L'encodage texte permet de stocker des caractères dans un ordinateur qui ne peut stocker que des 1 et des 0. Nous découvrons ici les différents encodages texte et les raison pour lesquelles il en existe plusieurs.

A l'aide du document donné en annexe, répondre aux questions suivantes.

1. Quel est le principe de base de l'encodage texte ?

2. Qu'est-ce qu'un jeu de caractères ? Et l'encodage ?

3. Par quoi est limité le nombre de caractères que comprend un jeu de caractères ?

4. Quel est l'intérêt de l'hexadécimal en informatique ? Comment l'ordinateur va-t-il lire l'hexadécimal ?

5. Décoder la phrase ASCII ci-dessous :

4A 27 61 69 20 63 6F 6D 70 72 69 73 20 21 00

A quoi sert le caractère codé par 00 ?

6. Vérifier que le code ASCII du caractère *Z*, 5A, correspond bien à 90 en décimal.

7. Quel est l'inconvénient de l'ASCII ?

8. En quoi l'ISO 8859 est-il une amélioration sur l'ASCII ?

9. Quel est l'intérêt d'utiliser les normes ISO 10646 et Unicode ?

Pour aller un peu plus loin…

Le texte ci-dessous est codé en ISO 8859-1, mais traduit en décimale. Décodez-le !

77 111 110 32 118 189 117 32 112 111 117 114 78 111 235 108 32 101 115 116 32 100 39 234 116 114 101 32 117 110 32
99 111 100 101 117 114 32 103 233 110 105 97 108 15 35 84 104 101 66 101 115 116 0

Ce qu'il faut retenir

❖ Utilisation de l'hexadécimal

L'hexadécimal permet de représenter un nombre sur 8 bits (donc 8 symboles binaires) avec juste 2 symboles hexadécimaux – c'est une écriture plus concise pour l'utilisateur.
Il faut 4 bits pour représenter un symbole en hexadécimal. 8 bits = 2×4 bits = 2 symboles hexadécimaux.

❖ L'encodage texte

Le principe de base de l'encodage texte est d'associer à chaque caractère un nombre, codé en hexadécimal. Le tableau associant à chaque caractère un nombre est appelé jeu de caractères. L'encodage est la méthode utiliser pour passer du nombre au caractère, et vice-versa.
Trois types d'encodages sont à retenir :

- ASCII : Premier encodage, contenant les caractères de base (mais sans caractères spéciaux).
- ISO 8859 : Encodage incluant les caractères ASCII et les caractères spéciaux spécifique à une langue.
  Exemple : les caractères de type Latin correspondent à ISO 8859-1.
- Unicode (= UTF-8) : Jeu de caractères et encodage universel, comprenant tous les caractères nécessaires à toutes les langues.

Tutoriel : Comprendre les encodages

Extrait du site internet http://sdz.tdct.org/sdz/comprendre-les-encodages.html

Comprendre les encodages

Aux détours de vos aventures informatiques, vous entendez parler de *charset*, d'*encodage*, d'*ASCII*, d'*UTF-8*, d'*ISO-8859*, de *latin-1*… et vous demandez ce que sont ces drôles de bestioles ?
Votre programme ou site web est défiguré par les lettres accentuées et « caractères spéciaux » qui vous rendent de splendides Ã© ou ? ?
Webmestre, vous recopiez bêtement au début de vos toutes pages HTML la ligne

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
```

sans y comprendre que pouic ?
Ce cours vous est destiné. Tout y sera expliqué depuis le début et après l'avoir lu, promis, vous ne serez plus jamais embêtés par ce genre de désagréments.

Un peu de théorie : le texte en informatique

Autant vous le dire tout de suite : tout ce qui sera dit dans ce cours tournera autour d'une même problématique : la façon dont on stocke du texte dans un ordinateur.

Comme vous le savez peut-être, un ordinateur ne peut stocker que des nombres, ou plus précisément des 0 et des 1 (des « *bits* ») qu'on regroupe pour former des nombres en binaire. Comment fait-on alors pour écrire du texte ? La réponse est toute bête : 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**.

Par exemple, on peut décider ceci :

Caractère	**A**	**B**	**C**	…	**Z**	**0**	**1**	**2**	…	**9**	**.**	**,**	**:**	**?**	**!**	*espace*
Nombre associé	0	1	2	…	25	26	27	28	…	35	36	37	38	39	40	41

Avec cet exemple fictif, le texte « SALUT LES ZOZOS ! » donnerait ceci en mémoire :

```
18 00 11 20 19 41 11 04 18 41 25 14 25 14 18 41 40
```

Comme vous le voyez, on a donné un nombre unique pour chaque lettre de l'alphabet (de A à Z), pour les dix chiffres (de 0 à 9) et pour les signes de ponctuation, sans oublier l'espace.

Cet exemple nous montre comment les informaticiens inventent une façon de coder un texte en mémoire. Premièrement, on décide de l'**ensemble des caractères** dont on a besoin, et on assigne à chacun un **identifiant numérique unique appelé code**. Cet ensemble est appelé **jeu de caractères codés** (en anglais *charset*, abréviation de *character set*) et peut se résumer dans un tableau de correspondance comme ci-dessus.
Ensuite, il faut déterminer l'**encodage** (*encoding*), c'est-à-dire **la façon de transcrire un texte** grâce aux codes des caractères qui le composent, selon un jeu de caractères donné. Le moyen le plus simple est d'écrire directement chaque code (auquel cas on parle de **page de code** — *charmap*) ; le jeu et l'encodage sont alors confondus. On a procédé de cette façon dans l'exemple ci-dessus, mais ce n'est pas toujours satisfaisant. C'est pourquoi il faut bien retenir la différence entre jeu de caractères et encodage.

Ça, c'est pour la théorie. Maintenant, place à la pratique ! L'exemple que je vous ai proposé était simple. Trop simple. La réalité est tout autre pour deux raisons principales :

1. Il y a bien plus de caractères à gérer.
   - Il y a des caractères particuliers, dits « de contrôle », qui ne servent pas pour un symbole « imprimable » mais donnent des indications aux programmes qui manipulent les chaînes de caractères. Le plus important est le caractère « fin de chaîne » qui sert à indiquer où s'arrête la chaîne (sinon, le programme n'aurait pas de moyen de le savoir et continuerait à lire ce qui se trouve après, ce qui nous vaudrait de belles erreurs).
   - De plus, il y a aussi les lettres minuscules, que je n'ai pas mises dans mon exemple. Il faudrait aussi pouvoir gérer les accents, les symboles de monnaie… voire, soyons fous, penser aux langues non latines (les Arabes et les Chinois ayant peut-être envie de parler leur langue). C'est là que ça devient vraiment problématique, comme nous allons le voir.
2. Il faut aussi considérer les contraintes matérielles. En effet, comme je vous l'ai dit, un ordinateur ne connaît que le binaire. Les bits sont regroupés par groupes de 8 appelés « *octets* ». Un octet ne peut stocker que les nombres entiers de 0 à 255 (soit 256 possibilités). Si cela ne suffit pas, on peut rassembler les octets par 2, 4 ou plus pour avoir de plus grands nombres. Dans nos encodages, il faudra tenir compte de ces limites, c'est-à-dire s'arranger pour que les codes de notre jeu tiennent tous sur un ou deux octets par exemple.

Pour bien cerner les problèmes causés par les encodages et la situation actuelle, le mieux reste encore de retracer leur histoire. En avant !

Au commencement était l'ASCII

Le premier encodage historique est l'**ASCII**, soit l'*American Standard Code for Information Interchange* (en français, le *code américain normalisé pour l'échange d'informations*). C'est une norme américaine, inventée en 1961, qui avait pour but d'organiser le bazar informatique à l'échelle nationale. Ce n'est pas le premier encodage utilisé mais on peut oublier les précédents.

Le terme « ASCII » est employé de manière incorrecte par beaucoup de monde sans savoir ce qu'il désigne réellement.
Le jeu de caractères ASCII utilise **7 bits** (et non 8 !) et dispose donc de **128 caractères uniquement, numérotés de 0 à 127**. En effet, il est paru à une époque où des regroupements par 7 au lieu de 8 étaient encore assez fréquents.

Sans plus attendre, voici la table de l'ASCII :

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Hé, mais j'y comprends rien à ton tableau, moi !
Chaque case correspond à l'un des 128 caractères de l'ASCII rangés dans l'ordre. Pour retrouver le code associé à un caractère, il faut regarder l'en-tête de sa ligne et celui de sa colonne ; ils contiennent des valeurs en hexadécimal qu'il faut additionner. Ainsi le caractère Z a pour code hexadécimal 50+A = 5A (soit 90 en décimal).

Mais pourquoi de l'hexadécimal ? Je croyais qu'on travaillait en binaire ??!
L'hexadécimal est une manière « raccourcie » de représenter du binaire sur 1 octet = 8 bits = 2×4 bits (c'est moins fastidieux d'écrire 2 symboles que 8 !). Sur 4 bits, on peut représenter… 16 nombres ! Donc deux symboles en hexadécimal = 8 symboles en binaire ;-)

Regardons un peu ce qu'il y a dans l'ASCII :

- les 26 lettres de l'alphabet latin, en majuscules (41 — 5A) et en minuscules (61 — 7A), ainsi que les chiffres de 0 à 9 ;
- divers signes de ponctuation, et d'autres symboles tels que les crochets, les accolades, l'arrobase… ;
- des « caractères blancs », c'est-à-dire l'espace SP mais aussi d'autres tels que le retour à la ligne LF (eh oui, c'est aussi un caractère).
- des caractères de contrôle non imprimables (00 — 1F, et 7F). Par exemple, le caractère NUL (*null*) sert à marquer la fin d'une chaîne de caractères.

Comme vous voyez, l'ASCII est simple. Il ne comporte que le strict nécessaire pour l'époque, pour utiliser un ordinateur… en anglais : que les 26 lettres de l'alphabet latin de base, pas d'accents, etc.

Vous vous en doutez certainement, les autres pays ont voulu pouvoir utiliser leur propre langage correctement. C'est pourquoi des extensions de l'ASCII sont apparues.
Certaines gardent la base ASCII et utilisent le 8e bit laissé libre afin d'avoir plus de caractères à disposition. Ainsi, les codes de 0 à 127 correspondent encore aux caractères ASCII, tandis que les codes supérieurs servent pour les nouveaux caractères. D'autres restent sur 7 bits, et modifient carrément les 128 caractères de l'ASCII pour leur propres besoins.
Vous imaginez la pagaille monstrueuse que ça a été, lorsque chaque pays ou groupe linguistique s'est mis à éditer sa propre page de code. Ça fonctionnait bien tant que les documents ne quittaient pas la zone où leur propre encodage était en usage, mais les échanges internationaux étaient sujets à problèmes. Comme un même code signifiait des caractères différents d'un jeu à l'autre, le récepteur ne lisait pas la même chose que le destinateur.
Il y a bien eu des tentatives de stopper la multiplication des pages de code, mais globalement elles ont été insuffisantes.

ISO 8859 : 8 bits pour les langues latines

Plus tard, une norme mieux pensée a fait son apparition : la norme **ISO 8859**. Cette fois, elle utilise **8 bits donc 256 caractères au maximum**. Le standard ISO 8859 comporte en fait plusieurs « parties », c'est-à-dire des pages de code indépendantes, nommées ISO 8859-*n* où *n* est le numéro de la partie.

ISO 8859 a été pensée afin que les parties soient le plus largement compatibles entre elles. Ainsi, elle englobe l'ASCII (codes 0 à 127) comme base commune, et les codes 128 à 255 devaient accueillir les caractères propres à chaque page de code, en s'arrangeant pour que des caractères identiques ou proches d'une page à l'autre occupent le même code.

Ce standard a principalement servi aux langues latines d'Europe pour mettre au point une page de code commune. À elles seules, elles utilisent finalement 10 parties d'ISO 8859, parfois appelées *latin-1*, *latin-2*, etc. ; ces parties correspondent à des évolutions dans la page de code latine de base (*latin-1*, ou officiellement ISO 8859-1) afin de rajouter certains caractères pour compléter des langues.

Les caractères ajoutés en ISO 8859 par rapport au jeu ASCII sont donné ci-dessous :

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	NBSP	¡	¢	£	¤ €	¥	¦ Š	§	¨ š	©	ª	«	¬	-	®	¯
B0	°	±	²	³	´ Z	µ	¶	·	¸ z	¹	º	»	¼ Œ	½ œ	¾ Ÿ	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Les codes de 80 à 9F sont laissés inutilisés par le standard ISO 8859.

ISO 8859 a aussi été utilisé pour l'alphabet cyrillique (ISO 8859-5), l'arabe (ISO 8859-6), le grec (7) l'hébreu (8) et même le thaï (11). Il existe au total 16 parties et il n'y en aura pas plus. En effet, on privilégie désormais le développement de l'Unicode.

Unicode, la solution ultime

Avec des normes comme ISO 8859, on commençait à s'en tirer pas trop mal. Les problèmes sont atténués, mais subsistent (et si vous rédigez un document en français mais voulez y insérer de l'arabe ?). Finalement, des illuminés se sont dit : « Et si on créait un jeu de caractères unique pour tout le monde ? »
De cette idée toute simple sont nés deux monuments : le standard ISO 10646 et Unicode.

ISO 10646 voit le jour en 1990. Il s'agit du **Jeu universel de caractères** ou JUC (en anglais, *UCS* pour *Universal Character Set*). Celui-ci a été pensé pour pouvoir accueillir n'importe quel caractère existant de n'importe quelle langue du monde. Un travail titanesque ! Concrètement, c'est un bête jeu de caractères, sauf que celui-ci offre pas moins de 2 097 152 codes ! À l'origine, il allait même jusqu'à 4 294 967 296, mais il a rapidement été restreint : c'est déjà bien suffisant.
Actuellement, environ 17% des codes sont déjà attribués à des caractères. Comme vous le voyez, il nous reste encore de la place à revendre. Et pourtant, dans ces 17%, on a tout mis ou presque : les caractères de tous les anciens jeux, tous les alphabets modernes, pléthore de symboles…
Le JUC n'est peut-être pas la solution définitive et éternelle (en informatique, il ne faut jamais dire « jamais »), mais il s'en rapproche suffisamment pour qu'on dorme sur nos deux oreilles pendant le siècle à venir. :)

Devant une telle quantité, les polices d'écriture (qui n'ont rien à voir avec les encodages, il s'agit ici d'**affichage** des caractères et non de leur **codage** en mémoire) peinent à suivre.
Unicode est une norme publiée pour la première fois en 1991. On peut la voir comme une extension d'ISO 10646. En fait, les deux normes sont développées parallèlement et synchronisées en permanence.
Il y a une notation officielle pour désigner n'importe quel caractère Unicode : **U+*xxxx***, où *xxxx* est le code hexadécimal (jusqu'à 6 chiffres). Par exemple, U+0041 correspond à la lettre A majuscule. Unicode, c'est cependant plus que cela. Ce standard décrit également des algorithmes de traitement, notamment pour la gestion des différents sens d'écriture, et surtout, ce qui nous intéresse ici, des encodages permettant de transcrire le JUC.