

Caractères accentués

Ce complément expose quelques bases concernant les caractères accentués, et notamment les précautions à prendre pour pouvoir en insérer dans un programme python. Nous allons voir que cette question, assez scabreuse, dépasse très largement le cadre de python *per se*.

Complément - niveau basique

Notre premier conseil, si vous voulez éviter les soucis et vous concentrer sur notre objectif qui est d'apprendre le langage python, est autant que possible **d'éviter d'utiliser des accents** dans le code source.

D'ailleurs nous vous recommandons dans toute la mesure du possible d'écrire votre code en anglais, comme c'est le cas pour la quasi-totalité du code que vous serez amenés à utiliser sous forme de librairies – même si dans ce cours nous avons essayé de produire nos exemples en français.

Complément - niveau intermédiaire

Où peut-on mettre des accents ?

Cela étant dit, si vous devez vraiment mettre des accents dans vos sources, sachez toutefois que:

- il n'est **pas possible** d'utiliser un caractère accentué dans le **nom d'une variable** (ou d'un identificateur au sens large), cela n'est tout simplement pas autorisé par le langage

```
❏ # ceci est interdit par le langage :
```

```
nb_élèves = 1
```

- Le seul endroit du programme – hors commentaires – où vous pouvez mettre des accents est dans les chaînes de caractères. Cela dit, les données manipulées par un programme proviennent pour l'essentiel de sources externes, comme une base de données ou un formulaire web, mais pas directement du code source. Les chaînes de caractères présentes dans du vrai code sont bien souvent limitées à des messages de logging, et le plus souvent d'ailleurs en anglais, donc sans accent.

```
❏ message = "on peut mettre un caractère accentué dans une chaîne"
```

- Enfin on peut aussi envisager de mettre des caractères accentués dans les commentaires, si on choisit malgré tout d'écrire le code en français




```
# on peut mettre un caractère accentué dans un commentaire
```


Précautions à prendre

Si votre ordinateur est configuré pour le français, il y a une chance pour que vous puissiez faire tourner localement des sources contenant des accents – par exemple écrits avec IDLE – sans prendre de disposition particulière.


Toutefois, nous vous conseillons dans ce cas de prendre l'habitude de faire figurer dans vos fichiers, **en première ou deuxième ligne**, une déclaration comme ceci

```
 # -*- coding: <nom_de_l_encodage> -*-
```

Ainsi il est sans doute raisonnable d'utiliser ceci :


```
 # -*- coding: utf-8 -*-
```

Le nom **utf-8** fait référence à **Unicode** (ou pour être précis, à l'encodage le plus répandu parmi ceux qui sont définis dans la norme Unicode, comme nous le verrons plus bas). Sur certains systèmes plus anciens vous pourriez être amenés à utiliser un autre encodage. Pour déterminer la valeur à utiliser dans votre cas précis vous pouvez faire dans l'interpréteur interactif :

```
 # ceci doit être exécuté sur votre machine
```

```
import sys print sys.getdefaultencoding()
```

Par exemple avec d'anciennes versions de Windows vous pouvez être amenés à écrire :

```
 # -*- coding: cp1252 -*-
```

La syntaxe de la ligne `coding` est précisée dans [cette documentation]
(https://docs.python.org/2/reference/lexical_analysis.html)
(https://docs.python.org/2/reference/lexical_analysis.html)

encoding-declarations

).

Qu'est-ce qu'un encodage ?

Comme vous le savez, la mémoire – ou le disque – d'un ordinateur ne permet que de stocker des représentations binaires. Il n'y a donc pas de façon "naturelle" de représenter un caractère comme 'A' ou un guillemet ou un point-virgule.

On utilise pour cela un encodage; par exemple le code US-ASCII – <http://www.asciitable.com/>

(<http://www.asciitable.com/>) – stipule pour faire simple qu'un 'A' est représenté par l'octet 65 qui s'écrit en binaire 01000001. Il se trouve qu'il existe plusieurs encodages, bien sûr incompatibles, selon les systèmes et les langues. Vous trouverez plus de détails ci-dessous.

Le point important est que pour pouvoir ouvrir un fichier "proprement", il faut disposer du contenu du fichier, mais il faut aussi connaître l'encodage.

Le grand malentendu

Cela signifie que si je vous envoie un fichier contenant du français encodé avec, disons, ISO-latin-15 -- http://en.wikipedia.org/wiki/ISO/IEC_8859-15 (http://en.wikipedia.org/wiki/ISO/IEC_8859-15) -- vous pouvez voir dans la table qu'un caractère '€' va être matérialisé dans mon fichier par un octet '0xA4', soit 164.

Imaginez maintenant vous essayez d'ouvrir ceci depuis un ordinateur Windows configuré pour le français. Si on ne lui donne aucune indication sur l'encodage, le programme qui va lire ce fichier sur windows va utiliser l'encodage par défaut du système, c'est-à-dire cp1252 – <http://en.wikipedia.org/wiki/Windows-1252> (<http://en.wikipedia.org/wiki/Windows-1252>). Comme vous le voyez dans cette table, l'octet '0xA4' correspond au caractère ¤ et c'est ça que vous allez voir à la place de €.

C'est à cela que sert la balise # -*- coding: <nom_de_l_encodage> -*-

De cette manière python a les moyens d'interpréter correctement votre fichier source car il sait quel encodage utiliser.

Pourquoi ça marche en local ?

Lorsque le producteur (le programme qui écrit le fichier) et le consommateur (le programme qui le lit) tournent dans le même ordinateur, tout fonctionne bien – en général – parce que les deux programmes se ramènent à l'encodage défini comme défaut. On a vu pourquoi il vaut mieux toutefois être explicite, et spécifier la balise coding :

Il y a une limite toutefois; si vous utiliser un linux configuré de manière minimale, il se peut qu'il utilise par défaut l'encodage US-ASCII – voir plus bas – qui étant très ancien ne "connaît" pas un simple é, ni a fortiori €. Pour écrire du français il faut donc au minimum que l'encodage par défaut de votre ordinateur contienne les caractères français, comme par exemple

- iso-latin-1
- iso-latin-15
- utf-8
- cp1252

Un peu d'histoire sur les encodages

Le code US-ASCII

Jusque dans les années 1980, les ordinateurs ne parlaient pour l'essentiel que l'anglais. La première vague de standardisation avait créé l'encodage dit ASCII, ou encore US-ASCII – voir

par exemple <http://www.asciitable.com> (<http://www.asciitable.com>) , ou en version longue <http://en.wikipedia.org/wiki/ASCII> (<http://en.wikipedia.org/wiki/ASCII>) .

Le code ASCII s'étend sur 128 valeurs, soit 7 bits, mais est le plus souvent implémenté sur un octet pour préserver l'alignement, le dernier bit pouvant être utilisé par exemple pour ajouter un code correcteur d'erreur – ce qui à l'époque des modems n'était pas superflu. Bref, la pratique courante était alors de manipuler une chaîne de caractères comme un tableau d'octets.

Les encodages ISO-latin

Dans les années 1990, pour satisfaire les besoins des pays européens, ont été définis plusieurs encodages alternatifs, connus sous le nom de ISO- latin (http://en.wikipedia.org/wiki/ISO/IEC_8859) , ou encore ISO-8859 (http://en.wikipedia.org/wiki/ISO/IEC_8859) . Idéalement, on aurait pu et **certainement dû** définir un seul encodage pour représenter tous les nouveaux caractères; mais entre toutes les langues européennes, le nombre de caractères à ajouter était substantiel, et cet encodage unifié aurait largement dépassé 256 caractères différents, il n'aurait donc **pas été possible** de tout faire tenir sur un octet.

Mais on a préféré préserver la "bonne propriété" du modèle *un caractère == un octet*, ceci afin de préserver le code existant qui aurait sinon dû être retouché ou récrit.

Dès lors il n'y avait pas d'autre choix que de définir **plusieurs** encodages distincts; par exemple pour le français on a utilisé à l'époque ISO- latin-1 (http://en.wikipedia.org/wiki/ISO/IEC_8859-1) ; pour le russe ISO- latin-5 (http://en.wikipedia.org/wiki/ISO/IEC_8859-5) .

À ce stade, le ver était dans le fruit. Depuis cette époque pour ouvrir un fichier il faut connaître son encodage.

Unicode

Lorsqu'on a ensuite cherché à manipuler aussi les langues asiatiques, il a de toutes façons fallu définir de nouveaux encodages beaucoup plus larges. C'est ce qui a été fait par le standard Unicode (<http://en.wikipedia.org/wiki/Unicode>) qui définit 3 nouveaux encodages:

- UTF-8 (<http://en.wikipedia.org/wiki/UTF-8>) : un encodage à taille variable, à base d'octets, qui maximise la compatibilité avec ASCII,
- UTF-16 (<http://en.wikipedia.org/wiki/UTF-16>) : un encodage à taille variable, à base de mots de 16 bits
- UTF-32 (<http://en.wikipedia.org/wiki/UTF-32>) : un encodage à taille fixe, à base de mots de 32 bits

Ces 3 standards couvrent le même jeu de caractères (113 021 tout de même dans la dernière version). Parmi ceux-ci le plus utilisé est certainement *utf-8*. Un texte ne contenant que des caractères du code US-ASCII initial peut être lu avec l'encodage UTF-8.

Pour être enfin tout à fait exhaustif, si on sait qu'un fichier est au format Unicode, on peut déterminer quel est l'encodage qu'il utilise, en se basant sur les 4 premiers octets du document; ainsi dans ce cas particulier (lorsqu'on est sûr qu'un document est dans un des formats Unicode) il n'est plus nécessaire de connaître son encodage de manière "externe".

Un caractère ce n'est pas un octet

Avec Unicode, vous voyez qu'on a cassé le modèle *un caractère == un octet*. Avec ISO-latin, l'humanité a gagné du temps pour récrire tous les programmes qui faisaient, de manière erronée, la confusion entre les deux.

Pour revenir à python, lorsqu'il s'agit de manipuler des chaînes de caractères (qui à nouveau, peuvent provenir de n'importe quelle source de données), et pour anticiper un peu

- Dans la version 2, il y a deux types prédéfinis `str` et `unicode` pour implémenter les deux modèles, respectivement
 - `str`: un caractère par octet, qui fonctionne encore avec les langues européennes
 - `unicode`: un type qui permet de manipuler des chaînes de caractères unicode. Je vous renvoie à la page sur `unicode` (<https://docs.python.org/2/howto/unicode.html>) pour plus de détails sur le type `unicode`, que nous **n'approfondissons pas dans le MOOC**.
- Pour information, dans la version python 3, les deux types ont été unifiés, et le nom `str` désigne maintenant un type capable de manipuler des chaînes en Unicode, ce qui semble logique compte tenu de ce qui précède; les encodages à la ISO-latin sont certainement amenés à tomber en désuétude de toutes façons.

Je vous avais bien dit que les accents c'était scabreux !