Lycée Jean Moulin **Draguignan**
NSI 1re Année .....

Les caractères

Représentation des caractères

En informatique, une information est codée exclusivement par des séquences de 0 et 1. On a vu précédemment comment coder des entiers et des réels en binaire mais on a également besoin de trouver un moyen de coder du texte (caractères '0', '1', 'A', 'é' ; ponctuation ' !' , '?' espace, . . . ; symboles spéciaux comme €, $, @) comme une séquence de 0 et 1.

Pour coder/décoder des caractères, il faut se mettre d'accord sur le nombre de bits utilisés pour représenter un caractère (pour pouvoir segmenter une longue séquence de 0 et de 1 en caractères) .
On associe à chaque caractère (une lettre, un signe de ponctuation, une espace. . .) un nombre.
Un texte est alors une suite de ces nombres, on parle de **chaîne de caractères**.

Dans les années 1980-1990, il y avait plusieurs normes pour coder les caractères qui sont apparues à travers le monde : l'ASCII aux État-unis, le KOI8-R en Russie, . . .
Ces normes proposaient d'encoder les caractères sur 7 ou 8 bits mais le fait que les normes soient nées indépendemment les unes des autres rendait les systèmes difficilement inter-opérables et très spécifiques (en Russe, il faut pouvoir représenter l'alphabet cyrillique).

Par exemple, **l'ASCII (American Standard Code for Information Interchange)** développé aux État-Unis, utilise 7 bits (128 valeurs possibles) pour coder des caractères dont les codes apparaissent sur la table suivante .

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0010	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

On lit dans cette table que la représentation binaire du caractère 'a' est '0110 0001' (7e ligne, 2e colonne).
Ce qui correspond à l'écriture en binaire du nombre 97
On dit que le code ASCII de 'a' est 97
Et celle du 'z' est 01111010 soit 122
Le code ASCII de 'z' est 122

? Exercice 1 :

1. Déterminer les entiers qui permettent de coder les lettres 'A' et 'Z'
2. La différence entre les minuscules et majuscules est elle constante ?

? Exercice 2 :

Donner la représentation en binaire du mot 'NSI'

• Les 33 caractères de contrôle (les deux premières lignes du tableau et le caractère DEL) correspondent aux significations suivantes :

NUL	Null (nul)
SOH	Start of Header (début d'en-tête)
STX	Start of Text (début du texte)
ETX	End of Text (fin du texte)
EOT	End of Transmission (fin de transmission)
ENQ	Enquiry (End of Line) (demande, fin de ligne)
ACK	Acknowledge (accuse de réception)
BEL	Bell (caractère d'appel)
BS	Backspace (espacement arrière)
HT	Horizontal Tab (tabulation horizontale)
LF	Line Feed (saut de ligne)
VT	Vertical Tab (tabulation verticale)
FF	Form Feed (saut de page)
CR	Carriage Return (retour chariot)
SO	Shift Out (fin d'extension)
SI	Shift In (démarrage d'extension)

DLE	Data Link Escape
DC1-DC4	Device Control (contrôle de périphérique)
NAK	Negative Acknowledge (accuse de réception négatif)
SYN	Synchronous Idle
ETB	End of Transmission Block (fin du bloc de transmission)
CAN	Cancel (annulation)
EM	End of Medium (fin de support)
SUB	Substitute (substitution)
ESC	Escape (echappement)
FS	File Separator (séparateur de fichier)
GS	Group Separator (séparateur de groupe)
RS	Record Separator (séparateur d'enregistrement)
US	Unit Separator (séparateur d'unité)
SP	Space (espace)

DEL	Delete (effacement)

• Les chiffres sont rangés dans l'ordre croissant (codes 48 à 57), et les 4 premiers bits définissent la valeur en binaire du chiffre.

Fonctions prédéfinies en Python

Deux fonctions prédéfinies permettent d'effectuer la conversion entre un caractère et son identifiant numérique correspondant. La fonction ord donne l'identifiant numérique correspondant à un caractère tandis que la fonction chr fait l'opération inverse.

```
>>> ord('a')
97
>>> chr(97)
'a'
```

Exercice 3 :

1. Tester les fonctions suivantes :

```
def f1(lettre):
    return lettre,ord(lettre),bin(ord(lettre))
```

```
def f2(nombre):
    return chr(nombre),nombre,bin(nombre)
```

2. Écrire un programme qui affiche pour toutes les lettres de l'alphabet un résultat du type:
('a', 97, '0b1100001')
('b', 98, '0b1100010')
...

Et les autres caractères ?

La nécessité de représenter des textes comportant des caractères non présents dans la table ASCII tels ceux de l'alphabet latin utilisés en français comme le 'à', le 'é' ou le 'ç' impose l'utilisation d'un autre codage que l'ASCII.

Plusieurs propositions de codage coexistent.
Afin de faciliter les choses, ces propositions sont des extensions du codage ASCII :
- le codage des caractères présents dans la table ASCII est conservé ;
- le principe du codage de chacun des caractères sur un octet est conservé.

Mais les 8 bits de l'octet vont être utilisés. Cela permet de coder $2^8 = 256$ caractères, soit 128 caractères supplémentaires.

L'ISO, organisation internationale de normalisation, propose de son côté plusieurs variantes de codages adaptées aux différentes langues (ISO-8859-x, x variant de 1 à 16).

La plus utilisée concerne les langues européennes occidentales. Il s'agit de l'ISO-8859-1, aussi nommé ISO-Latin1.

Microsoft propose le codage dit Windows-1252 (encore appelé ANSI, bien que cela puisse paraître abusif, l'ANSI, American National Standards Institute, n'ayant jamais validé cette table !).
Ce codage ne diffère de l'ISO-8859-1 que pour quelques caractères tels le signe euro, €, la ligature o-e, œ, ou certains guillemets qui utilisent des codes réservés par ISO-Latin-1 pour des caractères de contrôle.

Et les autres langues ?

À l'évidence, 256 caractères ne suffisant pas pour représenter les lettres de tous les alphabets utilisés (pensons au russe, à l'hébreu, etc.), un nouveau standard a été introduit : Unicode.
La table Unicode comporte la définition de près de cent cinquante mille caractères.
Le codage de cette table est multiple.
Le codage le plus couramment utilisé se nomme UTF-8. Son principe est le suivant : une première série de caractères sont codés sur un octet. D'autres caractères sont codés sur deux octets.

De même des codages sur 3 ou 4 octets sont utilisés pour d'autres caractères. (Cette rapide introduction à UTF-8 est volontairement simplifiée.)
Les 128 premiers caractères de la table UTF-8 sont compatibles avec le codage ASCII. Ainsi le codage UTF-8 d'un texte ne comportant que des caractères présents dans la table ASCII sera le même que le codage ASCII de ce texte.

? Exercice 4 :

1. Rechercher avec un programme les codes numériques des lettres é, è, ê, à, â, ï, ç et ù.
2. Même question pour les majuscules accentuées.
3. Écrire une fonction qui prend en paramètres deux entiers et qui affiche toutes les lettres dont les codes numériques sont compris entre ces deux entiers
4. Sauriez vous décoder cette exclamation en Binaire : 01000010 01110010 01100001 01110110 01101111 00100001
5. Peut-on coder cette phrase avec la table ASCII : "un âne est passé par là" (justifier la réponse)

L'importance de l'encodage

L'encodage en UTF-8 se fait sur plusieurs octets (jusqu'à 4), tandis qu'en IS0-latin-1 il se fait sur un octet.
La lettre 'é' se code sur deux octets en UTF-8 par : 11000011 10101001
Elle sera décodée en ISO-Latin-1 par deux lettres : Ã©
Comme le montre cette fonction (à tester)

```
def f(lettre):
    a=lettre.encode('utf8')
    b=a.decode('latin_1')
    return b
```

Faire la différence entre une valeur et son représentant

? Exercice 5 :

Étant donné quatre octets qui contiennent les valeurs binaires 00101000, 10011100, 01000011 et 10000000. Que représentent-ils si :

1. ce sont quatre nombres non signés (sur 8 bits)?
2. ce sont deux nombres non signés sur 16 bits ?
3. ce sont quatre nombres en complément à deux (sur 8 bits)?
4. ils sont sensés coder quatre caractères ASCII ?, ISO-8859-1 ?, UTF ?