

Introduction à l'informatique

Cours 5

kn@lmf.cnrs.fr

<https://usr.lmf.cnrs.fr/~kn>



- 1 Présentation du cours ✓
- 2 Le système Unix ✓
- 3 Le système Unix (2) ✓
- 4 Python (1) : expressions, types de bases, if/else ✓
- 5 Python (2) : boucles, tableaux, exceptions ✓
- 6 Python (3) : Textes, chaînes de caractères, entrées/sorties
 - 6.1 Le standard UTF-8
 - 6.2 Chaînes de caractères
 - 6.3 Entrées et sorties
 - 6.4 Étude de cas

Représentation des textes



Comment sont représentés les textes dans un ordinateur ?

(on ne parle pas ici de documents riches comme des PDFs ou des fichiers .docx ou HTML, mais simplement des fichiers textes)

Exemple de texte réaliste:

" سلام, Здравей, ¡Hola!, Πίψι, Γειά σου, 你好, Góðan daginn, 안녕, こんにちは, Salut, 🖐️, ... "

Historiquement...



Encodage 1 caractère = 1 octet (8 bits) :

- ◆ Encodage ASCII sur 7 bits (128 caractères)
- ◆ ASCII étendu 8 bits (256 caractères, dont 128 de « symboles »)
- ◆ Latin 1 : ASCII 7 bits + 128 caractères « ouest-européens » (lettres accentuées française, italienne, ...)
- ◆ Latin 2 : ASCII 7 bits + 128 caractères « est-européens » (Serbe, Hongrois, Croate, Tchèque, ...)
- ◆ Latin 3 : ASCII 7 bits + 128 caractères turques, maltais, espéranto,
- ◆ Latin 4 : ASCII 7 bits + 128 caractères islandais, lituanien, ...
- ◆ ...
- ◆ Latin 15 : Latin 1 avec 4 caractères « inutiles » remplacés (par exemple pour « € » à la place de « ¤ »)

... et pendant ce temps là, ailleurs dans le monde

Encodage multi-octets:

- ◆ Encodages spécifiques pour le Chinois (Big5, GB, ...)
- ◆ Encodages spécifiques pour le Japonais (Shift-JIS, EUC, ...)

Impossibilité de mettre plusieurs « alphabets » dans un même texte

Chaque logiciel « interprétait » les séquences d'octet de manière prédéfinie

Unicode



Un standard international (ISO) qui donne à chaque caractère un code :

A → 65

B → 66

...

y → 121

...

é → 233

...

α → 945

...

↵ → 8623

Rajouter un nouvel alphabet ou un nouveau caractère est un processus complexe.

Actuellement 143,859 caractères (154 « alphabets » et symboles, dont les symboles Latins, le Chinois, Arabe, ...)

Encodage ?



Unicode permet un maximum de 1,111,998 (donc on a attribué ~ 13% des caractères possibles, on a de la marge).

Mais comment stocker ces codes dans un fichier texte par exemple (qui est une suite d'octets) ?

Rappel:

1 octet → 8 bits → 256 valeurs

2 octets → 16 bits → 65 536 valeurs

3 octets → 24 bits → 16 777 216 valeurs

On pourrait utiliser 3 octets pour chaque caractère :

00 00 41	00 00 42	00 03 b1	...
A	B	a	

- ◆ Les textes « courants » gâchent de l'espace (00 00...)
- ◆ Pas compatible avec ascii 7bits (donc il faut convertir tous les vieux fichiers textes)

UTF-8



Universal (Character Set) Transformation Format 8 bit

- ◆ Encodage à taille variable « universel » (permet de stocker tout Unicode)
- ◆ Assez simple
- ◆ Compatible avec ASCII 7 bits

Encodage

Nombre d'octets	valeurs	Octet 1	Octet 2	Octet 3	Octet 4
1	0-127	0xxxxxxx			
2	127-2047	110xxxxx	10xxxxxx		
3	2048-65535	1110xxxx	10xxxxxx	10xxxxxx	
4	65536-1114111	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

(dans le derniers cas, certains codes sont inutilisés)


Exemples



A → 65_{10} → $0100\ 0001_2$ (représenté sur un seul octet)

ë → 7877_{10} → $0001\ 1110\ 1100\ 0101_2$ (représenté 3 octets) :

$11100001\ 1011\ 10\ 11\ 1000\ 0101 \equiv 225\ 187\ 133$

 → 128053_{10} → ... $\equiv 240\ 159\ 144\ 181$

Avantages :

- ◆ compatible ASCII 7 bits (d'anciens documents texte en anglais sont toujours lisibles)
- ◆ pas d'espace gaspillé (à l'inverse d'UTF-32 ou tous les caractères font 32 bits)

Inconvénients :

- ◆ Caractères à taille variable: il faut parcourir le texte pour trouver le n^{ème} caractère
- ◆ Les vieux logiciels doivent être adaptés

Si ça vous est déjà arrivé



É en Unicode est le caractère 201. C'est plus grand que 127, donc codé sur deux octets :
195 137

En Latin 1 : 195 → Ã, 137 → ©

Beaucoup d'autres choses



- ◆ Caractères précomposés vs. composition: é / ◌é

Ces deux « caractères » sont égaux (en Français), convertir de l'un à l'autre, ...

- ◆ Ligatures

- ◆ Direction d'écriture (droite gauche vs gauche droite)

- ◆ Spécificités régionales.

En français : I (majuscule) → i (minuscule)

En turc : I (maj.) → ı (min.) et İ (maj.) → i (min.)

- ◆ ...



- 1 Présentation du cours ✓
- 2 Le système Unix ✓
- 3 Le système Unix (2) ✓
- 4 Python (1) : expressions, types de bases, if/else ✓
- 5 Python (2) : boucles, tableaux, exceptions ✓
- 6 Python (3) : Textes, chaînes de caractères, entrées/sorties
 - 6.1 Le standard UTF-8 ✓
 - 6.2 Chaînes de caractères
 - 6.3 Entrées et sorties
 - 6.4 Étude de cas

Retour sur les chaînes



Les chaînes de caractères sont le type de données permettant de représenter du texte.

```
>>> t = 'Bonjour, ça va ?'
>>> print(t)
Bonjour, ça va ?
>>> t + ' oui, ça va !'
'Bonjour, ça va ? oui, ça va !'
>>> len(t)
16
>>>
```

Séquences d'échappement



Que se passe-t-il si on veut insérer un caractère « ' » dans une chaîne ?

```
>>> 'C'est moi!'
File "", line 1
  'C'est moi!'
    ^
SyntaxError: invalid syntax
```

On doit indiquer que le « ' » ne marque pas la fin de la chaîne. On utilise une **séquence d'échappement** :

```
>>> 'C\'est moi!'
'C\'est moi!'
>>> print('C\'est moi!')
C'est moi!
>>>
```

Attention, le programmeur saisit **deux** caractères (« \' ») mais Python n'en retient qu'un seul (« ' »).

D'autres séquences d'échappement



Comment saisir un caractère \ dans une chaîne ?

Imaginons qu'on veuille le mettre en dernière caractère d'une chaîne :

```
>>> 'Caractère antislash: \''
File "", line 1
    'Caractère antislash: \''
                        ^
SyntaxError: EOL while scanning string literal
```

Il faut aussi échapper le caractère \

```
>>> 'Caractère antislash: \\'
'Caractère antislash: \\'
```

Il existe d'autres séquences d'échappement : \n (retour à la ligne), \uxxxx (code Unicode en base 16), ...

```
>>> '\u0041\n\u0042'
'A\nB'
```

D'autres chaînes



En Python, on peut aussi délimiter une chaîne de caractères par " ou encore par ""

- ◆ Les chaînes " sont comme les chaînes '. Elles permettent juste d'éviter les \ si on a beaucoup de ': **"Plein d'apostrophes c'est super !"**
- ◆ Les chaînes "" permettent aussi les retour à la ligne dans la chaîne

```
>>> """Je peux
... écrire une chaîne
... sur
... plusieurs
... lignes!"""
'Je peux\nécrire une chaîne\nsur\nplusieurs\nlignes!'
```


Manipulation des chaînes



On peut accéder aux n^{ème} caractère d'une chaîne comme si c'était un tableau

```
>>> t = 'Bonjour'
>>> t[3]
'j'
```

L'opération [...] renvoie une chaîne de taille 1 contenant uniquement le caractère ce trouvant à cet endroit.

Par contre, il n'est pas possible de mettre à jour une chaîne :

```
>>> t[3] = 'X'
Traceback (most recent call last):
  File "", line 1, in
TypeError: 'str' object does not support item assignment
>>>
```

Autres opérations



En Python les chaînes sont représentée en UTF-8

La fonction `chr(n)` renvoie le caractère dont le code Unicode est `n`.

La fonction `ord(s)` affiche le code Unicode du premier caractère de la chaîne `s`

```
>>> chr(65)
'A'
>>> chr(945)
'a'
>>> chr(128169)
'🐼'
>>> ord('B')
66
>>> ord('😎')
128526
```

⚠️ : pour les « emoji », il faut que les polices soient installées correctement sinon on risque de voir un caractère de substitution « ❓ »

Notations pointée



En Python, certaines opérations ne sont pas des fonctions (comme `len(...)`, `ord(...)`, `chr(...)`) mais doivent être appelées directement sur la valeur à laquelle on veut appliquer l'opération.

C'est le cas de nombreuses opérations sur les chaînes.

```
>>> t='Bonjour, ça va ?'
>>> t.upper()
'BONJOUR, ÇA VA ?'
>>> t.lower()
'bonjour, ça va ?'
>>> t.split(' ')
['Bonjour', 'ça', 'va', '?']
>>> u=['Oui', 'ça', 'va']
>>> "_".join(u)
'Oui_ça_va'
```

Attention, il est encore un peu tôt pour justifier de cette notation. Donc on ne l'explique pas pour l'instant.

Fonctions sur les chaînes



Si on a une chaîne **t** :

- ◆ **t.upper()** : renvoie une copie de la chaîne en majuscules
- ◆ **t.lower()** : renvoie une copie de la chaîne en minuscules
- ◆ **t.split(c)** : découpe la chaîne en utilisant le premier caractère de **c** comme séparateur et renvoie les éléments dans un tableau
- ◆ **t.join(tab)** : prends les chaînes du tableau **tab** et les concatène en les séparant par **t**.
C'est équivalent à **tab[0] + t + tab[1] + t + ... + t + tab[len(tab)-1]**



- 1 Présentation du cours ✓
- 2 Le système Unix ✓
- 3 Le système Unix (2) ✓
- 4 Python (1) : expressions, types de bases, if/else ✓
- 5 Python (2) : boucles, tableaux, exceptions ✓
- 6 Python (3) : Textes, chaînes de caractères, entrées/sorties
 - 6.1 Le standard UTF-8 ✓
 - 6.2 Chaînes de caractères ✓
 - 6.3 Entrées et sorties
 - 6.4 Étude de cas

Retour sur print



On a utiliser `print` pour afficher **une chaîne de caractères**.

En fait on peut passer un nombre arbitraire de valeur à `print`, qui ne sont pas forcément des chaînes (la fonction `str(...)` est alors appelée automatiquement).

Toute les valeurs sont affichées, séparées par des **espaces**.

```
>>> print('Bonjour', 'ça', 'va', '?')
Bonjour, ça va ?
>>> print(1, 'A', True, [1,2,3])
1 A True [1, 2, 3]
>>>
```

Lecture et écriture de fichiers



En python, la fonction `open(chemin, mode)` permet d'ouvrir un fichier pour le lire.

◆ **chemin** est une chaîne de caractères contenant le chemin vers le fichier. Le chemin peut être absolu (commencer par `/`) ou relatif. Il ne peut **pas** contenir de caractères spéciaux du shell tel que `~` ou des motifs glob (`[a-z]*.txt`)

◆ **mode** est une chaîne de caractères indiquant le mode d'ouverture :

r Le fichier est ouvert en lecture seule.

w Le fichier est ouvert en écriture seule. Le fichier est créé s'il n'existe pas et vidé de son contenu s'il existe.

w+ Comme **w**, mais aussi accès en lecture.

a Le fichier est ouvert en écriture et lecture. Le fichier est créé s'il n'existe pas. Le contenu est conservé si le fichier existe.

a+ Comme **a**, mais aussi accès en lecture.

Attention aux erreurs



Les opérations sur les fichiers peuvent lever des exceptions (des erreurs)

- ◆ Si le chemin n'existe pas
- ◆ Si le fichier est un répertoire (et pas un fichier)
- ◆ Si on a pas les droits nécessaires (par exemple pas les droits en écriture et qu'on ouvre avec le mode `w`)
- ◆ Si on essaye d'écrire dans le fichier et qu'il n'y a plus de places sur le disque

On pourra utiliser `try/except`: pour rattraper certaines de ces erreurs.

Opérations sur les fichiers



Le résultat de `open(...)` est une valeur spéciale, appelée **descripteur de fichier**. C'est un objet opaque qui possède de nombreuses opérations. On se limite aux plus simples. On suppose que dans le répertoire courant, on a un fichier `test.txt`.

```
>>> f = open("test.txt", "r")
>>> f
<_io.TextIOWrapper name='test.txt' mode='r' encoding='UTF-8'>
>>> lignes = f.readlines()
>>> lignes
['Heureux qui, comme Ulysse, a fait un beau voyage,\n',
 'Ou comme cestuy-là qui conquiert la toison,\n',
 'Et puis est retourné, plein d'usage et raison,\n', 'Vivre entre ses parents le
>>>
```

L'opération `f.readlines()` renvoie le tableau de toutes les lignes du fichier `f`. Les retours à la ligne sont conservés.

Opérations sur les fichiers (2)



```
>>> for i in range(len(lignes)):
...     ligne[i] = ligne[i].upper()
>>> lignes
['HEUREUX QUI, COMME ULYSSE, A FAIT UN BEAU VOYAGE,\n',
 'OU COMME CESTUY-LÀ QUI CONQUIT LA TOISON,\n',
 'ET PUIS EST RETOURNÉ, PLEIN D'USAGE ET RAISON,\n", ... ]
>>> f2 = open ("test2.txt", "w")
>>> f2.writelines(lignes)
>>> f2.close()
>>> # on quitte Python et on est dans le terminal

$ ls
test.txt test2.txt
$ cat test2.txt

...
PLUS MON LOIR GAULOIS, QUE LE TIBRE LATIN,
PLUS MON PETIT LIRÉ, QUE LE MONT PALATIN,
ET PLUS QUE L'AIR MARIN LA DOULCEUR ANGEVINE.
```

Opérations sur les fichiers (3)



L'opération `f.writelines(tab)` écrit le tableau de chaînes de caractères dans le fichier `f`. Lorsque l'on a fini, il faut refermer le fichier en appelant `f.close()` sinon il se peut que certaines lignes ne soient pas écrites dans le fichier.

Plan



- 1 Présentation du cours ✓
- 2 Le système Unix ✓
- 3 Le système Unix (2) ✓
- 4 Python (1) : expressions, types de bases, if/else ✓
- 5 Python (2) : boucles, tableaux, exceptions ✓
- 6 Python (3) : Textes, chaînes de caractères, entrées/sorties
 - 6.1 Le standard UTF-8 ✓
 - 6.2 Chaînes de caractères ✓
 - 6.3 Entrées et sorties ✓
 - 6.4 Étude de cas



On souhaite écrire un petit programme Python qui compte le nombre de lignes et de caractères dans un fichier.

- ◆ Le chemin vers le fichier est placé dans une variable **chemin**
- ◆ En cas d'erreur, on affiche un message à l'utilisateur

Le programme



```
chemin = "test.txt"      #Modifier pour changer de fichier
try:
    f = open(chemin, "r")
    lignes = f.readlines()
    f.close()
    nb_car = 0
    for i in range(len(lignes)):
        nb_car = nb_car + len(lignes[i])
    print("Il y a", len(lignes), "lignes dans le fichier")
    print("Il y a", nb_car, "caractères dans le fichier")

except FileNotFoundError:
    print("Le fichier", chemin, "n'existe pas !")

except IsADirectoryError:
    print(chemin, "est un répertoire !")

except PermissionError:
    print("Vous n'avez pas les droits en lecture sur le fichier", chemin)

except UnicodeDecodeError:
    print("Le fichier", chemin, "n'est pas un fichier contenant du texte")
```

Conclusion



On n'a vu comment représenter des textes (en UTF-8) et lire et écrire des fichiers de façon sommaire.

- ◆ C'est juste le début, mais suffisant pour commencer à écrire des programmes sympatiques
- ◆ Attention ! Il n'y a pas de **Undo** si vous écrasez un fichier important en écrivant dedans, il est perdu !
- ◆ Il existe des opérations beaucoup plus « bas niveau » sur les fichiers (lecture caractères par caractères, octet par octet, déplacement, ...)

Les manipulations de chaînes et de fichiers seront un bon prétexte pour continuer à travailler sur les tableaux et les boucles **while** et **for**.

