

## Séance 6

# Lecture et écriture de fichiers



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

# Rappels

- Programmation **défensive**
  - Gestion d'erreur avec valeur de retour spéciale
  - Spécification de fonction (pré et postconditions)
  - Instruction assert
- Utiliser et créer des **exceptions**
  - Capturer une erreur avec l'instruction `try-except-finally`
  - Générer une erreur avec l'instruction `raise`
  - Définir une nouvelle exception

# Objectifs

- Lire et écrire des **fichiers textes**
  - Ouverture et fermeture d'un fichier
  - Fonctions de lecture et d'écriture
  - Encodage des caractères
- Lire et écrire des **fichiers binaires**
  - Fonctions de lecture et d'écriture
  - Définition du format

# Fichier



# Fichier

- Un **fichier** stocke des informations sur le disque dur
  - Il est créé, modifié, supprimé*
- Manipulation des fichiers grâce au **système d'exploitation**
  - L'interpréteur fait le relais avec Linux, Windows, Mac OS X...*
- Des **informations** sont associées à un fichier
  - Nom, taille, date de création, dernière date de modification...*

# Type de fichier

- Au plus bas niveau, un fichier est une **séquence de bits** (0 et 1)  
*La séquence doit être interprétée pour en obtenir le sens*
- **Deux types** de fichiers manipulés à des niveaux différents
  - Fichier texte est une séquence de caractères ou de lignes
  - Fichier binaire est une séquence d'octets (8 bits)

# Chemin

- Fichier identifié sur une machine à l'aide de son **chemin**

*Identifie l'endroit où se situe le fichier*

- **Deux façons** de spécifier un chemin

- Chemin absolu à partir de la racine
- Chemin relatif à partir du répertoire courant

---

Chemin relatif	Chemin absolu
data.txt	/Users/combefis/Desktop/data.txt
src/program.py	/Users/combefis/Desktop/src/program.py
../image.png	/Users/combefis/image.png
../movies/food.mp4	/Users/combefis/movies/food.mp4

---

# Fichier texte

PROGRAM  
TEXT FILE ?



34  
St

# Ouverture d'un fichier

- Pour manipuler un fichier, il faut d'abord l'**ouvrir**

*On utilise la fonction `open`, qui renvoie un identifiant de fichier*

- **Deux erreurs** possibles lors de l'ouverture

*Fichier introuvable ou erreur d'ouverture*

```
1 try:  
2     file = open('data.txt')  
3 except FileNotFoundError:  
4     print('Le fichier est introuvable')  
5 except IOError:  
6     print("Erreur d'ouverture")
```

# Mode d'ouverture (1)

- Par défaut, fichier ouvert en **lecture seule**

*Tout ce qu'on peut faire c'est donc lire le contenu du fichier*

- On peut spécifier le **mode d'ouverture** désiré

*Avec le deuxième paramètre de la fonction open (mode)*

```
1 try:  
2     file = open('data.txt', 'w')          # Ouverture en écriture  
3 except FileNotFoundError:  
4     print('Le fichier est introuvable')  
5 except IOError:  
6     print("Erreur d'ouverture")
```

# Mode d'ouverture (2)

- Mode d'ouverture définit avec des **caractères**

Caractère	Description
r	Lecture (par défaut)
w	Écriture (avec remise à zéro)
x	Création exclusive (erreur si fichier déjà existant)
a	Écriture (avec ajout à la fin)
b	Mode binaire
t	Mode texte (par défaut)

```
1 try:  
2     file = open('data.txt', 'rt')          # Mode par défaut  
3 except FileNotFoundError:  
4     print('Le fichier est introuvable')  
5 except IOError:  
6     print("Erreur d'ouverture")
```

# Fermeture d'un fichier

- Une fois les opérations finies, il faut **fermer** le fichier

*On utilise la fonction `close` avec l'identifiant de fichier*

- **Libération des ressources** et sauvegarde sur disque

*Le système d'exploitation limite le nombre de fichiers ouverts*

```
1 try:  
2     file = open('data.txt')  
3     file.close()                      # Fermeture du fichier  
4 except FileNotFoundError:  
5     print('Le fichier est introuvable')  
6 except IOError:  
7     print("Erreur d'ouverture")
```

# Lecture

- **Lecture** intégrale du fichier comme une chaîne de caractères

*On utilise la fonction `read` avec l'identifiant de fichier*

- La lecture peut échouer et provoquer une **exception IOError**

*Par exemple si le disque est déconnecté pendant la lecture*

```
1 try:  
2     file = open('data.txt')  
3     print(file.read())  
4     file.close()  
5 except FileNotFoundError:  
6     print('Le fichier est introuvable')  
7 except IOError:  
8     print("Erreur d'entrée/sortie")
```

# Instruction finally (1)

- En cas d'erreur, le fichier pourrait **ne pas être fermé**

*Car l'exécution du code saute directement dans l'except*

- **Instruction finally** exécutée dans tous les cas

*Après la fin du bloc try ou après un except éventuel*

```
1  try:  
2      file = open('data.txt')  
3      print(file.read())  
4  except FileNotFoundError:  
5      print('Le fichier est introuvable')  
6  except IOError:  
7      print("Erreur d'ouverture")  
8  finally:  
9      file.close()
```

# Instruction finally (2)

- Bug dans le code précédent si le fichier n'a pas su être ouvert  
*La variable file ne sera pas initialisée et close disponible*

- On utilise une instruction **try/finally** additionnelle

```
1 try:
2     file = open('data.txt')
3     try:
4         print(file.read())
5     finally:
6         file.close()
7 except FileNotFoundError:
8     print('Le fichier est introuvable')
9 except IOError:
10    print("Erreur d'entrée/sortie")
```

# Instruction with

- Instruction **with** pour fermeture propre des ressources

*L'appel à close sera fait automatiquement*

- Il faut garder le **try/except** pour les IOError

```
1 try:
2     with open('data.txt') as file:
3         print(file.read())
4 except FileNotFoundError:
5     print('Le fichier est introuvable')
6 except IOError:
7     print("Erreur d'entrée/sortie")
```

Facebook:mlt@ecam.be:melo:8dj,Sj0m1  
Skype:mar@ecam.be:cedou:arduino  
Facebook:fle@ecam.be:fingerfood:b8ur,g2er

# Écriture (1)

- Écriture en ajoutant des chaînes de caractères au fichier

*On utilise la fonction `write` avec l'identifiant de fichier*

- L'écriture peut échouer et provoquer une exception `IOError`

*Par exemple si l'espace disque devient plein pendant l'écriture*

```
1 with open('out.txt', 'w') as file:  
2     file.write('Table de 10\n')  
3     for i in range(10):  
4         file.write('{0} x 10 = {1}\n'.format(i, i * 10))
```

# Écriture (2)

A screenshot of a Mac OS X desktop environment showing a window with three tabs. The tabs are labeled "program.py", "data.txt", and "out.txt". The "data.txt" tab is currently selected, indicated by a blue border around its title. The window contains a dark gray text area with white text. The text is a multiplication table of 10, starting from line 1 and ending at line 12. Each line shows the result of multiplying 10 by a number from 0 to 9.

Line	Content
1	Table de 10
2	0 x 10 = 0
3	1 x 10 = 10
4	2 x 10 = 20
5	3 x 10 = 30
6	4 x 10 = 40
7	5 x 10 = 50
8	6 x 10 = 60
9	7 x 10 = 70
10	8 x 10 = 80
11	9 x 10 = 90
12	

# Copie d'un fichier

- Copie en faisant une lecture puis écriture du contenu lu

*Deux instruction with imbriquées*

- Si le fichier destination existe déjà, il est effacé

*Il faut utiliser le mode x au lieu de w pour empêcher cela*

```
1 with open('data.txt', 'r') as src, open('copy.txt', 'w') as dest:  
2     dest.write(src.read())
```

# Lecture ligne par ligne (1)

- Utilisation d'un **itérateur** sur le fichier ouvert, avec `for`

*Parcours ligne par ligne, avec le retour de ligne inclus*

- Fonction `rstrip` pour retirer les caractères blancs de droite

```
1 with open('data.txt') as file:  
2     for line in file:  
3         cleaned = line.rstrip()  
4         tokens = cleaned.split(':')  
5         print('Compte {} de {} (mode de passe : {})'.format(tokens  
[0], tokens[2], tokens[3]))
```

```
Compte Facebook de melo (mode de passe : 8dj,Sj0m1)  
Compte Skype de cedou (mode de passe : arduino)  
Compte Facebook de fingerfood (mode de passe : b8ur,g2er)
```

# Lecture ligne par ligne (2)

- L'**itérateur** est un raccourci d'appel de `readline`

*Lis une ligne s'il en reste à lire dans le fichier*

- Renvoie une **chaine de caractères vide** lorsqu'au bout du fichier

*Pratique lorsqu'on sait combien de lignes lire*

```
1 with open('data.txt') as file:  
2     line = file.readline()  
3     while line != '':  
4         cleaned = line.rstrip()  
5         tokens = cleaned.split(':')  
6         print('Compte {} de {} (mode de passe : {})'.format(tokens  
7             [0], tokens[2], tokens[3]))  
8         line = file.readline()
```

# Lecture ligne par ligne (3)

- La fonction `readlines` lis l'**intégralité des lignes** en une traite  
*La fonction renvoie une liste de chaînes de caractères*

- On peut **supprimer la variable** `cleaned` inutile

*En enchainant directement les appels à `rstrip` et `split`*

```
1 with open('data.txt') as file:  
2     content = file.readlines()  
3  
4 for line in content:  
5     tokens = line.rstrip().split(':')  
6     print('Compte {} de {} (mode de passe : {})'.format(tokens[0],  
7                 tokens[2], tokens[3]))
```

# Lecture ligne par ligne (4)

- Amélioration du code avec une fonction de formatage

*On définit une fonction qui formate une ligne*

- On définit une liste par compréhension et on joint ses éléments

*Jointure des éléments d'une liste réalisée avec fonction join*

```
1 def format(line):
2     tokens = line.rstrip().split(':')
3     return 'Compte {} de {} (mode de passe : {})'.format(tokens[0],
4         tokens[2], tokens[3])
5
6 with open('data.txt') as file:
7     content = file.readlines()
8
9 print('\n'.join([format(line) for line in content]))
```

# Exception

- L'erreur principale d'**entrée/sortie** est `IOError`

*On peut se limiter à capturer cette unique erreur*
- Erreur **spécialisée** selon le type précis
  - `FileNotFoundException`, si le fichier n'est pas trouvé
  - `FileExistsError`, si le fichier existe déjà
  - `PermissionError`, si l'utilisateur n'a pas les droits d'accès
  - `IsADirectoryError`, si le fichier est en fait un dossier

# Encodage (1)

- Les caractères sont **stockés au format binaire** dans l'ordinateur

*Table de correspondance associant un entier à chaque caractère*

- La **table de caractères ASCII** (iso-646) contient 128 caractères

*Table suffisante pour des textes en anglais*

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STH	ETH	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	CD2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	spc	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	-
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

# Encodage (2)

- La fonction `ord` donne le **code** d'un caractère

*Sous forme d'un nombre entier*

- La fonction `chr` donne le **caractère** correspondant à un code

*Sous forme d'une chaîne de caractères*

```
1 print(chr(65))          # Affiche A  
2  
3 print(ord('z'))        # Affiche 90
```

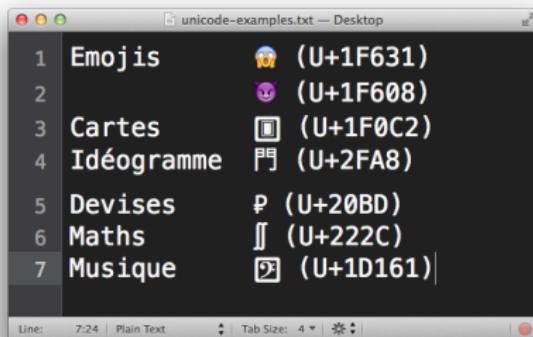
# Unicode et UTF-8

- **Unicode** (ISO 10646) est un standard d'échange de texte

*Associe à tout caractère un nom et un identifiant numérique*

- **UTF-8** est un encodage pour les caractères Unicode

*Python travaille par défaut avec l'encodage UTF-8*



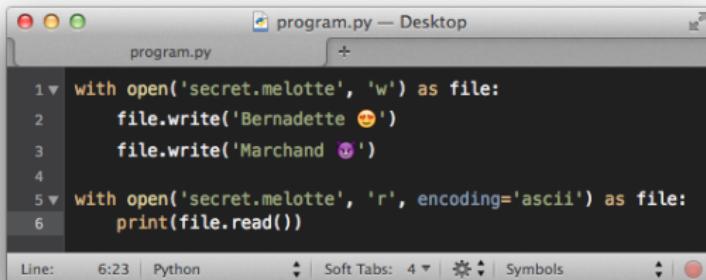
A screenshot of a terminal window titled "unicode-examples.txt" on a desktop. The window displays a list of seven entries, each consisting of a number, a name, and a character followed by its hex code in parentheses. The names include "Emojis", "Cartes", "Idéogramme", "Devises", "Maths", and "Musique". The terminal interface includes a menu bar, a status bar at the bottom with "Line: 7/24 | Plain Text" and "Tab Size: 4", and a scroll bar on the right.

1	Emojis	😱 (U+1F631)
2		Pussy (U+1F608)
3	Cartes	🃏 (U+1F0C2)
4	Idéogramme	門 (U+2FA8)
5	Devises	₽ (U+20BD)
6	Maths	⅀ (U+222C)
7	Musique	🎵 (U+1D161)

Parcourir les caractères Unicode en ligne : <http://unicode-table.com>

# Choisir l'encodage

- On spécifie l'**encodage des fichiers** avec le paramètre encoding



A screenshot of a Python code editor window titled "program.py — Desktop". The code in the editor is:

```
1 with open('secret.melotte', 'w') as file:  
2     file.write('Bernadette 😊')  
3     file.write('Marchand 🍎')  
4  
5 with open('secret.melotte', 'r', encoding='ascii') as file:  
6     print(file.read())
```

The status bar at the bottom shows "Line: 6" and "Time: 6:23".

```
Traceback (most recent call last):  
  File "program.py", line 6, in <module>  
    print(file.read())  
  File "/Library/Frameworks/Python.framework/Versions/3.4/lib/  
    python3.4/encodings/ascii.py", line 26, in decode  
      return codecs.ascii_decode(input, self.errors)[0]  
UnicodeDecodeError: 'ascii' codec can't decode byte 0xf0 in  
position 11: ordinal not in range(128)
```

## Fichier binaire

# Fichier binaire vs texte

- Deux types de fichiers
  - **Fichier texte** : séquence de caractères en UTF-8
  - **Fichier binaire** : séquence d'octets (huit bits)
- Fichier binaire plus compact et rapide à lire/écrire
- Connaitre le **format d'un fichier binaire** pour le manipuler

*L'organisation des données dans le fichier*

# Écriture (1)

- Écrire dans un fichier binaire avec le **module pickle**

*La fonction `dump` écrit un objet dans le fichier*

- Les objets sont **écrits successivement** dans le fichier

*Erreur de type `PicklingError` si l'objet ne sait pas être converti*

```
1 import pickle
2
3 name = "Temperatures (2015)"
4 data = [12, -9, 7, 112, 99]
5 try:
6     with open('data.bin', 'wb') as file:
7         pickle.dump(name, file, pickle.HIGHEST_PROTOCOL)
8         pickle.dump(data, file, pickle.HIGHEST_PROTOCOL)
9 except pickle.PicklingError:
10     print("Erreur d'écriture")
```

## Écriture (2)

- Un fichier binaire n'est pas lisible directement

*Il faut pouvoir interpréter son contenu*

```
<80>^D<95>^W^@^@^@^@^@^@^@^@<8c>^STemperatu  
res (2015)<94>.<80>^D<95>^R^@^@^@^@^@^@^@^@  
]<94>(K^LJ+yyyK^GKpKce.  
~  
~  
~  
~  
~  
"data.bin" [noeol][converted] 1L, 74C
```

# Lecture

- Lire dans un fichier binaire avec le **module pickle**

*La fonction `load` lit un objet depuis le fichier*

- Les objets sont **lus successivement** dans le fichier

*Erreur de type `UnpicklingError` si l'objet ne sait pas être lu*

```
1 import pickle
2
3 try:
4     with open('data.bin', 'rb') as file:
5         name = pickle.load(file)
6         data = pickle.load(file)
7         print(name, data, sep='\n')
8 except pickle.UnpicklingError:
9     print("Erreur de lecture")
```

# Module struct

- Le **module struct** permet de convertir des données en bytes

*Manipulation de données binaires brutes*

- Deux fonctions à utiliser

- pack convertit des données en un bytes
- unpack convertit un bytes en une donnée

---

Caractère	Description
h	Nombre entier signé (court)
H	Nombre entier non signé (court)
i	Nombre entier signé
I	Nombre entier non signé
f	Nombre flottant
s	Chaine de caractères
?	Booléen

---

# Encodage/décodage de chaînes de caractères

- **Encodage** d'un str vers un bytes avec encode

*On spécifie l'encodage désiré en paramètre*

- **Décodage** d'un bytes vers un str avec decode

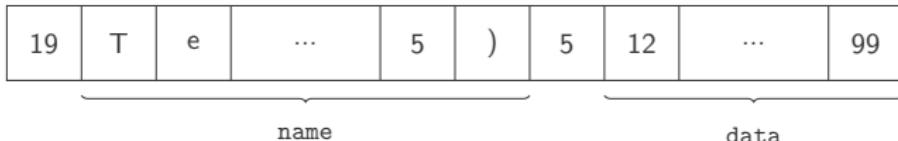
```
1 s = "Hello"
2
3 data = s.encode('utf-8')
4 print(type(data))
5 print(data)
6
7 print(data.decode('utf-8'))
```

```
<class 'bytes'>
b'Hello'
Hello
```

# Écriture avec pack

## ■ Écritures successives dans le fichier avec `struct.pack`

*On définit le format de la donnée à écrire, et la donnée*



```
1 import struct
2
3 name = "Temperatures (2015)"
4 data = [12, -9, 7, 112, 99]
5
6 with open('data2.bin', 'wb') as file:
7     n = len(name)
8     file.write(struct.pack('H', n))
9     file.write(struct.pack('{}s'.format(n), name.encode('utf-8')))
10    file.write(struct.pack('H', len(data)))
11    for elem in data:
12        file.write(struct.pack('h', elem))
```

# Lecture avec unpack

- Lectures successives dans le fichier avec `struct.unpack`

*On définit le format de la donnée à lire, et l'objet bytes*

```
1 import struct
2
3 with open('data2.bin', 'rb') as file:
4     n = struct.unpack('H', file.read(2))[0]
5     name = struct.unpack('{}s'.format(n), file.read(n))[0].decode(
6         'utf-8')
7     n = struct.unpack('H', file.read(2))[0]
8     data = [struct.unpack('H', file.read(2))[0] for i in range(n)]
9     print(name, data, sep='\n')
```

# pickle ou struct

- pickle plus facile à utiliser et plus **automatique**

*Ne nécessite pas de définir le format de stockage des données*

- struct plus flexible et plus **compact**

*63 octets pour data.bin contre 33 octets pour data2.bin*

# Crédits

- <https://www.flickr.com/photos/quinnanya/2493803790>
- <https://www.flickr.com/photos/jasoneppink/2081701562>
- <https://www.flickr.com/photos/rakka/123380632>