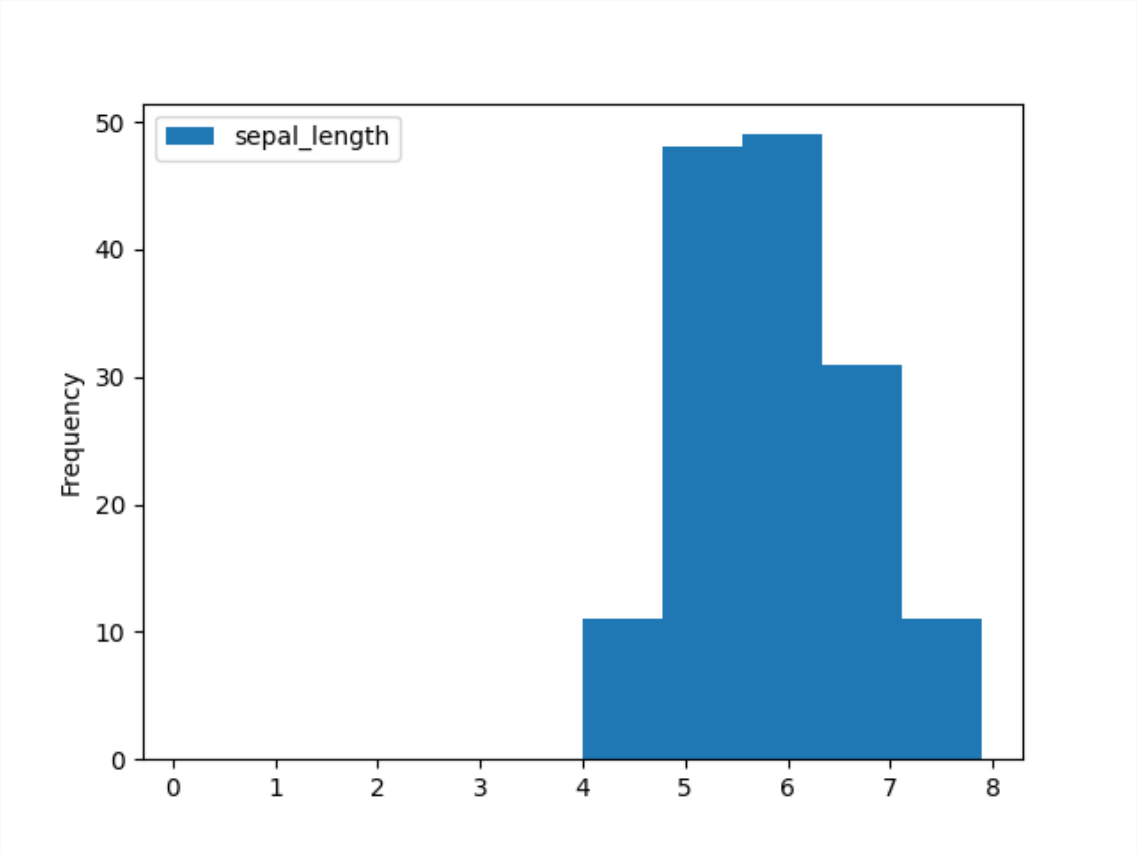
- matplotlib peut s'utiliser seul `plot(x, y)`
- matplotlib est aussi inclus dans **pandas** avec l'attribut `plot`

```
df.plot.line()
plt.show()
```



Inclusion dans pandas (II)

```
df.plot.hist(column="sepal_length")
plt.show()
```



Inclusion dans pandas (III)

```
df.plot.hist(column="sepal_length",
    by="class")
plt.show()
```