Représentation d'un texte en machine

1 Etude d'un texte

Lire la partie du texte suivant `https://docs.python.org/fr/3.6/howto/unicode.html` **concernant l'histoire du codage des caractères** et répondre aux questions suivantes :

1. En quelle année le code ASCII a été standardisé ?
2. Quels sont les points faibles du code ASCII ?
3. Combien de bits un code ASCII ?
4. Qu'est que l'ISO ?
   Pourquoi est il nécessaire de normaliser le codage des caractères dans l'échange d'informations au niveau mondial ?
5. Quelle plage de valeurs numériques pour A ; ;B et a..b ?
   (Dans une console Python utiliser la fonction `ord()` qui prend en paramètre un caractère et renvoie le nombre décimal correspondant au point de code ASCII.
   Par exemple :

```
>>> ord('A')
65
```

)
6. Ecrire un programme Python affichant les caractères dans l'ordre de leur point de code ASCII, dans un tableau à deux dimensions de 8 lignes et 16 colonnes.
   Vous devez observer les 95 caractères imprimables du code ASCII, visibles sur la page wikipedia concernant le code ASCII.

```
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
```

   Utiliser la fonction `chr()` de Python qui prend en paramètre un point de code ASCII en décimal et renvoie le caractère correspondant

```
>>> chr(65)
'A'
```

7. Au vu du tableau ci-dessus quelle logique a guidé la disposition des chiffres et des lettres de l'alphabet latin (majuscules et minuscules) ?
   Cette logique pousse à utiliser comme point de code ASCII plutôt l'hexadécimal que le décimal ainsi le point de code ASCII du caractère 'A' est 0x41 (cinquième ligne et deuxième colonne) plutôt que 65

La fonction `chr()` prend en compte cette logique :

```
>>> chr(0x41)
'A'

>>> chr(65)
'A'
```

8. Le caractère SP (pour espace) a pour point de code 0x20 a été affiché avant le point d'exclamation mais cela ne se voit pas.
   Combien de caractères n'ont pas été affichés ? A quoi servent-ils ?
9. Le caractère permettant de sauter de ligne \n a pour point de code 0x0a.
   La lettre 'C' a pour point de code 0x43.
   La chaîne 'Ceci est' peut être encodé par :
   '\x43\x65\x63\x69\x20\x65\x73\x74'
   D'ailleurs on peut afficher cette chaîne encodée :

```
>>> print('\x43\x65\x63\x69\x20\x65\x73\x74')
Ceci est
```

   Ecrire une fonction Python `encode(chaine:str)` permettant d'encoder une chaine de caractère passée en paramètre
   Par exemple :

```
>>> encode('Ceci est une ligne\nVoici une autre')
'\x43\x65\x63\x69\x20\x65\x73\x74\x20\x75\x6e\x65\x20
\x6c\x69\x67\x6e\x65\xa\x56\x6f\x69\x63\x69\x20\x75
\x6e\x65\x20\x61\x75\x74\x72\x65'
```

2 Normes ISO 8859

Comme on l'a vu précédemment le code ASCII ne prend pas en compte les lettres accentuées et plus généralement tous les caractères ne faisant pas partie de l'alphabet latin.

L'ISO (Organisation Internationale de Normalisation) a donc proposé la norme ISO 8859 pour remédier à ce problème.

L'encodage d'une caractère se fait sur 8 bits, on peut donc encoder au plus 256 caractères.

D'où la déclinaison de cette norme en plusieurs pages ou tables par exemple la table 8859-1 (dit aussi Latin-1) concerne l'Europe Occidentale.

Par exemple voici un extrait d'une ancienne page html encodée avec la norme Latin-1

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml">
5
```

Il y a eu ainsi 16 tables ISO 8859-n pour n variant de 1 à 16, ce qui a posé un problème d'incompatibilité entre les tables, car ce qui est codé par 0xdf dans la norme ISO 8859-1 est le ß de la langue allemande que l'on trouve dans le mot straße alors que 0xdf dans la norme ISO 8859-5 code le caractère П dans l'alphabet cyrillique

3 Unicode

Avec l'avènement du Web et l'internationalisation des échanges des informations, il était nécessaire de fonder un codage universel de tous les caractères.

Ainsi est apparu à partir de 1991, un organisme international appelé Unicode `https://home.unicode.org/` , qui a mis en place un codage universel de tous les caractères appelé aussi Unicode.

L'idée est d'attribuer à tout caractère un point de code noté U+xxxx où x est un chiffre hexadécimal. En pratique il convient pour certains caractères de rajouter d'autres chiffres hexadécimaux.

Pour des raisons de compatibilité ascendante le code ASCII et la table ISO 8859-1 occupent les 8 premiers bits, ainsi le caractère 'A' a pour point de code Unicode U+0041 (voir ici `https://www.unicode.org/charts/PDF/U0000.pdf`), et le ß de la langue allemande a pour point de code U+00DF (voir ici `https://www.unicode.org/charts/PDF/U0080.pdf`)

Le caractère П de l'alphabet cyrillique a pour point de code U+041F

Va-t-on ensuite juxtaposer les codes Unicode pour coder les chaînes de caractères ?

Encodages

Pour résumer la section précédente : une chaîne Unicode est une séquence de points de code, qui sont des nombres de `0` à `0x10FFFF` (1 114 111 en décimal). Cette séquence de points de code doit être stockée en mémoire sous la forme d'un ensemble de **unités de code**, et les **unités de code** sont ensuite transposées en octets de 8 bits. Les règles de traduction d'une chaîne Unicode en une séquence d'octets sont appelées un **encodage de caractères** ou simplement un **encodage**.

Le premier encodage auquel vous pouvez penser est l'utilisation d'entiers 32 bits comme unité de code, puis l'utilisation de la représentation des entiers 32 bits par le CPU. Dans cette représentation, la chaîne « Python » ressemblerait à ceci :

```
   P           y           t           h           o           n
0x50 00 00 00 79 00 00 00 74 00 00 00 68 00 00 00 6f 00 00 00 6e 00 00 00
   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
```

Cette représentation est simple mais son utilisation pose un certain nombre de problèmes.

1. Elle n'est pas portable ; des processeurs différents ordonnent les octets différemment.
2. Elle gâche beaucoup d'espace. Dans la plupart des textes, la majorité des points de code sont inférieurs à 127, ou à 255, donc beaucoup d'espace est occupé par des octets `0x00`. La chaîne ci-dessus occupe 24 octets, à comparer aux 6 octets nécessaires pour une représentation en ASCII. L'utilisation supplémentaire de RAM n'a pas trop d'importance (les ordinateurs de bureau ont des gigaoctets de RAM et les chaînes ne sont généralement pas si grandes que ça), mais l'accroissement de notre utilisation du disque et de la bande passante réseau par un facteur de 4 est intolérable.
3. Elle n'est pas compatible avec les fonctions C existantes telles que `strlen()`, il faudrait donc utiliser une nouvelle famille de fonctions, celle des chaînes larges (*wide strings*).

4 UTF-8

Pour pallier aux difficultés vues précédemment, il a fallu trouver un système astucieux d'encodage appelé UTF-8 pour *Universal Transformation Format* (8 désigne le nombre minimal de bits pour représenter un point de code)

Table 3-6. UTF-8 Bit Distribution

Scalar Value	First Byte	Second Byte	Third Byte	Fourth Byte
00000000 0xxxxxxx	0xxxxxxx			
00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
000uuuuu zzzzyyyy yyxxxxxx	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

Regardons sur des exemples comment ça marche :

1. Tout caractère de la table ASCII codé sur 7 bits est codé sur un octet ainsi le caractère 'A' dont le point de code Unicode est U+0041 en binaire 00000000 01000001 sera codé par 01000001
2. le ß de la langue allemande a pour point de code U+00DF ce qui en binaire vaut 0000 0000 1101 1111.
   On prend pour modèle la deuxième ligne du tableau précédent où 00000yyy vaut 00000000 et yy xxxxxx vaut 11 011111.
   Donc l'encodage donne pour premier octet 110 000 11 en hexadécimal c3 et pour deuxième octet 10 011111 en hexadécimal 9f.
   On peut vérifier ce résultat à la console de Python

```
>>> s = chr(0xdf)
>>> s.encode()
b'\xc3\x9f'
```

   Ou encore

```
>>> s = '\xdf'
>>> s.encode()
b'\xc3\x9f'
```

   Ou encore

```
>>> s = '\u00df'
>>> s.encode()
b'\xc3\x9f'
```

   Réciproquement

```
>>> b'\xc3\x9f'.decode()
'ß'
```

3. La chaîne 'Straße' est encodée par b'\x53\x74\x72\x61\xc3\x9f\x65'
   Si on décode b'\x53\x74\x72\x61\xc3\x9f\x65' on obtient 'Straße'

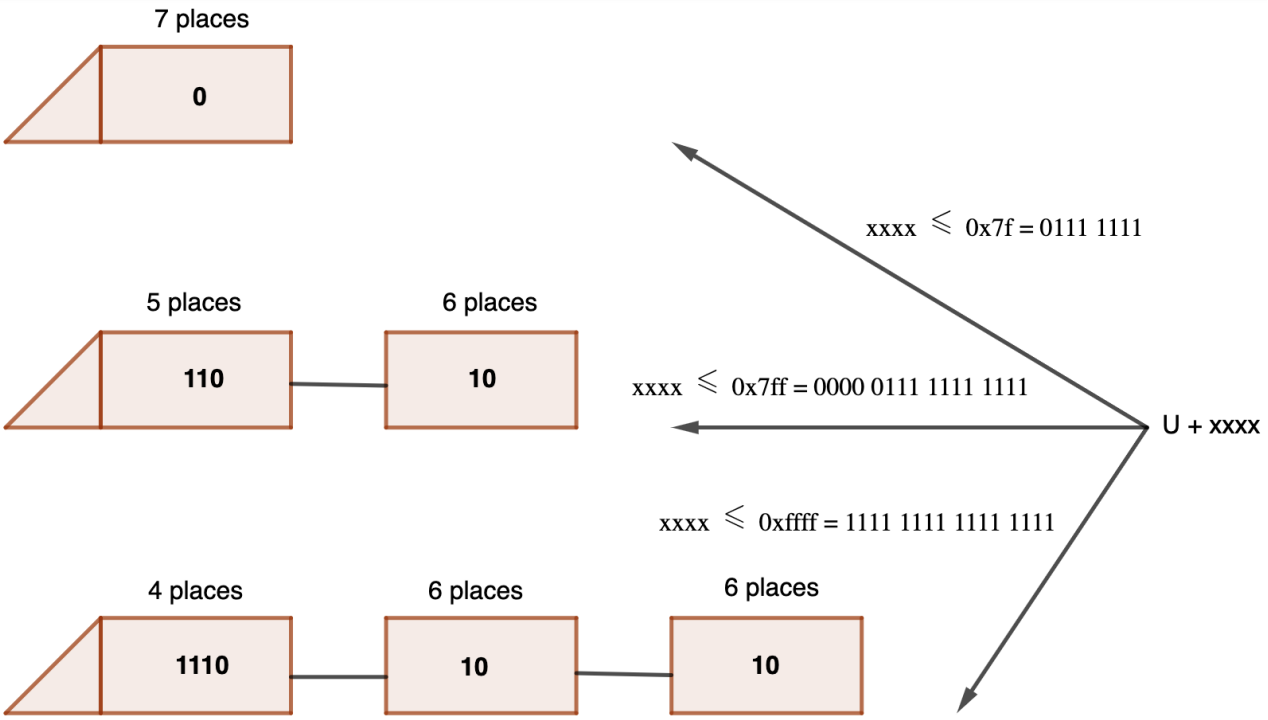
```
>>> b'\x53\x74\x72\x61\xc3\x9f\x65'.decode()
'Straße'
```

5 Algorithme UTF-8

En entrée on a le point de code U+xxxx qui contient au moins deux octets.

Ensuite il faut répartir ces octets dans des "trains" d'octets contenant des **méta-données**

1. Un octet dont le bit de poids fort est 0 est seul
2. Un octet dont les deux bits en tête sont 10 fait partie d'un ensemble d'octets dont la longueur est précisé par le premier octet de cet ensemble
3. 110 signifie qu'il y a deux octets dans l'ensemble, 1110, trois octets etc...



d'où la fonction Python qui prend en paramètre un entier sous forme hexadécimale un point de code Unicode et qui renvoie une chaine de caractères représentant l'encodage en binaire.

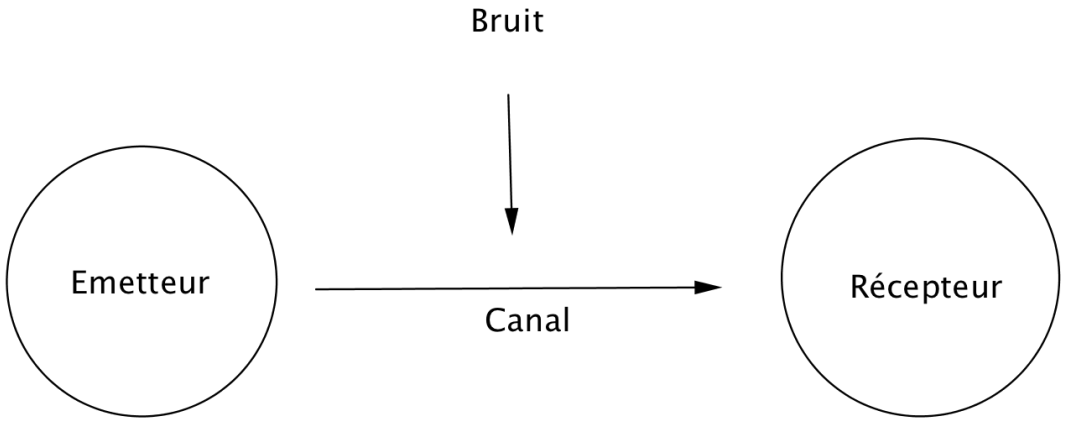
```
def encode_utf8(p_code:int)->str:
    """
    >>> encode_utf8(0x41)
    '01000001'
    >>> encode_utf8(0xdf)
    '11000011 10011111'
    """
    chaine = bin(p_code)[2:]
    if p_code <= 0x7f:
        return '0'*(8-len(chaine))+chaine
    elif p_code <= 0x7ff:
        chaine = '0'*(11-len(chaine)) + chaine
        return '110'+chaine[0:5]+' '+'10'+chaine[5:]
    # à compléter
```

6 Exercices

Ex 1

1. Visionner la vidéo sur Youtube de computerphile sur UTF-8 `https://www.youtube.com/watch?v=MijmeoH9LT4`
2. Quelle est la limite de l'algorithme de l'UTF-8 ?

Ex 2 : bit de parité



"Le premier engin spatial de la série Mariner prend des photographies en noir et blanc de la planète Mars en 1965. Les images ont une résolution de 200 x 200 pixels avec 64 nuances de gris codées sur 6 bits. Les données sont transmises avec un débit binaire de 8 bits par seconde et par conséquent la transmission d'une image prend environ 8 heures.

Plus tard en 1972, Mariner 9 arrive en orbite autour de Mars. Pour permettre une correction des erreurs de transmission, le code Reed-Muller avec des mots de code de 32 bits est utilisé (6 bits pour la nuance de gris du pixel et 26 bits de clé de redondance). Par contre les progrès au niveau des techniques de transmission ont alors permis d'atteindre un débit de 16000 bits par seconde" (Architecture des ordinateurs-Jean-Jacques Schwarz-Eyrolles- 2005)(Voir aussi sur Youtube Mariner4 Mars)

De manière générale, la plupart des messages (texte, son, image) sont **numérisés** et transformés en une suite de bits c'est à dire de 0 et de 1. Au cours de leur transmission de l'émetteur au récepteur avec une certaine probabilité (taux d'erreur) qui dépend du canal de transmission un bit transmis peut être différent du bit émis et ceci n'est pas dû à un acte de malveillance mais à des perturbations physiques dans le canal de transmission, on parle de canal bruité.

Il existe des techniques, pour détecter les erreurs , on parle de codes détecteurs d'erreurs et d'autres pour détecter et corriger les erreurs, on parle de codes correcteurs d'erreurs. Regardons sur un exemple simplifié l'idée de la détection d'erreurs.

On veut envoyer le message "Bonjour".

1. Chaque caractère est encodé sur 8 bits, mais seulement 7 bits portent l'information.

   Si $s_7 s_6 ... s_1 s_0$ avec $s_i \in \{0, 1\}$

   alors le bit de poids fort s_7 est appelé **bit de parité** , dans le but de **détecter** s'il y a une erreur dans la transmission, ainsi :

Si la somme $s_0 + s_1 + ... + s_6$ est paire alors $s_7 = 0$ sinon $s_7 = 1$ (cela revient à compter le nombre de 1)

Par exemple la lettre n est codée par ' \6e' c'est à dire '0110 1110', il y a donc cinq 1 donc le bit de parité sera ici **1** on remplace le bit de poids fort par 1 donc on obtient '1110 1110'

Sachant que l'encodage de 'Bonjour' est

```
\x42\x6f\x6e\x6a\x6f\x75\x72
```

mettre en place les bits de parité

2. Vérifier qu'on peut détecter à coup sûr une erreur sur 1 bit avec le bit de parité
3. Peut on détecter deux erreurs sur 7 bits ?
4. Y-a-t-il conflit avec l'algorithme de l'UTF-8 ?

   Comment procéder ?

Ex 3

Il s'agit d'encoder la chaîne $\Delta = b^2 - 4ac$

1. Chercher ici `https://www.unicode.org/charts` les points de code Unicode des caractères
2. En déduire l'encodage des caractères par UTF-8
3. En déduire l'encodage de la chaîne
4. Vérifier l'encodage avec la fonction `decode()`

```
>>> b'\xce\x94\x20\x3d\x20\x62\xc2\xb2\x20\x2d\x20\x34\x61\x63'.decode()
'Δ = b² - 4ac'
```

Ex 4

Créer un fichier html minimal dans lequel il y a la phrase $\Delta = b^2 - 4ac$ (encadré par des balises p)

Ex 5

1. Trouver les points de code des symboles mathématiques ∞ et $\in$
2. En déduire leur encodage par UTF-8
3. Encoder la chaîne '$\forall n \in \mathbb{N}, n < +\infty$'

Ex 6

Le point de code pour le symbole alchimique du vinaigre est U+1F70B

1. Dessiner ce caractère
2. Encoder ce caractère par UTF-8 et le décoder

Ex 7

Comment sont codés les émoticones utilisées dans les SMS ?

1. Choisir un émoticone au hasard et l'encoder
2. Vérifier

Ex 8

1. Aller voir sur le Web la technique du chiffrement par décalage

2. Ecrire une fonction `chiffrer(chaine,cle)` qui prend en paramètre une chaine de caractères ne contenant que des minuscules de 'a' à 'z' ainsi que des espaces ' ', et un nombre cle compris entre 1 et 25 et qui renvoie la chaine chiffrée (les espaces ne sont pas chiffrés)
3. Ecrire la fonction réciproque `chiffrer(chaine,cle)` qui renvoie la chaine 'en clair'