

Bases d'algorithmique et de programmation

Sébastien Gagné, Université d'Orléans

L1 Algo - Prog - S1

Cours 06 - Lecture et écriture dans un fichier

Plan du cours

- Pérennisation des données
- Lecture d'un fichier
- Ecriture dans un fichier

Pérennisation des données

Quand on exécute un script "classique", écrit dans un langage quelconque (dont python), se posent des problèmes essentiels :

- Une fois le programme terminé, que deviennent les variables du programme ?
- Que deviennent les valeurs de ces variables ?
- Quel est l'intérêt d'un programme qui ne garde pas trace de ses résultats ?
- Comment garder "en mémoire" une trace de ces valeurs, pour une réutilisation future ?

Pérennisation des données

Quand on exécute un script "classique", écrit dans un langage quelconque (dont python), se posent des problèmes essentiels :

- Une fois le programme terminé, que deviennent les variables du programme ? **détruites**
- Que deviennent les valeurs de ces variables ? **perdues**
- Quel est l'intérêt d'un programme qui ne garde pas trace de ses résultats ? **limité à une utilisation immédiate**
- Comment garder "en mémoire" une trace de ces valeurs, pour une réutilisation future ? **c'est LA question ...**

Pérennisation des données

Procédure que tout le monde connaît :

1. J'ouvre un logiciel éditeur de texte ;
2. J'écris un contenu sur la page courante ;
3. J'enregistre mon travail dans un fichier, (ex : docu.txt) ;
4. Plus tard, je reprends mon travail (qui a été sauvegardé dans docu.txt) à l'endroit où je l'ai laissé précédemment ;
5. et ainsi de suite, autant de fois que nécessaire.

Cette procédure n'est rendue possible que par le mécanisme de sauvegarde des données dans un fichier.

La fonction *open* de python

La fonction *open* de python permet de travailler sur un fichier. Elle prend en paramètres (dans son utilisation courante) deux chaînes de caractères.

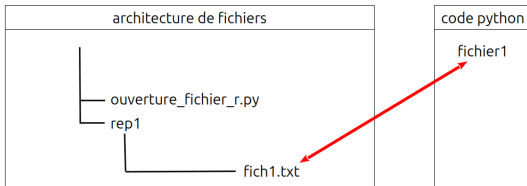
- La première chaîne de caractères est le *nom du fichier* à ouvrir (en fait, le chemin vers ce fichier) ;
- La deuxième désigne le *mode d'ouverture* du fichier.
 - ▶ Le mode "**r**" (**read**) permet la lecture seule du fichier ;
 - ▶ Le mode "**w**" (**write**) permet l'écriture sur le fichier en "l'écrasant" ;
 - ▶ Le mode "**a**" (**append**) permet l'écriture sur le fichier en y ajoutant des données (sans l'écraser) ;

La fonction *open* de python

La fonction *open* ouvre donc un "canal" vers le fichier, en créant ce qu'on peut appeler un **objet-fichier**.

Il y a donc un **fichier physique** et un **fichier logique**.

```
ouverture_fichier_r.py
1  # ouverture, en mode lecture, du fichier situé,
2  # par rapport au fichier script, dans le sous-
3  # répertoire rep1 et nommé fich1.txt
4
5  fichier1 = open('rep1/fich1.txt', 'r')
```



⇒ ***rep1/fich1.txt*** est le chemin vers le fichier physique

⇒ ***fichier1*** est l'objet-fichier (variable python).

Attributs de l'objet-fichier

- L'objet-fichier possède des attributs et des méthodes accessibles par la notation pointée déjà présentée ;
- Les attributs ressemblent à des variables, les méthodes à des fonctions. Ils appartiennent à l'objet-fichier ;
- Principaux attributs : *name*, *mode*, *closed* et *encoding* ;

```
>>> fichier1 = open('rep1/fich1.txt', 'r')
>>> fichier1.name
'rep1/fich1.txt'
>>> fichier1.mode
'r'
>>> fichier1.closed
False
>>> fichier1.encoding
'UTF-8'
```

Méthodes de l'objet-fichier

- La méthode *close* : elle ferme le canal vers le fichier (mais l'objet-fichier python existe toujours);

Cette méthode est importante : elle permet de déverrouiller le fichier pour des applications qui demanderaient son ouverture.

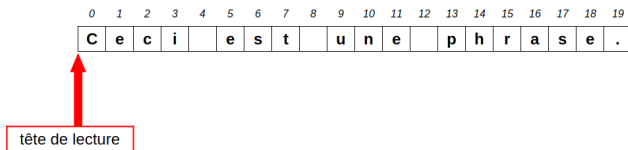
Il est donc nécessaire de fermer le canal vers le fichier grâce à cette méthode *close* quand on a fini d'utiliser son contenu.

```
>>> fichier1 = open('rep1/fich1.txt', 'r')
>>> fichier1.closed
False
>>> fichier1.close()
>>> fichier1.closed
True
```

Tête de lecture du fichier

La lecture du fichier (ouvert en mode 'r') se fait par une "tête de lecture" qui indique jusqu'où le fichier a déjà été lu.

A l'ouverture du fichier, la tête de lecture est à la position 0 (donc, prête à lire le caractère d'indice 0);



Chaque fois qu'une lecture (partielle ou totale) du fichier sera exécutée, la tête de lecture se déplacera ;

Quand tout a été lu, la tête de lecture est en fin de fichier ;

Méthodes de l'objet-fichier

- La méthode ***tell*** indique à quel endroit se trouve la tête de lecture du fichier ;
- La méthode ***read*** (uniquement en mode 'r') prend en charge un entier positif n (optionnel) et lit (et retourne), à partir de la position actuelle de la tête de lecture, les n caractères suivants du fichier ;

Si l'entier n n'est pas précisé, *read()* affiche (et retourne) le reste du fichier (jusqu'à sa fin) ;

Remarque : pour un fichier texte, la fin d'une ligne est un caractère caché "`\n`" de retour à la ligne.

Exemples *read* - *tell* - mode 'r'



```
>>> fichier1 = open('rep1/fich1.txt', 'r')
>>> fichier1.tell()
0
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
C	e	c	i		e	s	t		u	n	e		p	h	r	a	s	e	.



tête de lecture

Exemples *read* - *tell* - mode 'r'



```
>>> fichier1.read(7)
'Ceci es'
>>> fichier1.tell()
7
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
C	e	c	i		e	s	t			u	n	e		p	h	r	a	s	e	.



Exemples *read* - *tell* - mode 'r'



```
>>> fichier1.read(4)
't un'
>>> fichier1.tell()
11
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
C	e	c	i		e	s	t		u	n	e		p	h	r	a	s	e	.

tête de lecture

Exemples *read* - *tell* - mode 'r'

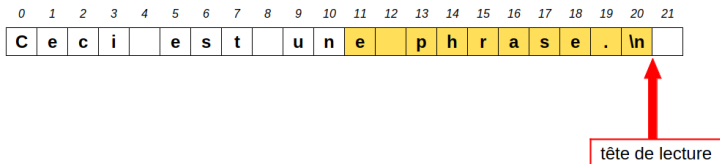


```
>>> fichier1.read()
'e phrase.\n'
>>> fichier1.tell()
21
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
C	e	c	i		e	s	t		u	n	e		p	h	r	a	s	e	.	\n	



Exemples *read* - *tell* - mode 'r'



- Le caractère `\n` à la fin de la ligne existe bien, même s'il n'est pas visible. La lecture du fichier le prend en compte ;
- De façon générale, si la méthode *tell* retourne un entier *i*, alors le prochain caractère lu sera celui d'indice *i* ;
- Si on est en fin de fichier, *read* ne renvoie qu'une chaîne vide et *tell* retourne le même résultat que précédemment.

Méthodes de l'objet-fichier

- La méthode *seek* replace la tête de lecture à l'indice précisé en paramètre.

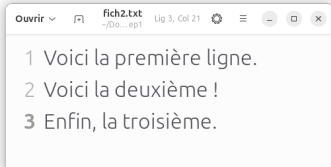
En particulier, *fichier1.seek(0)* la replace au début.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
C	e	c	i		e	s	t		u	n	e		p	h	r	a	s	e	.	\n

```
>>> fichier1 = open('rep1/fich1.txt', 'r')
>>> fichier1.tell()
0
>>> fichier1.read(8)
'Ceci est'
>>> fichier1.tell()
8
>>> fichier1.seek(13)
13
>>> fichier1.tell()
13
>>> fichier1.read()
'phrase.\n'
```

Méthodes de l'objet-fichier

- La méthode ***readline*** (mode 'r') retourne la partie du fichier entre la position actuelle de la tête de lecture et la prochaine fin de ligne du fichier (prochain caractère `\n`);



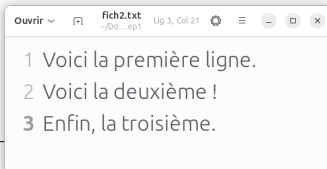
```
>>> f2 = open('rep1/fich2.txt', 'r')
>>> f2.tell()
0
>>> f2.read(4)
'Voic'
>>> f2.readline()
'i la première ligne.\n'
>>> ligne = f2.readline()
>>> ligne
'Voici la deuxième !\n'
>>> ligne2 = f2.readline()
>>> ligne2
'Enfin, la troisième.\n'
>>> f2.readline()
''
>>> f2.close()
```

Méthodes de l'objet-fichier

- La méthode *readlines* (mode 'r') retourne une liste composée des lignes lues entre la position actuelle de la tête de lecture et la fin du fichier ;

```
>>> f2 = open('rep1/fich2.txt', 'r')
>>> liste_lignes = f2.readlines()
>>> liste_lignes
['Voici la première ligne.\n', 'Voici la deuxième !\n', 'Enfin, la troisième.\n']
>>> for ligne in liste_lignes :
    print(ligne, end = '')

Voici la première ligne.
Voici la deuxième !
Enfin, la troisième.
>>> f2.close()
```



L'instruction *with*

- Comme expliqué précédemment, il est nécessaire de fermer l'objet-fichier par la méthode `close` dès que l'accès au fichier physique n'est plus nécessaire ;
- Il y a une instruction en python qui permet de se passer de la méthode `close` : l'instruction *with* ;
- L'ouverture du fichier est alors limitée au bloc *with*.

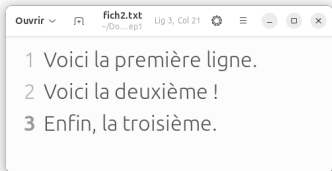


A screenshot of a text editor window. The title bar shows 'ouvrir', 'fich3.txt', and the path '~/.Do...ep1'. The editor content shows three lines of text: '1 Hello.', '2 Salut.', and '3 Coucou.'.

```
>>> with open('rep1/fich3.txt', 'r') as f3 :  
    lignes = f3.readlines()  
    for ligne in lignes :  
        print(ligne, end = '')  
  
Hello.  
Salut.  
Coucou.  
  
>>> f3.closed  
True
```

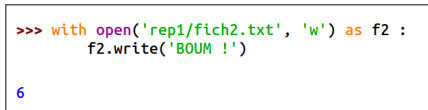
Méthodes de l'objet-fichier

- En mode 'w', la méthode **write** prend en paramètre une chaîne de caractères, et écrit cette chaîne dans le fichier :
 - ▶ Soit le fichier physique existe déjà, et dans ce cas la chaîne de caractères écrase le contenu précédent ;
 - ▶ Soit le fichier physique n'existe pas, et dans ce cas il est au préalable créé, puis la chaîne y est écrite ;



```
Ouvrir  fich2.txt  Lig 3, Col 21
~/Do... ep1

1 Voici la première ligne.
2 Voici la deuxième !
3 Enfin, la troisième.
```



```
>>> with open('rep1/fich2.txt', 'w') as f2 :
        f2.write('BOUM !')
```

6

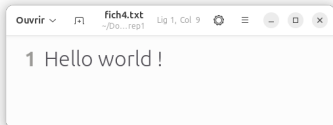


```
Ouvrir  fich2.txt  Lig 1, Col 7
~/Do... /rep1

1 BOUM !
```

Méthodes de l'objet-fichier

- En mode 'a', la méthode **write** prend en paramètre une chaîne de caractères, et écrit cette chaîne dans le fichier :
 - ▶ Soit le fichier physique existe déjà, et dans ce cas la chaîne de caractères est écrite en fin du fichier ;
 - ▶ Soit le fichier physique n'existe pas, et dans ce cas il est au préalable créé, puis la chaîne y est écrite ;



```
Ouvrir ▾ fich4.txt  
~/.Do...rep1 Lig 1, Col 9  
1 Hello world !
```



```
>>> with open('rep1/fich4.txt', 'a') as f4 :  
        f4.write('Bonjour tout le monde !')
```

23



```
Ouvrir ▾ fich4.txt  
~/.Do...rep1 Lig 2, Col 24  
1 Hello world !  
2 Bonjour tout le monde !
```

Exemple - Gestion de stock - v1

- On saisit des noms de fruit, ainsi que les quantités ;
- A chaque saisie, on met à jour un fichier de stock ;
- On recommence jusqu'à la saisie d'un message d'arrêt ;
- A la fin un fichier **stock.txt** existe avec le détail du stock.

```
1 while True :
2     fruit = input("entrez le fruit : ")
3     quantite = int(input("entrez la quantité : "))
4     with open('stock.txt', 'a') as stock :
5         stock.write(fruit + ":" + str(quantite) + '\n')
6     reponse = input("entrez 's' pour stop : ")
7     if reponse == 's' :
8         break
9
```

Exemple - Gestion de stock - v1

- On saisit des noms de fruit, ainsi que les quantités ;
- A chaque saisie, on met à jour un fichier de stock ;
- On recommence jusqu'à la saisie d'un message d'arrêt ;
- A la fin un fichier **stock.txt** existe avec le détail du stock.

```
>>> %Run stock.py  
entrez le fruit : pomme  
entrez la quantité : 12  
entrez 's' pour stop :  
entrez le fruit : poire  
entrez la quantité : 17  
entrez 's' pour stop :  
entrez le fruit : banane  
entrez la quantité : 5  
entrez 's' pour stop : s
```

```
>>>
```

A screenshot of a text editor window titled "stock.txt" with the following content:

```
1 pomme:12  
2 poire:17  
3 banane:5
```

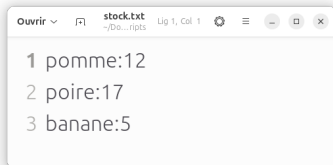
Exemple - Gestion de stock - v2

- On récupère le fichier stock.txt en mode lecture ;
- Chaque ligne est découpée en un fruit et une quantité ;
- On alimente ainsi un dictionnaire python dynamique.

```
1 with open('stock.txt', 'r') as fichier_stock :
2     lignes = fichier_stock.readlines()
3
4 stock = {}
5 for ligne in lignes :
6     # pour retirer le \n de fin de ligne
7     contenu = ligne[0:-1]
8     # pour extraire du contenu le fruit et sa quantité
9     liste = contenu.split(':')
10    fruit = liste[0]
11    quantite = int(liste[1])
12    # pour créer dans le dictionnaire stock l'entrée correspondante
13    stock[fruit] = quantite
14
15 print(stock)
16
```

Exemple - Gestion de stock - v2

- On récupère le fichier stock.txt en mode lecture ;
- Chaque ligne est découpée en un fruit et une quantité ;
- On alimente ainsi un dictionnaire python dynamique.



```
Ouvrir  stock.txt  
~/De...ripts  Lig 1, Col 1  
1 pomme:12  
2 poire:17  
3 banane:5
```



```
>>> %Run stock2.py  
{'pomme': 12, 'poire': 17, 'banane': 5}  
>>>
```

Exemple - Gestion de stock - v3

```
stock3.py
1 with open('stock.txt', 'r') as fichier_stock :
2     lignes = fichier_stock.readlines()
3
4 stock = {}
5 for ligne in lignes :
6     contenu = ligne[0:-1]
7     liste = contenu.split(':')
8     fruit = liste[0]
9     quantite = liste[1]
10    stock[fruit] = quantite
11
12 print(stock)
13
14 while True :
15     reponse = input("voulez-vous modifier le stock (o/n) ? ")
16     if reponse != "o" :
17         break
18     else :
19         fruit = input("quel fruit ? ")
20         quantite = int(input("quelle quantité ? "))
21         stock[fruit] = quantite
22         print(stock)
23
24 with open('stock.txt', 'w') as fichier_stock :
25     fichier_stock.write('')
26
27 with open('stock.txt', 'a') as fichier_stock :
28     for fruit, quantite in stock.items() :
29         fichier_stock.write(fruit + ":" + str(quantite) + "\n")
```

Fichiers .csv

- On peut aussi utiliser des fichiers au format "csv" ;
- Les fichiers .csv (**C**omma **S**epareted **V**alues) sont des fichiers texte, facilement lisibles par des applications tableurs ;
- Si l'application tableur est paramétrée, elle reconnaît le séparateur de valeurs (une virgule, un point-virgule, une tabulation, parfois " :" etc), alors on a l'impression de travailler avec un fichier tableur classique ;
- Ainsi, chaque sauvegarde sera facile à relire et modifier via un tableur, ou alors via le programme écrit en python (ou dans un autre langage) ;