NSI en première (2019-2020)
Résumé Types simples

1 Représentation numérique de l'information

2 Nombres entiers

3 Booléens

4 Nombres réels

5 Textes

Un ordinateur traite du texte. Un programme en Python peut également traiter du texte. Et il est lui-même écrit sous forme de texte.

Le type **str** sert à représenter des chaînes de caractères en Python. Il y a plusieurs manières de définir une variable de type str, en utilisant des guillemets, des apostrophes, etc. Si on écrit `ch = "bon"+"jour"`, on obtient la chaîne `"bonjour"` que l'on peut afficher dans l'interpréteur. La manière dont sont représentés les différents caractères en machine n'a pas vraiment d'importance. Mais pour lire du texte à partir d'un fichier ou écrire du texte dans un fichier c'est différent.

5.1 Représentation

Une machine ne connaît que les nombres 0 et 1. Mais nous avons vu qu'elle pouvait ainsi utiliser des nombres entiers naturels. Donc pour écrire du texte, une solution est d'associer chaque caractère à un entier. Auparavant, il faut définir précisément l'ensemble des caractères à coder. On parle de "charset", abréviation de l'anglais "character set".

Au début des années 1960, a été inventé le codage américain ASCII permettant de représenter sur 7 bits les caractères d'un clavier anglophone. Le "A" correspond au nombre 65, le "B" au nombre 66, ..., le "a" au nombre 97, le chiffre 0 au nombre 48, etc. On peut visualiser les 128 caractères et les codes correspondants avec les fonctions Python `chr` et `ord`.

```
>>> ord("k")
107
>>> chr(52)
'4'
```

En binaire, le caractère "A" est représenté par 100 0001 (65), le caractère "B" est représenté par 100 0010 (66), le caractère "C" est représenté par 100 0011 (67), etc. Le caractère "a" est représenté par 110 0001 (97), le caractère "b" est représenté par 110 0010 (98), le caractère "c" est représenté par 110 0011 (99), etc. Nous pouvons remarquer qu'entre les caractères en capitales et les caractères en minuscules, il y a un seul bit de différence, le deuxième à partir de la gauche. Les nombres choisis ne le sont donc probablement pas "au hasard" !

Ce système était suffisant pour la langue anglaise mais pas pour toutes les autres. De nombreux systèmes d'encodage pour les autres langues ont donc été créés depuis, par exemple l'ISO 8859-1 (ou latin-1) pour la langue française pour laquelle les accents sont importants. Ce système reprend les 128 codes ASCII (de 0 à 127), et en ajoute 128 (de 128 à 255). Le codage est donc fait sur 8 bits et par exemple le "é " est représenté par le nombre 233. Le problème est que d'une langue à une autre les systèmes ne sont

pas compatibles excepté avec l'ASCII. Le système UNICODE a été créé au début des années 1990 pour avoir une compatibilité mondiale, pour permettre des échanges de textes dans toutes les écritures. Pour chaque langue, un identifiant numérique est affecté à un caractère. Il était prévu au début un codage sur 2 octets, soit 16 bits, donc 65536 valeurs possibles. Cela s'est avéré insuffisant et actuellement, on en est à plus de 130 000 caractères codés. L'UNICODE utilise différents formats comme UTF-8 ou UTF-16. Pour écrire par exemple un fichier texte avec l'extension html qui va produire une page Web, il est conseillé d'utiliser l'encodage UTF-8 qui est totalement compatible avec l'ASCII.

5.2 Gestion des fichiers textes en Python

Ouverture et fermeture d'un fichier

La fonction `open` prend deux paramètres, le nom du fichier et le mode d'ouverture : 'w' pour le mode "écriture", 'r' pour le mode "lecture" et 'a' pour le mode "ajout", (write, read, append).

```
fic = open('fichier','w')
# ou bien
fic = open('fichier','r')
# ou bien
fic = open('fichier','a')
```

Il est essentiel de fermer un fichier qui a été ouvert. L'instruction est : `fic.close()`.

Écriture et lecture

La méthode `write` prend en paramètre une chaîne de caractères (type str), par exemple : `fic.write("J'écrit dans un fichier")`.

Pour écrire sur plusieurs lignes, on utilise le caractère '\n' qui force un retour à la ligne. Par exemple : `fic.write("J'écrit dans un fichier\nA la fin, je le ferme.")`, ou plus lisible `fic.write("J'écrit dans un fichier" + "\n" + "A la fin, je le ferme.")`.

Si le fichier est fermé puis à nouveau ouvert en écriture, c'est un nouveau fichier qui est écrit. Ce qui était écrit auparavant est perdu. Pour écrire à nouveau dans un fichier déjà fermé, on l'ouvre en mode "ajout". Le texte est écrit à la suite du précédent.

```
fic = open('fichier.txt','w')
fic.write("J'écrit dans un fichier\nA la fin, je le ferme.")
fic.write("J'écrit à nouveau.")
fic.close()

fic = open('fichier','a')
fic.write("Je continue.")
fic.close()
```

L'extension ".txt" précise avec quel logiciel le fichier peut être ouvert, ici un éditeur de texte. Par défaut, le fichier `fichier.txt` est créé dans le dossier où est enregistré le fichier Python.

Si les données à écrire sont de type numérique, il faut les convertir au préalable en type str.

```
a, b = 2, 5
fic = open('fichier','w')
fic.write(str(a) + " x " + str(b) + " = " + str(a*b))
fic.close()
```

Lecture

L'ouverture d'un fichier en mode lecture s'effectue avec la méthode `read`.

Plusieurs instructions sont disponibles : `ch=fic.read(n)` lit n caractères, `ch=fic.read()` lit tout le fichier, `ch=fic.readline()` lit la ligne courante et passe à la suivante. Dans les trois cas, la variable `ch` est une chaîne de caractères.

L'instruction `ch=fic.readlines()` lit toutes les lignes. Dans ce cas, la variable `ch` est une liste de chaînes de caractères. Chaque élément est une ligne du fichier.

L'instruction `ch=[x for x in fic]` produit le même résultat.

Deux méthodes sur les chaînes de caractères sont importantes dans le traitement des données lues. Si `ch` est une chaîne de caractères, alors :

`ch.rstrip()` supprime le caractère de fin de ligne (par exemple "\n");

`ch.split(sep)` coupe la chaîne `ch` suivant le délimiteur `sep`, et renvoie une liste de sous-chaînes de `ch`. Avec `ch="un,deux,trois"`, `ch.split(',')` renvoie `['un','deux','trois']`.

Le séparateur par défaut est l'espace.

Si les données à lire sont des nombres qui doivent être utilisés dans des calculs, il faut les convertir en type int ou en type float.

Voici un exemple :

```
fic = open("fichier.dat", "w")
fic.write(str(5) + "\t" + str(8.3) + "\t" + str(1e-4) + "\n")
fic.write(str(8) + "\t" + str(32.7) + "\t" + str(1e2))
fic.close()

fic = open("fichier.dat", "r")
for ligne in fic:
    liste = ligne.rstrip().split("\t")
    a, b, c = [float(x) for x in liste]
    print(a, b, c)

fic.close()
```

Voyons maintenant où interviennent les notions d'encodage.

Voici un code en Python qui permet d'écrire dans deux fichiers `test1.txt` et `test2.txt`. Dans le premier fichier, on écrit "bonjour" suivi d'un retour à la ligne (`'\n'`). Dans le second fichier, on écrit 'é è ê à ù' avec les caractères é, è, ê, à, ù séparés par des tabulations (`'\t'`) et suivis d'un retour à la ligne.

```
f = open('test1.txt', 'w')
f.write('bonjour\n')
f.close()

f = open('test2.txt', 'w')
f.write('é\tè\tê\tà\tù\n')
f.close()
```

Nous ouvrons les deux fichiers avec le logiciel Notepad++, et si nous cliquons sur le menu Encodage, nous constatons que le premier est encodé en UTF-8 et le second en ANSI. ANSI est un nom donné à un encodage Windows (Windows-1252 ou CP1252). La chaîne de caractères qui comporte des caractères accentués a été convertie pour être écrite.

Faisons un test dans l'interpréteur :

```
>>> ch = 'é\tè\tê\tà\tù\n'
>>> ch.encode('cp1252')
b'\xe9\t\xe8\t\xea\t\xe0\t\xf9\n'
>>> ch.encode('utf-8')
b'\xc3\xa9\t\xc3\xa8\t\xc3\xaa\t\xc3\xa0\t\xc3\xb9\n'
```

Écrivons dans un nouveau fichier avec un encodage UTF-8.

```
ch = 'é\tè\tê\tà\tù\n'
ch = ch.encode('utf-8')
f = open('test3.txt', 'wb')
f.write(ch)
f.close()
```

La chaîne `ch` encodée en utf-8 est en fait une suite d'octets ou bytes. Pour écrire ces octets dans un fichier, il est nécessaire d'utiliser un mode d'écriture différent. Avec le mode "w", la fonction `write` prend en paramètre une variable de type str. Nous utilisons donc le mode "wb", pour une écriture en mode binaire. La fonction `write` prend alors en paramètre une suite d'octets.

Nous vérifions alors avec Notepad++ que le fichier est encodé en UTF-8.

En ouvrant les fichiers avec un éditeur hexadécimal comme EditHexa, nous constatons que le contenu du fichier test2.txt est e9 09 e8 09 ea 09 e0 09 f9 0d 0a et le contenu du fichier test3.txt est c3 a9 09 c3 a8 09 c3 aa 09 c3 a0 09 c3 b9 0a

Ceci signifie que le caractère é est encodé en CP1252 avec le nombre e9 en hexadécimal soit 233 en décimal. Il est encodé en UTF-8 avec le nombre c3a9 en hexadécimal, soit 11000011 10101001 en binaire. Sans entrer dans les détails, la norme précise qu'on ne garde que les parties entre parenthèses 110(00011) 10(101001), soit 5+6=11 bits. Ces 11 bits forment le nombre 00011 101001, qui est noté U+00E9. Ce nombre est bien sûr 233 en décimal.

Ouvrons le fichier test2.txt avec Notepad++. L'encodage est en ANSI. Dans la barre des menus, nous cliquons sur Encodage puis Convertir en UTF-8. Nous enregistrons le fichier et l'ouvrons avec EditHexa. Le contenu est identique à celui du fichier test3.txt.

L'encodage est important particulièrement pour les fichiers destinés à être partagés. Une page d'un site Web, écrite en HTML, est vue sur des machines différentes, avec des navigateurs différents. Par exemple, si le fichier HTML est encodé en UTF-8 avec BOM, le caractère é est correctement affiché sur les trois navigateurs Chrome, Firefox ou Microsoft Edge (successeur de Internet Explorer). Si le fichier HTML est encodé en UTF-8 (sans BOM), il est affiché correctement avec le navigateur Chrome et ne l'est pas avec les deux autres navigateurs. Si on précise dans l'entête du fichier HTML l'ensemble des caractères utilisés en ajoutant entre les balises `<head>` et `</head>` la ligne `<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>` alors l'affichage est correct dans les trois navigateurs.

En Python la fonction `ord` renvoie le point de code Unicode d'un caractère. Donc ord('é') renvoie le nombre 233. Notons que les nombres de 0 à 127 inclus sont aussi les points de code du codage ASCII identiques à ceux du codage Unicode (famille UTF) et que les nombres de 0 à 255 inclus sont aussi les points de code du codage ISO-8859-1 ou Latin-1. La fonction `chr` permet d'effectuer des tests. Par exemple, `chr(1604)` est un caractère arabe.

Les fonctions `encode` et `decode` permettent de passer d'un encodage à un autre. On vérifie que pour l'ASCII, il ne faut pas dépasser le code 127.

```
>>> ch="î"
>>> ch.encode()
b'\xc3\xae'
>>> ch.encode("ascii")
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    ch.encode("ascii")
UnicodeEncodeError: 'ascii' codec can't encode character '\xee'
in position 0: ordinal not in range(128)
>>> ch.encode("ansi")
b'\xee'
```

```
>>> ch=b'\xc3\xae'
>>> ch.decode() # par défaut utf-8
'î'
>>> ch=b'\xee'
>>> ch.decode("ansi")
'î'
```

Exercice

On considère le code Python suivant qui crée un fichier HTML pour obtenir une page contenant les caractères éêé :

```
f = open('page.html', 'w')
ch = """<!DOCTYPE html>
<html>
    <head>
    <meta charset="utf-8"/>
    </head>
    <body>
    éêé
    </body>
</html>"""
f.write(ch)
f.close()
```

Exécuter le fichier et ouvrir la page HTML avec différents navigateurs. L'affichage des caractères est-il correct ?

Effectuer les modifications dans le programme pour avoir un affichage correct dans tous les navigateurs.

Penser à l'encodage et au mode d'écriture dans un fichier.

Solution

Les navigateurs affichent des caractères "bizarres".

Il faut encoder la chaîne en UTF-8 avec l'instruction `ch=ch.encode("utf-8")`.

Mais en mode "w" on ne peut pas écrire des octets. Il faut donc passer en mode "wb". On obtient le programme :

```
f = open('page.html', 'wb')
ch = """<!DOCTYPE html>
<html>
    <head>
    <meta charset="utf-8"/>
    </head>
    <body>
    éé é
    </body>
</html>"""
ch = ch.encode("utf-8")
f.write(ch)
f.close()
```