Chapitre 1

Codage numérique

1.1 Les bases 2, 8, 10 et 16

1.1.1 En mathématiques

Base b

Le système de base b utilise les b chiffres : $0, 1, \ldots, n-1$
Ub nombre $a_n a_{n-1} \ldots a_2 a_1 a_0$ écrit en base b se note $(a_n a_{n-1} \ldots a_2 a_1 a_0)_b$
et on a $(a_n a_{n-1} \ldots a_2 a_1 a_0)_b = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_2 b^2 + a_1 b^1 + a_0 b^0$

Base 10

Le système décimal (ou à base 10) utilise les 10 chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
$3247 = 3 \times 1000 + 2 \times 100 + 4 \times 10 + 7 \times 1 = 3 \times 10^3 + 2 \times 10^2 + 4 \times 10^1 + 7 \times 10^0$
($a^0 = 1$ si a n'est pas nul)

Base 2

Le système de base 2 utilise les 2 chiffres : 0 et 1.
$(1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9 + 4 + 1 = 14 = (14)_{10}$.

Base 8

Le système de base 8 utilise les 8 chiffres : 0, 1, 2, 3, 4, 5, 6, 7.
$(751)_8 = 78^2 + 58^1 + 18^0 = 8 \times 64 + 5 \times 8 + 1 = 553 = (553)_{10}$.

Base 16

Le système de base 16 utilise les 10 chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
$(FA2)_2 = 15 \times (16)^2 + 10 \times 16^1 + 2 \times 2^0 = 15 \times 16 \times 16 + 10 \times 16 + 2 = 4002 = (4002)_{10}$.

Passage de la base 10 vers les bases 2, 8, 16 ou autre.

On utilise la division euclidienne.
Par exemple 1000 en base 8.
$1000 = 125 \times 8 + 0$
$125 = 15 \times 8 + 5$
$15 = 1 \times 8 + 7$
Donc $1000 = ((1 \times 8 + 7) \times 8 + 5) \times 8 + 0 = (1 \times 8^2 + 7 \times 8 + 5) \times 8 + 0 = 1 \times 8^3 + 7 \times 8^2 + 5 \times 8 + 0 \times 8^0 = (1750)_8$.
Convertir 1000 en base 2 puis en base 16.

1.1.2 En python

Pour écrire un nombre en base 2, 8 ou 10 on ajoute un préfixe au nombre *(tableau ci-contre)*.

	binaire	octal	hexadécimal
préfixe	**0b**	**0o**	**0x**
exemple	0b110	0o10	0xa ou 0xA

Voici quelques fonctions python pour faire des convertions

- ➔ ***bin()*** pour convertir un nombre en binaire.
- ➔ ***oct()*** pour convertir un nombre en octal.
- ➔ ***hex()*** pour convertir un nombre en hexadécimal.
- ➔ ***ord()*** pour obtenir le code ascii d'un caractère.
- ➔ ***char()*** pour obtenir un caractère à partir de son code ascii.

Tester ceci avec un shell python comme ipython3

```
In [1]: 0b10
In [2]: 0o10
In [3]: 0x10
In [4]: bin(10)
In [5]: oct(10)
```

```
In [6]: hex(10)
In [7]: L="Bonjour"
In [8]: [i for i in L]
In [9]: [ord(i) for i in L]
```

1.2 Le codage numérique du texte

1.2.1 Da ASCII code !

Au commencement, chaque caractère était identifié par un code unique qui est un entier naturel et la correspondance entre le caractère et son code était appelée un ***Charset***. Le code n'étant pas utilisable tel quel par un ordinateur qui ne comprend que le binaire, il fallut donc représenter les codes par des octets, et cela fut appelé ***Encoding***.

Dans de nombreux grimoires anciens on découvre le code ASCII qui était utilisé pour représenter du texte en informatique. ASCII signifiait American Standard Code for Information Interchange. Il paraît que ce code est toujours en usage...

Le code ASCII se base sur un tableau contenant les caractères les plus utilisés en langue anglaise : les lettres de l'alphabet en majuscule (de A à Z) et en minuscule (de a à z), les dix chiffres arabes (de 0 à 9), des signes de ponctuation (point, virgule, point-virgule, deux points, points d'exclamation et d'interrogation, apostrophe ou quote, guillemet ou double quotes, parenthèses, crochets etc.), quelques symboles et certains caractères spéciaux invisibles (espace, retour-chariot, tabulation, retour-arrière, etc.).

Les créateurs de ce code limitèrent le nombre de ses caractères à 128, c'est-à-dire 2 , pour qu'ils puissent être codés avec seulement 7 bits [1] : les ordinateurs utilisaient des cases mémoires de un octet, mais ils réservaient toujours le 8e bit pour le contrôle de parité (c'est une sécurité pour éviter les erreurs, qui étaient très fréquentes dans les premières mémoires électroniques).

Exemple : Le caractère A est codé en ASCII par le nombre 65 (dans notre système décimal habituel), qui correspond en binaire au nombre 1000001.

Devinette : Quel est le code (en décimal et en binaire) du caractère '1' ?, du caractère '*' ? Chaque caractère d'un texte codé en ASCII occupe ainsi un octet.

Un texte de 5000 caractères occupe donc 5 kibioctets [2].

1. Les caractères sur 7 bits sont donc numérotés entre 0 et 127 (7F en hexadécimal, 0111 1111 en binaire).
2. Rappel : 1 Kio c'est 1024 octets.

1.2.2 Activité Taille d'un texte

Ouvrez une console et tapez

```
echo "a" > test    # pour écrire un fichier test avec écrit a
stat -c "%s" test  # pour lire la taille du fichier
```

Remarque: On peut aussi vérifier la taille en octets du fichier obtenu en cliquant d'abord avec le bouton droit sur l'icône du fichier puis sur « Propriétés ».

Le fichier fait il 1 octet ? Faites des tests comme ci-dessus pour vérifier le poids d'un texte avec "n" caractères ascii.

Éditer alors ce fichier avec ghex qui est un éditeur hexadécimal.

```
ghex test
```

On en profitera pour lire les caractères en hexadécimal, octal ou binaire

Combien voyez vous de caractères ? Pourquoi ?

Quelle est la taille (en octets) de la phrase : « Enfin ! Je viens de comprendre. » (attention, il faut compter les espaces, et signes de ponctuation ...) ?

Vérifiez en tapant cette phrase comme précédemment.

As t on le même résultat en utilisant avec un éditeur de texte comme geany, gvim ou encore nano ?

On peut ensuite écrire la même chose dans un logiciel de traitement de texte LibreOffice et se rendre compte que la taille du fichier obtenu n'est pas du tout la même. Quelle peut en être l'explication ?

- il y a des informations (renseignements) en plus (métadonnées)
- il se souvient de la position du curseur, cette information doit donc être rangée quelque part
- le texte n'est pas sauvegardé tel quel mais compressé
- il y a aussi la mise en page (police de caractères, couleur, etc.).

1.2.3 Activité : Utilisation de la table ASCII

1. À l'aide de la table ASCII, coder en binaire la phrase suivante : « L'an qui vient ! ».
2. Voici maintenant une exclamation codée en binaire :
   01000010 01110010 01100001 01110110 01101111 00101100
   Retrouver cette exclamation !
3. 3) Peut-on coder en binaire la phrase « Un âne est-il passé par là ? » à l'aide de la table ASCII ? (Justifier la réponse)

1.3 Quand la table ASCII ne suffit plus

Il va donc falloir étendre la table ASCII pour pouvoir coder les nouveaux caractères. Les mémoires devenant plus fiables et, de nouvelles méthodes plus sûres que le contrôle de parité ayant été inventées, le 8 ième bit a pu être utilisé pour coder plus de caractères.

1.3.1 Micro-activité

Combien le fait d'avoir 8 bits amène t il de nouvelles possibilités ?

On élimine ainsi l'inconvénient très gênant de ne coder que les lettres non accentuées, ce qui peut suffire en anglais, mais pas dans les autres langues (comme le français et l'espagnol par exemple). On a pu aussi rajouter des caractères typographiques utiles comme des tirets de diverses tailles et sortes.

Par exemple, en français les caractères é, è, ç, à, ù, ô, æ, œ, sont fréquemment utilisés alors qu'ils ne figurent pas dans la table ASCII.

1.4 De la difficulté de convenir d'une norme

Le fait d'utiliser un bit supplémentaire a bien entendu ouvert des possibilités mais malheureusement tous les caractères ne pouvaient être pris en charge. La norme ISO 8859 16 appelée aussi **Latin-1** ou Europe occidentale est la première partie d'une norme plus complète appelée **ISO 8859** (qui comprend 16 parties) et qui permet de coder tous les caractères des langues européennes. Cette norme ISO 8859 1 permet de coder 191 caractères de l'alphabet latin qui avaient à l'époque été jugés essentiels dans l'écriture, mais omet quelques caractères fort utiles (ainsi, la ligature œn'y figure pas).

Dans les pays occidentaux, cette norme est utilisée par de nombreux systèmes d'exploitation, dont Linux et Windows. Elle a donné lieu à quelques extensions et adaptations, dont Windows-12527 (appelée ANSI) et ISO 8859-158 (qui prend en compte le symbole créé après la norme ISO 8859-1). C'est source de grande confusion pour les développeurs de programmes informatiques car un même caractère peut être codé différemment suivant la norme utilisée.

Voici deux tableaux présentant côte à côte ces deux encodages :

ISO/CEI 8859-15

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	*non utilisé*															
1x																
2x		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8x	*non utilisé*															
9x																
Ax		¡	¢	£	€	¥	Š	§	š	©	ª	«	¬		®	¯
Bx	°	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Windows-1252 (CP1252)

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
9x		‘	’	“	”	•	–	—	˜	™	š	›	œ		ž	Ÿ
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

`http://fr.wikipedia.org/wiki/ISO_8859-15`

À première vue, on a les mêmes caractères aux mêmes places ; un regard plus attentif montre l'affectation de caractères supplémentaires sur des zones inutilisées dans ISO 8859-15. Un examen encore plus attentif montre enfin que les deux tables sont tout à fait incompatibles.

1.4.1 Activité Utilisation d'un logiciel

On commence parce se connecter au site suivant :

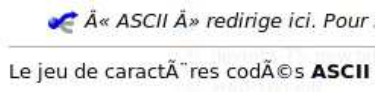
`https://wiki.inria.fr/sciencinfolycee/Convertisseur_texte/binaire/hexa_en_ligne`

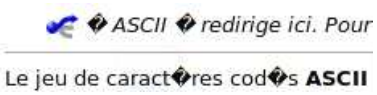
Voici le code binaire d'un texte :

01000010 01110010 01100001 01110110 01101111 00101100 00100000 01110100 01110101 00100000 01100001 01110011 00100000 01110000 01110010 01100101 01110011 01110001 01110101 01100101 00100000 01110100 01101111 01110101 01110100 00100000 01110100 01110010 01101111 01110101 01110110 11101001 00101110 00101110 00101110

1. À l'aide du logiciel fourni sur le site, retrouver le texte contenu dans le code.
2. Ce logiciel est-il compatible avec la norme ISO 8859-1, ISO 8859-15 ou Windows 1252 ? (justifier la réponse)
3. Trouver une astuce pour savoir laquelle des trois est utilisée !

Nous avons tous un jour reçu un courriel bizarre ou lu une page web telle que celle-ci :





Bien que ceci soit de moins en moins fréquent (nous comprendrons bientôt pourquoi), on trouve parfois des phrases dans lesquelles certains caractères sont remplacés par d'autres qui n'ont rien à voir et qui empêchent la lecture et la compréhension du texte. Il s'agit ici d'un problème d'encodage et de décodage [3]. La personne qui écrit le texte utilise une norme différente de celle utilisée par celui qui le lit ! Lorsque c'est un courriel on a la plupart du temps affaire à un spam venant de l'étranger, ce n'est pas sans raison...

1.5 Et l'Unicode vint...

La globalisation des échanges culturels et économiques a mis l'accent sur le fait que les langues européennes coexistent avec de nombreuses autres langues aux alphabets spécifiques voire sans alphabet. La généralisation de l'utilisation d'Internet dans le monde a ainsi nécessité une prise en compte d'un nombre beaucoup plus important de caractères (à titre d'exemple, le mandarin possède plus de 5000 caractères !). Une autre motivation pour cette évolution résidait dans les possibles confusions dues au trop faible nombre de caractères pris en compte ; ainsi, les symboles monétaires des différents pays n'étaient pas tous représentés dans le système ISO 8859-1, de sorte que les ordres de paiement internationaux transmis par courrier électronique risquaient d'être mal compris. La norme Unicode a donc été créée pour permettre le codage de textes écrits quel que soit le système d'écriture utilisé. On attribue à chaque caractère un nom, une position normative et un bref descriptif qui seront les mêmes quelle que soit la plate-forme informatique ou le logiciel utilisés. Un consortium composé d'informaticiens, de chercheurs, de linguistes et de personnalités représentant les États ainsi que les entreprises s'occupe donc d'unifier toutes les pratiques en un seul et même système : l'Unicode.

L'Unicode [4] est une table de correspondance Caractère-Code (Charset), et l'UTF-8 est l'encodage correspondant (Encoding) le plus répandu11. Maintenant, par défaut, les navigateurs Internet utilisent le codage UTF-8 et les concepteurs de sites pensent de plus en plus à créer leurs pages web en prenant en compte cette même norme ; c'est pourquoi il y a de moins en moins de problèmes de compatibilité.

3. Une analyse un peu plus fine de ce texte est intéressante : nous observons que les caractères accentués ou typographiques sont codés sur deux caractères, ce qui suggère un texte d'origine rédigé en UTF-8 et relu (à tort) en tant que texte codé en ISO-8859-1. L'erreur contraire aurait produit des caractères illisibles sur une position (et pas sur deux). On évite ce genre de confusions en remplissant correctement les balises <META> en HTML.
4. Voir : `http://fr.wikipedia.org/wiki/Unicode`

1.5.1 Activité Codage et Internet

Ouvrez un navigateur Internet. Dans la barre d'outils du premier on peut voir à « Affichage », « Encodage des caractères » que c'est l'UTF-8 qui est sélectionné par défaut. Changeons cela et sélectionnons Europe Occidentale (Windows). Les petits caractères désagréables apparaissent. Que s'est-il passé ? En allant dans « Outils », « Informations sur la page », on voit que cette page est encodée en UTF-8. Lorsque le lecteur est lui aussi en UTF-8 tout va bien. Dès qu'on change le paramètre du lecteur (ici, le navigateur), des incompatibilités apparaissent.

En utilisant le navigateur web, et en allant dans « Affichage », « Source », on obtient ceci :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<head>
<title>Phare</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
Prenons l'exemple typique de la lumière émise par un phare maritime :
elle est d'abord indivisible, son coût de production étant alors indépendant du
   nombre d'utilisateurs ;

elle possède une propriété de non-rivalité (elle ne se détruit pas dans l'usage et
   peut donc être adoptée par un nombre illimité d'utilisateurs); elle est é
   galement non excluable car il est impossible d'exclure de l'usage un utilisateur
   , même si ce dernier ne contribue pas à son financement.
</body>
```

On peut lire l'entête de la page html visitée. Où se situe l'information relative à l'encodage ?

On peut aussi dans « Affichage », « Codage », sélectionner Grec (ISO) et se rendre compte en lisant le texte, que le « à » a été remplacé par un « L » à l'envers dit Gamma.

1.5.2 Utiliser recode

Il reste à se donner les bons outils. Le plus simple est d'installer recode et de jouer un peu avec ! On constate vite que certaines conversions ne fonctionnent pas bien ; en effet, il existe souvent des caractères sans équivalent quand on change d'encodage.

1.6 Quelques précisions sur l'UTF-8

1.6.1 Des règles, encore des règles

L'encodage UTF-8 utilise 1, 2, 3 ou 4 octets en respectant certaines règles :

- Un texte en ASCII de base (appelé aussi US-ASCII) est codé de manière identique en UTF-8. On utilise un octet commençant par un bit 0 à gauche (bit de poids fort).

Caractère	Point de code hexadécimal	Valeur scalaire		Codage UTF-8
		décimal	binaire	binaire
A	U+0041	65	1000001	01000001

- Les octets ne sont pas remplis entièrement. Les bits de poids fort du premier octet forment une suite de 1 indiquant le nombre d'octets utilisés pour coder le caractère. Les octets suivants commencent tous par le bloc binaire 10.

Définition du nombre d'octet utilsés

Représentation binaire UTF-8	Signification
0xxxxxxx	1 octet codant 1 à 7 bits
110xxxxx 10xxxxxx	2 octets codant 8 à 11 bits
1110xxxx 10xxxxxx 10xxxxxx	3 octets codant 12 à 16 bits
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	4 octets codant 17 à 21 bits

- Dans la norme ISO 8859-1 le « é » est codé 11101001, en UTF-8 on le code sur deux octets en respectant les précisions apportées dans le tableau ci-dessus. Les bits imposés sont en gras, le code du « é » est écrit en commençant par la droite et l'octet de gauche est rempli par des zéros (en italique). Voilà ce que l'on obtient : 110**00011** 10**101001**. On pourra remarquer que le codage ISO s'inscrit bien dans le codage UTF-8.

1.6.2 Activité Coder en UTF-8

```
In [1]: ord('à')
Out[1]: 224
In [2]: bin(224)
Out[2]: '0b11100000'
In [3]: ord('e')
Out[3]: 8364
In [4]: bin(8364)
Out[4]: '0b10000010101100'
```

ainsi pour coder "à" ; 11100000 (8 bits) s'écrit 11 100000 donc **100**00011 **10**100000

Il faut donc 2 octets pour coder "à".

Pour coder : 10000010101100 s'écrit 10 000010 101100 donc 11100010 10000010 10101100.

Il faut donc 3 octets pour coder "€".

Comment coder les nombres suivants :

1. è
2. É (pour obtenir É il faut se mettre en majuscule puis é.)
3. le code ascii de œ est 339.

1.6.3 Quelques remarques

- Ce codage permet de coder tous les caractères de la norme Unicode.
- Les caractères Unicode sont la plupart du temps représentés en hexadécimal. C'est un moyen de simplifier l'écriture en binaire qui devient lourde lorsqu'on manipule plusieurs octets. Il s'agit de la base 16 ce qui signifie que l'on a besoin de 16 symboles : 0 1 2 3 4 5 6 7 8 9 A B C D E F.

A représentant le nombre dix (10 10), B représentant le nombre onze (11 10) et ainsi de suite jusqu'à quinze. Pour passer du binaire à l'hexadécimal, rien de plus simple, on fait des paquets de quatre bits 12 que l'on représente par un des 16 symboles.

Par exemple : 1110 1001 qui est le code binaire du « é » en ISO 8859-1 devient E916 (l'indice signifie que l'on est en base 1613) ce que l'on peut retrouver dans le tableau donné plus haut. Il

faut noter que la notation en binaire est très peu utilisée sauf par les électroniciens et par ceux qui travaillent en langage machine.

- Le système de codage UTF-8 permet d'encoder un même caractère de plusieurs manières. Ceci peut poser un problème de sécurité car un programme détectant certaines chaînes de caractères (pour contrer des injections dans les bases de données par exemple), s'il est mal écrit, pourrait alors accepter des séquences nuisibles. En 2001 un virus a ainsi attaqué des serveurs http du web.

  Par exemple, le symbole € pourrait être codé sur 4 octets (forme super longue) de la manière suivante :

  11110000 **10**000010 **10**000010 **10**101100. Si elle n'est pas rejetée ou remise sous forme standard ce codage ouvrira une brèche potentielle de sécurité par laquelle on pourra faire passer un virus.

  Question-défi : quel est le nombre maximal de caractères que l'on peut encoder grâce à l'UTF-8 lorsqu'on utilise les quatre octets ?

1.7 Outils et références

- Informations générales sur Unicode :
  - `http://www.unicode.org`
  - `http://fr.wikipedia.org/wiki/Unicode`
  - `http://fr.wikipedia.org/wiki/UTF-8`
- Un éditeur de texte permettant de choisir l'encodage des caractères (Notepad++ sous Windows, TextEdit sous OSX ou encore Geany ou gvim sous Linux).
- Un logiciel convertisseur entre caractères et représentation binaire :
  `https://wiki.inria.fr/sciencinfolycee/Convertisseur_texte/binaire/hexa_en_ligne`
- Une table de caractères : Sous Windows, l'utilitaire qui montre les tables de caractères s'appelle simplement « charmap ». Sous Linux on pourra utiliser « gucharmap », et pour OSX (Apple) il faut chercher la « palette de caractères ».
- Un utilitaire pour le (re)codage des textes :
  `http://recode.progiciels-bpi.ca/index.html`
  Recode est toujours installé par défaut dans Linux.
- Enfin, un site web très utile à consulter en complément :
  `http://www.arcanapercipio.com/lessons/codage_binaire_du_texte/codage_binaire_du_tex`