Université Claude Bernard Lyon 1

Fichiers

Nicolas Pronost

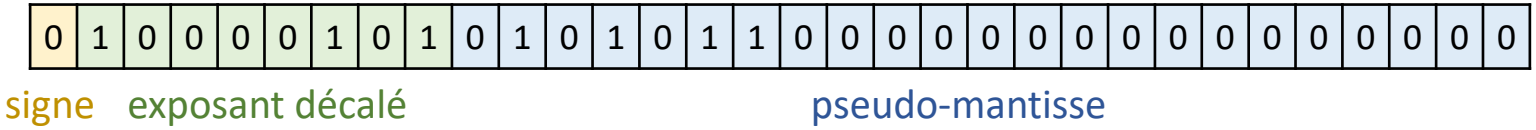
LIRIS

Différents types de fichiers

Format des informations	
« texte » (codes de caractères)	« binaire » (comme en mémoire)
Fichiers .py	Fichiers .pyd
Fichiers .txt	Fichiers mp3, avi
Fichiers de configuration Linux	Fichiers pdf
Fichiers .xml, .html	Fichiers exécutables
Etc.	Etc.

Fichier « texte »

- Les octets du fichier ne contiennent que des codes de caractères
- Les données numériques doivent être converties en caractères
- Exemple: un nombre au format IEEE 754 simple précision

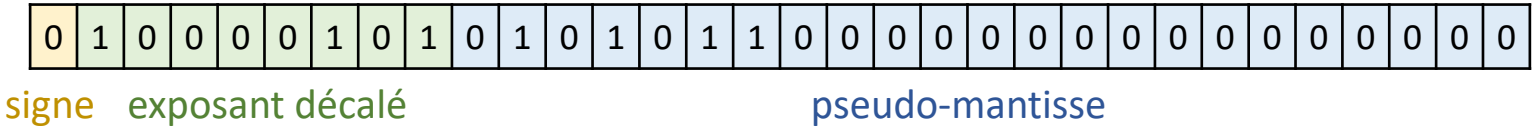


  - Décodage: $nombre = 1.3359375 \times 2^6 = 85.5 = 8.55e1$
  - Ecriture dans le fichier (virgule fixe, 3 chiffres après la virgule) des caractères: ‘8’ ’5’ ‘.’ ‘5’ ‘0’ ‘0’

00111000	00110101	00101110	00110101	00110000	00110000
(56='8')	(53='5')	(46='.')	(53='5')	(48='0')	(48='0')

Fichier « texte »

- Les octets du fichier ne contiennent que des codes de caractères
- Les données numériques doivent être converties en caractères
- Exemple: un nombre au format IEEE 754 simple précision



  - Décodage: $nombre = 1.3359375 \times 2^6 = 85.5 = 8.55e1$
  - Ecriture dans le fichier (virgule flottante, 3 chiffres pour la mantisse) des caractères: ‘8’ ‘.’ ‘5’ ‘5’ ‘e’ ‘1’

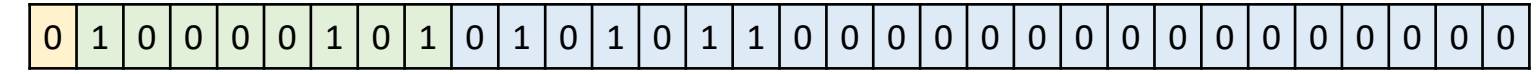
00111000	00101110	00110101	00110101	01100101	00110001
(56=‘8’)	(53=‘.’)	(46=‘5’)	(53=‘5’)	(101=‘e’)	(49=‘1’)

Fichier « texte »

- Avantages
  - portabilité
  - fichier lisible (et facilement modifiable) avec un simple éditeur de texte
- Inconvénients
  - coût en temps induit par les conversions nécessaires
  - penser à écrire suffisamment de décimales pour ne pas perdre en précision numérique

Fichier « binaire »

- On recopie l'information comme elle figure en mémoire
- Exemple



  - Ecriture dans le fichier (regroupement en octet juste pour lisibilité)

```
01000010  10101011  00000000  00000000
```

- Avantages
  - fichier plus compact (n'utilise pas de codage intermédiaire)
  - lecture/écriture sans conversion = manipulation plus rapide
  - pas de formatage de présentation = pas de perte de précision
- Inconvénients
  - fichier illisible (et difficilement éditable) par un éditeur de texte
  - il faut gérer les problèmes de portabilité
  - il faut savoir qu'est ce qui est écrit pour pouvoir le relire

Ouvrir un fichier

- Il nous faut passer par une instance de la classe **file**
- En utilisant la fonction **open**

```
file fichier = open (nomFichier [,modeAccès][, buffering])
```

  - *nomFichier* est le nom du fichier à ouvrir, avec chemin relatif (au fichier Python) ou absolu
  - modeAccès détermine le mode d'ouverture
  - buffering
    - <= 0 : pas de buffer
    - 1 : buffer de ligne
    - > 1 : taille du buffer

Mode d’accès

- Les modes d’accès à un fichier sont

‘r’	ouverture en lecture seule, lecture à partir du début, en texte
‘rb’	ouverture en lecture seule, lecture à partir du début, en binaire
‘r+’	ouverture en lecture/écriture, lecture à partir du début, en texte
‘rb+’	ouverture en lecture/écriture, lecture à partir du début, en binaire
‘w’	ouverture en écriture seule, écrase le contenu précédent, en texte
‘wb’	ouverture en écriture seule, écrase le contenu précédent, en binaire
‘w+’	ouverture en écriture/lecture, écrase le contenu précédent, en texte
‘wb+’	ouverture en écriture/lecture, écrase le contenu précédent, en binaire
‘a’	ouverture en ajout, écriture à partir de la fin, en texte
‘ab’	ouverture en ajout, écriture à partir de la fin, en binaire
‘a+’	ouverture en ajout/lecture, écriture à partir de la fin, en texte
‘ab+’	ouverture en ajout/lecture, écriture à partir de la fin, en binaire

Attributs de la classe file

- Une fois un fichier ouvert, vous obtenez une instance de la classe file
- Avec les fonctions membres
  - file.closed : retourne vrai si le fichier est fermé, faux sinon
  - file.mode : retourne le mode d'accès utilisé pour ouvrir le fichier
  - file.name : retourne le nom du fichier
- Exemple

```
monFichier = open ("fichier.txt" , 'wb')
print("Nom du fichier: ", monFichier.name)
print("Fermé? ", monFichier.closed)
print("Mode d'accès: ", monFichier.mode)
```

Fermeture d'un fichier

- Quand les opérations d'écriture et de lecture sont finies, il ne faut pas oublier de fermer le fichier

```
monFichier.close();
```

- Afin de notifier l'OS qu'il peut libérer la ressource et la rendre disponible à nouveau
- En effet, deux programmes (ou deux blocs d'instructions d'un même programme) n'ont pas le droit d'avoir le même fichier ouvert en même temps
  - pour des raisons évidentes de conflits

Ecriture dans un fichier

- La procédure **write** écrit n'importe quelle chaîne de caractères dans un fichier ouvert
  - les string de Python peuvent avoir des données binaires
- Attention, cette procédure n'ajoute pas d'espace ni de retour à la ligne à la fin de la chaîne de caractères
- Exemple

```
monFichier = open ("fichier.txt" , 'w')
monFichier.write("l'informatique c'est génial")
monFichier.write(" et les maths ...\naussi!")
monFichier.close()
```

  - contenu de « fichier.txt » :

```
l'informatique c'est génial et les maths ...
aussi!
```

Ecriture dans un fichier

- La procédure **writelines** permet décrire une liste de chaînes de caractères dans le fichier
- Exemple

```
monFichier = open ("fichier.txt" , 'w')
liste = ["l'informatique c'est génial", " et ", "les maths ...\naussi!"
monFichier.writelines(liste)
monFichier.close()
```

  - produit le même contenu que précédemment

Lecture depuis un fichier

- La fonction **read** lit une chaîne de caractères depuis un fichier ouvert

```
string chaine = monFichier.read([NbOctets])
```

- Le paramètre optionnel NbOctets indique le nombre d'octets à lire depuis le fichier
- Exemple (sur le fichier produit juste avant)

```
monFichier = open ("fichier.txt" , 'r')
str = monFichier.read(10)
print("Chaine lue: ",str)
monFichier.close()
```

  - contenu de str après exécution : « l'informat »
  - le reste serait lu grâce à un autre appel à **read**
    - lit jusqu'à la fin si pas de taille en paramètre

Lecture depuis un fichier

- Vous pouvez aussi lire le fichier ligne par ligne avec la fonction **readline**

```
string chaine = monFichier.readline()
```

- Exemple

```
monFichier = open ("fichier.txt" , 'r')
str1 = monFichier.readline()
str2 = monFichier.readline()
str3 = monFichier.readline()
monFichier.close()
```

  - contenu après exécution
    - str1 = « l'informatique c'est génial et les maths ...\n»
    - str2 = « aussi. »
    - str3 = « »

Lecture depuis un fichier

- Une autre fonction utile est **readlines** qui retourne la liste des lignes restantes jusqu'à la fin du fichier
- Exemple

```
monFichier = open ("fichier.txt" , 'r')
strs = monFichier.readlines()
monFichier.close()
```

  - Contenue de strs après exécution : [« l'informatique c'est génial et les maths ...\n», « aussi. »]

Appels systèmes sur les fichiers

- Dans Python, on peut directement appeler des fonctions systèmes de manipulations des fichiers grâce au module **os**
  - indépendantes du système d'exploitation utilisé
- Renommer un fichier

```
os.rename(nomCourant,nouveauNom)
```

- Supprimer un fichier

```
os.remove(nomFichier)
```

- Créer un répertoire

```
os.mkdir(nomRepertoire)
```

- Changer de répertoire courant

```
os.chdir(nomRepertoire)
```

- Supprimer un répertoire

```
os.rmdir(nomRepertoire)
```