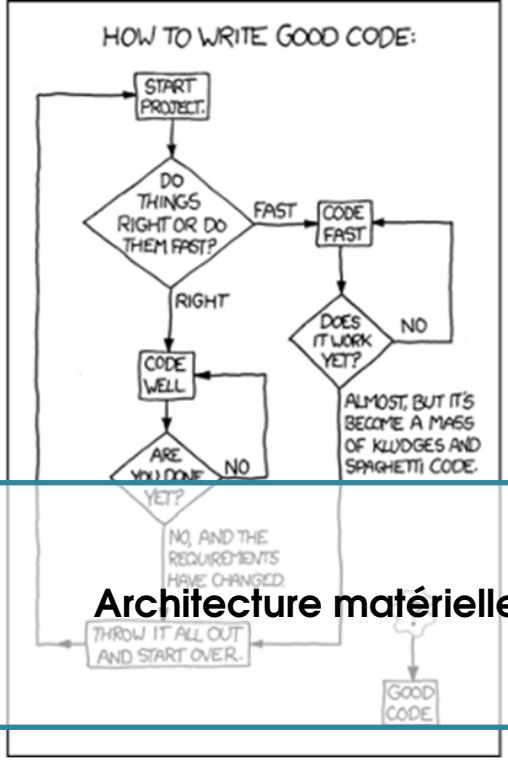

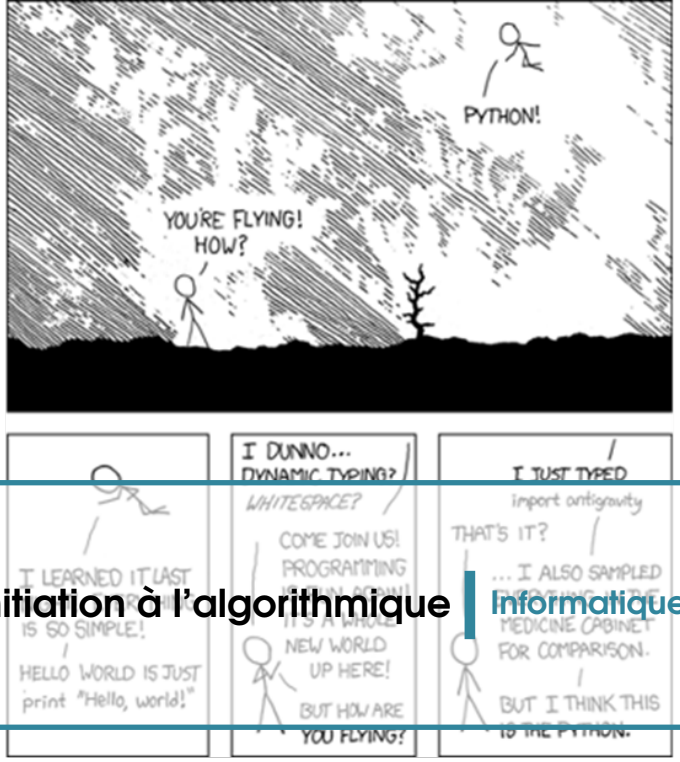

Chapitre 1 – 7
Fichiers

27 Novembre 2019

Cours

Savoirs et compétences :

- *AA.S1 : Se familiariser aux principaux composants d'une machine numérique*
- *AA.C9 : Choisir un type de données en fonction d'un problème à résoudre*
- *AA.S12 : Fichiers*

Émilien Durif
Xavier Pessoles
MPSI

1 Introduction

En informatique, l'information se code comme une suite de bits (en général regroupés en octets). Sur le disque dur, cette information est répartie dans des *fichiers*.

Définition Un fichier est une suite (généralement finie) d'octets [a].

a. Sous Unix, il y a des fichiers spéciaux qui sont une suite infinie d'octets, notamment `/dev/zero` et `/dev/random`

Un fichier peut contenir tout type d'information. Comment alors interpéter les données contenu dans ce fichier ? Il n'existe pas de règle *a priori*.

Ainsi, c'est celui qui écrit le fichier qui choisit la méthode à utiliser pour comprendre ce fichier. [1] Celui qui lit le fichier doit utiliser la même méthode pour obtenir l'information.

Heureusement, il y a des conventions qui permettent à tous de s'y retrouver.

Nous allons maintenant nous intéresser à la question suivante : comment lire/écrire des données dans des fichiers depuis python ?

2 The Good

C'est simple sur le plan conceptuel.

1. On accède au fichier (on dit qu'on l'*ouvre*), en créant un objet qui le représente (on dit que c'est un *objet fichier* ou un objet *de type fichier*).
2. On a des fonctions pour lire le contenu du fichier (un octet ou une ligne de texte ou tout ce qui reste à lire) ou pour écrire (un octet ou un texte) dans un fichier.
3. L'accès est par défaut séquentiel : il faut imaginer une tête de lecture positionnée au début du fichier qui se déplace au fur et à mesure de la lecture ou de l'écriture.
4. On *ferme* le fichier, c'est-à-dire que le système d'exploitation enregistre les éventuelles modifications sur le fichier, avant de détruire l'objet fichier créé dans Python.

Pour ouvrir un fichier, on utilise la fonction python `open(nom_de_fichier, m)` où

- `nom_de_fichier` est une chaîne de caractères (type `str`) contenant le nom du fichier (plus précisément : un nom de chemin—absolu ou relatif—vers ce fichier) ;
- `m` est une chaîne de caractères (type `str`) contenant le *mode d'ouverture* du fichier :
  - `'r'` pour une ouverture en lecture seule ;
  - `'w'` pour une ouverture en écriture seule (fichier tronqué à 0 octets s'il existe, créé s'il n'existe pas) ;
  - `'a'` pour une ouverture en ajout (tête d'écriture positionnée en fin de fichier, fichier créé s'il n'existe pas).

Cette fonction retourne un *objet fichier* (appelé aussi *descripteur de fichier*) à partir duquel on va pouvoir lire ou écrire dans le fichier.

Voici un exemple de lecture d'un fichier :

```
>>> f = open('Cy_01_Ch_07_Cours_PDF.tex', 'r')
>>> x = f.readline()
>>> y = f.readline()
>>> print(x)
\documentclass[10pt,fleqn]{article} % Default font size and left-justified equations

>>> print(y)
\usepackage[%

>>> r = f.read()
>>> f.close()
```

R

1. `readline`, `read` et `close` ne sont pas de « vraies fonctions », mais sont rattachées à l'objet `f`. Ce sont des *méthodes* applicables à cet objet.
2. La méthode `read` lit tout le fichier, depuis la position courante jusqu'à la fin. Elle renvoie une chaîne de caractères.
3. Il n'est pas nécessaire de lire l'intégralité d'un fichier avant de le fermer.
4. Lecture et écriture sur un fichier fermé renvoient une erreur.
5. La méthode `readline` retourne toute une ligne (sous forme de chaîne de caractères), retour chariot inclus. Par exemple, ici, `x` vaut `'root:x:0:0:root:/root:/bin/bash\n'`

Pour écrire dans un fichier, c'est le même principe.

1. En général c'est un programme, qui suit les choix de son programmeur.

```
>>> f = open('monfichier.txt', 'w')
>>> f.write("Bonjour,\n")
9
>>> f.write("Comment")
7
>>> f.write("vas-tu ?")
8
>>> f.close()
```

Contenu du fichier `monfichier.txt` après l'exécution :

```
Bonjour,
Commentvas-tu ?
```

Attention, il convient de bien se souvenir que :
- `open(nom_de_fichier, 'w')` efface le fichier `nom_de_fichier` s'il existait déjà ;
- `open(nom_de_fichier, 'a')` ajoute à la fin du fichier `nom_de_fichier` s'il existait déjà.

3 The Bad

Il ne faut surtout pas oublier `f.close()` ! En effet,
- Le nombre de fichiers ouverts simultanément est limité.
- Les écritures n'ont pas lieu immédiatement :
  - Python attend d'en avoir suffisamment avant de transmettre au système d'exploitation ;
  - Le système d'exploitation écrit quand cela lui semble approprié.

Un `f.close()` ferme le fichier et force le transfert au système d'exploitation.
Il existe une autre façon de forcer l'écriture : `f.flush()` (ne dispense pas de fermer).

Surtout, une instruction `with` a été introduite en Python pour s'assurer de la fermeture des fichiers (en lecture comme en écriture). Par exemple, on peut réécrire la liste d'instructions précédentes comme suit.

```
with open('encoremonfichier.txt', 'w') as f:
    f.write("Bonjour,\n")
    f.write("Comment")
    f.write("vas-tu ?")
```

À la sortie du bloc `with`, python fait automatiquement un `f.close()`. En cas d'erreur dans le bloc, `f` est fermé correctement.

4 The Ugly

Une question important est de choisir la convention à utiliser pour traduire un texte en suite d'octets. Malheureusement, il n'y a pas une norme, mais plusieurs ...

4.1 ASCII

Dans les années (19)60, la norme ASCII [2] est introduite, pour représenter les caractères. Chaque caractère codé par un entier dans $[\![0, 128[\![$ et peut donc être codé sur 7 bits, donc sans problème sur un octet. Cette norme fut très vite adoptée.

Les caractères 0 à 31 ainsi que 127 ne sont pas des caractères à proprement parler, mais représentent des instructions (à destinations des imprimantes, par exemple). La liste des caractères est donnée dans la figure 1

4.1.1 A comme ...

ASCII : American Standard Code for Information Interchange.
Ici, «American» veut dire «États-Unien», donc :
- sans l'Amérique latine (español, português)
- ni le Québec (de langue française)

Bref, aucune lettre accentuée n'est disponible...

2. Prononcez : «à ski».

N°	Symbole	N°	Symbole	N°	Symbole	N°	Symbole	N°	Symbole
32		52	4	72	H	92	\	112	p
33	!	53	5	73	I	93	]	113	q
34	"	54	6	74	J	94	^	114	r
35	#	55	7	75	K	95	_	115	s
36	$	56	8	76	L	96	`	116	t
37	%	57	9	77	M	97	a	117	u
38	&	58	:	78	N	98	b	118	v
39	'	59	;	79	O	99	c	119	w
40	(	60	<	80	P	100	d	120	x
41	)	61	=	81	Q	101	e	121	y
42	*	62	>	82	R	102	f	122	z
43	+	63	?	83	S	103	g	123	{
44	,	64	@	84	T	104	h	124	|
45	-	65	A	85	U	105	i	125	}
46	.	66	B	86	V	106	j	126	~
47	/	67	C	87	W	107	k		
48	0	68	D	88	X	108	l		
49	1	69	E	89	Y	109	m		
50	2	70	F	90	Z	110	n		
51	3	71	G	91	[	111	o		

FIGURE 1 – Table des caractères ASCII

4.2 Les tentatives d'amélioration

Dans un octet, on peut avoir 256 valeurs distinctes. On peut donc coder 256 caractères sur un octet. Le code ASCII ne prend que 128 valeurs, il y a donc 128 places restantes. La norme ISO 8859-1 (Latin 1) utilise la place disponible pour ajouter les accents. Mais :

- cela ne suffit pas pour l'Europe Centrale → ISO 8859-2 (différent) ;
- Et pour le turc et le maltais et l'esperanto ? ISO 8859-3
- Et les pays baltes ? ISO 8859-4
- En russe, on n'utilise pas un alphabet bizarre ? ISO 8859-5
- Et en arabe ? ISO 8859-6
- Et en grec ? ISO 8859-7
- Et l'hébreu ? ISO 8859-8
- …AAAAAAARGH !

4.2.1 Et bien sûr…

Fidèle à sa stratégie commerciale «Embrace, extend & extinguish», Microsoft a fait ses propres versions :

CP437, CP737, CP850, CP852, CP855, CP857, CP858, CP860, CP861, CP862, CP863, CP865, CP866, CP869, CP872, Windows-1250, Windows-1251, Windows-1252, Windows-1253, Windows-1254, Windows-1255, Windows-1256, Windows-1257, Windows-1258.

… et a décidé de représenter les retours à la ligne différemment de tous les autres (c'est plus simple pour les imprimantes matricielles…).

4.3 La solution : Unicode et …

À la fin des années (19)80 et au début des années (19)90, l'idée de standardiser un codage universel s'est répandue.

En 2012, on trouve $1,1 \times 10^5$ «caractères» définis, couvrant 100 écritures. Chacun est repéré par un «point de code» (*code point*), entier de $[\![0, 1\,114\,112[\![$.

Mais, cela ne tient pas sur un octet, ni même sur deux. En Python, une chaîne de caractère peut être vue comme un tableau de points de code unicode.

4.4 …UTF-8

C'est un code de taille variable, de 1 à 4 octets. Il est compatible avec ASCII : un caractère ASCII est codé en UTF-8 sur un octet, de la même manière qu'en ASCII.

Actuellement, c'est le format le plus utilisé pour les pages web. Notamment, c'est le format canonique des fichiers textes sur toutes les plateformes modernes (sauf Microsoft). Il est disponible sur toutes les plateformes, y compris Microsoft.

4.5 La lecture de fichiers textes en Python

Ce qui suit est valable pour Python à partir de la version 3

Python tente d'être aussi raisonnable que possible :

1. Quand il lit/écrit un fichier, il décode/code les textes en utilisant le codage standard du système sur lequel il tourne (donc UTF-8 sauf pour sous MS-Windows ; Windows-1252 sur un PC français sous MS-Windows).
2. `open` dispose d'une option permettant de spécifier l'encodage à utiliser. Par exemple

```
f=open('monfichier.txt', 'r', encoding='utf-8')
```

3. Il est maintenant possible d'utiliser des caractères spécifiques au code UTF-8.

```
f.write("Bonjour,\n")
f.write("Comment")
f.write("ça va ?")
```

NB : Il est possible d'ouvrir un fichier en mode binaire (`open(..., 'rb')`). La lecture retourne alors des objets de type `bytes` (tableaux d'octets), les fonctions pour écrire prennent en argument des `bytes`.

4.6 Nom de fichier et chemin

La fonction `open` prend en argument un nom de fichier considéré avec ou sans chemin. Il y a alors plusieurs possiblités d'appeler un fichier et il faut prendre ses précautions sous peine de voir le fameux message d'erreur :

```
>>> open('montexte.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'montexte.txt'

FileNotFoundError: [Errno 2] No such file or directory:
```

Pour corriger cette erreur il convient de prendre des précautions. Au choix :

- si l'instruction `open('montexte.txt')` est dans un script, il faut que le fichier 'montexte.txt' soit dans le même répertoire que le script en question.
- Même avec cette précaution selon l'environnement, il faut spécifier que le répertoire courant corresponde au répertoire qui contient le script et le fichier à lire. Par exemple dans Pyzo, il convient exécuter le script (au moins la première fois) en sélectionnant *run* puis *run file as script* (ou Ctrl+Maj+E).
- en remplaçant 'montexte.txt' par la chaine de caractère contenant le chemin absolu + le nom du fichier, il n'y a en principe aucun problème.

5 En résumé

Pour ouvrir un fichier en mode texte :

```
with open(nom_de_fichier, mode) as f:
    # faire ce qu'on veut avec f
# ici f a été fermé.
```

- `nom_de_fichier` : type `str`, chemin désignant le fichier.
- `mode` : type `str`, valeurs possibles : `'r'`, `'w'`, et `'a'`.

Méthodes utiles pour la lecture en mode texte :

- `f.readline()` : lit une ligne, la retourne sous forme de chaîne.
- `f.read()` : lit tout ce qui reste, le retourne sous forme de chaîne.
- `f.readlines()` : lit toutes les lignes restantes, les retourne sous forme de liste de chaînes.

Méthode utile pour l'écriture en mode texte : `f.write(chaine)`

Ne pas oublier de fermer les fichiers si on ne les pas ouverts avec `with` ! (méthode `f.close()`)

6 Un exemple.

On part du tableur suivant (voir figure 2), qui contient matière par matière les moyennes semestrielles de quelques étudiants, ainsi que les coefficients de chaque matière. On aimerait calculer la moyenne pondérée de chaque étudiant.

En enregistrant ce ficher au format CSV (pour *Comma-Separated Values*), on obtient le fichier `moyennes_matieres.csv` suivant.

```
Matière;Maths;Physique;Lettres;Anglais;Informatique;SII
Coefficient;8;7;4;3;2;2
Laura;8,15;9,84;11;5,5;10,64;7,16
Clémentine;8,93;11,6;12,33;11,4;11,16;10,04
Solène;7,08;8,53;10;7,4;9,66;10,06
```

	A	B	C	D	E	F	G
1	Matière	Maths	Physique	Lettres	Anglais	Informatique	SII
2	Coefficient	8	7	4	3	2	2
3	Laura	8,15	9,84	11	5,5	10,64	7,16
4	Clémentine	8,93	11,6	12,33	11,4	11,16	10,04
5	Solène	7,08	8,53	10	7,4	9,66	10,06
6	Donovan	12,33	11,97	12,17	10,3	16,25	15,98
7	Élodie	7,93	9,49	10,5	6,8	10,26	9,31
8	Ugo	12,43	13,28	10,33	11,2	17,38	12,11

FIGURE 2 – Tableau des moyennes de chaque étudiant, matière par matière.

```
Donovan;12,33;11,97;12,17;10,3;16,25;15,98
Élodie;7,93;9,49;10,5;6,8;10,26;9,31
Ugo;12,43;13,28;10,33;11,2;17,38;12,11
```

On peut commencer par extraire les données du tableau sous Python comme suit (la première ligne ne nous intéresse pas).

```
with open("moyennes_matieres.csv","r") as f :
    _ = f.readline()
    coeffs = f.readline().strip().split(';')
    T = f.readlines()
```

R Avec la méthode `strip`, on enlève directement le caractère `\n` et les espaces éventuels à la fin de la ligne lue.

On peut ensuite traiter la ligne des coefficients.

```
n = len(coeffs)
S = 0 # Somme des coefficients
for i in range(1,n):
    coeffs[i] = int(coeffs[i])
    S = S + coeffs[i]
coeffs.append(S)
```

R On a pris soin ici de convertir directement les coefficients dans le type adéquat et l'on a calculé la somme des coefficients, qui servira pour calculer la moyenne pondérée de chaque étudiant.

À la fin de ce traitement, voici l'état de la variable `coeffs`.

```
['Coefficient', 8, 7, 4, 3, 2, 2, 26]
```

On peut ensuite traiter le reste du tableau, comme suit.

```
for i in range(len(T)) :
    T[i] = T[i].strip().split(';')
    for j in range(1,n):
        T[i][j] = float(T[i][j].replace(',','.'))
```

R On a là encore pris soin d'enlever le caractère `\n` à la fin de chaque ligne et de remplacer les `,` (notation décimale française) par des `.` (notation décimale anglaise).

À la fin de ce traitement, voici l'état de la variable `T`.

```
[['Laura', 8.15, 9.84, 11.0, 5.5, 10.64, 7.16],
 ['Clémentine', 8.93, 11.6, 12.33, 11.4, 11.16, 10.04],
 ['Solène', 7.08, 8.53, 10.0, 7.4, 9.66, 10.06],
 ['Donovan', 12.33, 11.97, 12.17, 10.3, 16.25, 15.98],
 ['Élodie', 7.93, 9.49, 10.5, 6.8, 10.26, 9.31],
 ['Ugo', 12.43, 13.28, 10.33, 11.2, 17.38, 12.11]]
```

On peut ensuite calculer pour chaque étudiant sa moyenne pondérée comme suit.

```
for e in T :
    moy = 0
    for i in range(1,n):
        moy = moy + coeffs[i]*e[i]
    moy = moy / coeffs[-1]
    e.append(moy)
```

À la fin de ce traitement, voici l'état de la variable `T`.

```
[['Laura', 8.15, 9.84, 11.0, 5.5, 10.64, 7.16, 8.853076923076923],
 ['Clémentine', 8.93, 11.6, 12.33, 11.4, 11.16, 10.04, 10.713846153846152],
 ['Solène', 7.08, 8.53, 10.0, 7.4, 9.66, 10.06, 8.38423076923077],
 ['Donovan', 12.33, 11.97, 12.17, 10.3, 16.25, 15.98, 12.55653846153846],
 ['Élodie', 7.93, 9.49, 10.5, 6.8, 10.26, 9.31, 8.900384615384617],
 ['Ugo', 12.43, 13.28, 10.33, 11.2, 17.38, 12.11, 12.549999999999999]]
```

Il ne reste plus qu'à écrire les valeurs calculées dans un tableau, comme suit.

```
with open("moyennes_ponderees.csv","w") as f :
    f.write("Prénoms;Moyennes pondérées")
    for e in T :
        f.write('\n'+';'.join([str(e[0]),str(e[-1])]))
```

Le contenu du fichier `moyennes_ponderees.csv` est le suivant.

```
Prénoms;Moyennes pondérées
Laura;8.853076923076923
Clémentine;10.713846153846152
Solène;8.38423076923077
Donovan;12.55653846153846
Élodie;8.900384615384617
Ugo;12.549999999999999
```

Dans un tableur, on lit cela (voir figure 3).

	A	B
1	Prénoms	Moyennes pondérées
2	Laura	8.853076923076923
3	Clémentine	10.713846153846152
4	Solène	8.38423076923077
5	Donovan	12.55653846153846
6	Élodie	8.900384615384617
7	Ugo	12.549999999999999

FIGURE 3 – Tableau des moyennes pondérées.

7 Exercices.