Normes d'encodage des caractères

Nous avons vu que les nombres entiers et flottants sont codés en binaires, ainsi bien entendu que les booléens. Il en est de même pour les caractères. Cependant de nombreuses normes existent. Les raisons en sont pour la plupart historiques, mais pas seulement. Nous regarderons dans ce cours trois de ces normes, dont la dernière - l'**unicode** - est aujourd'hui celle majoritairement utilisée.

1. Historiquement - la norme ASCII

A partir des années 1960, les ordinateurs commencent à être équipés d'un clavier et d'un rouleau pour imprimer les sorties. Il fut donc nécessaire de décider comment manipuler les caractères alphanumériques.

La décision fut prise de représenter *les caractères de l'alphabet anglais* sur un octet de la manière suivante :

- les 7 octets de poids faible (les plus à droite) serviront à associer une valeur numérique à chaque caractère ;
- l'octet de poids fort (celui le plus à gauche) sera le **bit de parité** - une clé de contrôle.

Les ordinateurs de cette époque étant peu fiables au niveau des transmissions et traitement des bits, il arrivait des erreurs lors du traitement des signaux et un octet `1101 1011` pouvait vite se transformer en `1001 1011` : les données seraient corrompues et le mauvais caractère serait affiché. Le **bit de parité** est un bit qui prenait la valeur 0 ou 1 selon la parité du nombre de 1 dans les 7 bits représentant le caractère. Ainsi, si les 7 bits sont `101 1001`, le bit de parité est à `0`, et l'octet complet devient `0101 1001`. Si celui reçu est `0101 0001`, un simple calcul avec le bit de parité permet de constater que le traitement du signal est défectueux.

Il reste donc 7 bits pour encoder les caractères, soit $2^7 = 128$ possibilités. Ce codage est appelé American Standard Code for Information Interchange, soit **ASCII**, et est présenté dans la *table de jeu de caractères ASCII* ci-dessous :

Base 16		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	Base 2	0000	001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
000	0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
001	0001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
002	0010	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
003	0011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004	0100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005	0101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	}	^	_
006	0110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007	0111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

\noindent La table ASCII contient 95 caractère imprimables : \trianglenoir \begin{itemize} \item les chiffres de 0 à 9 ; \item les lettres minuscules de a à z et majuscules de A à Z ; \item des symboles mathématiques et de ponctuation. \end{itemize} \noindent Les 32 premiers caractères de 00 à $1F$, ainsi que le 128ème caractère FF ne sont pas imprimables, ils correspondent à des commandes de contrôle de terminaux informatiques (le caractère FF est la commande pour effacer le caractère précédent, le 07 provoque l'émission d'un signal sonore, etc).

\noindent Il faut différencier la notion de \textbf{jeu de caractère} (\textit{Character set} en anglais) de celle de \textbf{police de caractère}. Dans une police de caractère, chaque glyphe est associé à un numéro correspondant à un caractère du \textbf{jeu de caractère}. Intrinsèquement, un ordinateur ne fait aucune différence entre deux glyphes de la même lettre. \begin{info}{Glyphes} \noindent Un \textbf{glyphe} est une représentation graphique d'un signe typographique, c'est-à-dire d'un caractère ou bien un idéogramme.\ \parbox{0.3\linewidth}{ \begin{center} \includegraphics[width = 0.6\linewidth]{Glyphe_a_Caliban.png}\ Glyphe \texttt{a} police Caliban \end{center} }\hfill \parbox{0.3\linewidth}{ \begin{center} \includegraphics[width = 0.6\linewidth] {Glyphe_a_Caston_Italic.png}\ Glyphe \texttt{a} police Caston Italic \end{center} }\hfill\parbox{0.3\linewidth}{ \begin{center} \includegraphics[width = 0.6\linewidth]{Glyphe_Ki.png}\ Glyphe \texttt{ki} (kanji) \end{center} }

\end{info} \begin{ExerciceNomme}{En Python} \begin{enumerate} \item Pour obtenir un caractère correspondant à un code hexadécimal donné, il faut convertir en entier le nombre hexadécimalavec `int` puis passer par la fonction `chr` : texttt{chr(int('Ox41',16))}.\ Vérifiez quelques caractères. Comment faire à partir du code binaire ? \caserep{\linewidth}{1} \item testez \texttt{ord('A')}. Que renvoie cette fonction ? (\textit{Vous pourrez tester les fonctions \texttt{hex()} et \texttt{bin()}}) \caserep{\linewidth}{2} \end{enumerate} \end{ExerciceNomme} \begin{ExerciceNomme}{Discussion} \noindent Quelles sont les limites de la norme ASCII ? \caserep{\linewidth}{2} \end{ExerciceNomme}

\section{La norme ISO-8859}

Dès la fin des années 60, alors que la qualité des ordinateurs s'améliore, il devient possible de bénéficier de l'octet de poids fort pour disposer de $2^8 = 256$ possibilités de codage. Les différentes constructeurs d'ordinateurs se précipitent sur cette possibilité afin de palier aux défauts de l'ASCII, malheureusement sans se coordonner. Différentes normes voient le jour, appelées \textbf{ASCII étendues}, pour la plupart incompatibles entre elles. Par exemple IBM produit une table, la $CP437$, possédant des accents, ainsi que de nombreux symboles de tracés de boites - les interfaces graphiques n'existant pas encore sur ces machines : \begin{center} \includegraphics[width=0.6\linewidth]{CP437.jpg} \end{center}

\noindent Malgré tout, une norme arrive à être établie, l'ISO-8859, avec les conventions suivantes : \trianglenoir \begin{itemize} \item le codage des caractères présents dans la tables ASCII est conservé (principe de \textbf{rétro-compatibilité}) ; \item on conserve le principe de caractères sur 1 octet, avec utilisation complète de 8 bits. \end{itemize}

\noindent Plusieurs jeus de caractères coexistent alors :\ \parbox{0.45\linewidth}{ \begin{center} \includegraphics[width=0.9\linewidth]{ISO88591.png}\ ISO 8859-1 (Occidental) \end{center}

}\hfill\parbox{0.45\linewidth}{ \begin{center} \includegraphics[width=0.9\linewidth]{ISO88595.png}\ ISO 8859-5 (Cyrillique) \end{center} }\ \noindent Le jeu occidental ISO 8859-1, aussi appelé \textbf{ISO-Latin-1}, a été le jeu par défaut du web avant d'être à son tour remplacé par l'unicode. La norme ISO 8859-1 a été révisée en ISO 8859-15 à la fin des années \np{1990} pour y ajouter de nouveaux caractères (comme le symbole \euro),

\noindent A noter que les efforts de normalisation ont parfois mené à des absurdités : pas moins de 5 normes ISO 8859 différentes coexistent pour l'aplphabet cyrillique...

\noindent Cependant un problème majeur demeurait : certaines langues, comme le chinois, ne pouvaient tout simplement pas être utilisées, le nombres de glyphes nécessaires étant bien supérieur à 256. \begin{ExerciceNomme}{Avec Python} Il est possible avec Python de basculer d'un jeu de caractère à un autre. Le principe est de récupérer l'octet correspondant à un caractère dans un \texttt{charmap}, et d'afficher le caractère correspondant dans le nouveau \texttt{charmap}. Pour cela on utilise les méthodes suivantes : \begin{itemize} \item \texttt{.encode(charmap)} : transforme une chaine de caractère de la \textit{charmap} passée en argument en une séquence d'octets ; \item \texttt{.decode(charmap)} : transforme une séquence d'octets en une chaine de caractère de la \textit{charmap} passée en argument. \end{itemize}

\begin{enumerate} \item Tester l'encodage de la chaine \texttt{'A'} d'abord en \texttt{ascii}, puis en \texttt{ISO8859-1}, puis en \texttt{ISO8859-5} et enfin en \texttt{CP437}. Que constate-t-on ? \caserep{\linewidth}{1} \item Faire de même avec la chaine \texttt{'é'}. Que constate-t-on ? \caserep{\linewidth}{2} \item A quels caractères correspond l'octet dont le code Python est \texttt{b'$\xe7$'} selon l'encodage utilisé : \coche \begin{itemize} \item en Latin-1 : \hfill\caserepsanssaut{3}{0.5} \item en CP437 : \hfill\caserepsanssaut{3}{0.5} \item en ISO8859-5 : \hfill\caserepsanssaut{3}{0.5} \end{itemize} \item Quel est le code Latin-1 cordresponant à la lettre '\oe{}' ? \caserep{\linewidth}{1} \end{enumerate}

\end{ExerciceNomme}

\section{L'encodage unicode UTF-8} \noindent\parbox{0.6\linewidth}{ \noindent Internet naisant, les problèmes liés aux différents encodages ont augmentés exponentiellement (et perdurent toujours aujourd'hui sur certains logiciels). La solution trouvée est le standard \textbf{unicode}, née au début des années 1990.\ L'objectif de cette norme est triple : \coche \begin{itemize} \item Rétro-compatibilité avec ASCII, et en grande partie avec ISO8859-1 ; \item Gestion d'un plus grand nombre de caractères ; \item Affichage de textes bi-directionnels. \end{itemize}

}\hfill\parbox{0.35\linewidth}{ \begin{center} \includegraphics[width=0.9\linewidth]{Martine.jpg} \end{center}

} \noindent Un des formats d'encodage de l'unicode, et le plus utilisé, est l'\textbf{UTF-8} pour \og\textit{8-Bit Universal Character Set Transformation Format}\fg. L'UTF-8 code les caractères en utilisant jusqu'à 4 octets. Il attribue à chaque caractère unicode existant une séquence de bits précise. La force de cette norme est de \textbf{ne pas forcément utiliser 4 octets}. La table ASCII est d'ailleurs codée sur 8 bits, pour garantir la rétro-compatibilité.\ En utf-8, chaque caractère est représenté sous la forme d'un bloc \texttt{U+xxxx} (où \texttt{xxxx} est un hexadécimal de 4 à 6 chiffres, entre \texttt{U+0000} et \texttt{U+10FFFF}). La plage ainsi définie permet d'attribuer jusqu'à \np{1 114 112} caractères. A l'heure actuelle, on recense environ \np{130 000} caractères dans unicode.\ Pour savoir combien d'octets on va utiliser les bits de poids fort du premier octet (celui de gauche) et pour savoir qu'un octet est la suite du précédent on commence par 10 : \begin{center} \begin{tabular}{|c|c|c|} \hline Codes& Encodage en UTF-8& Caractères dans cet intervalle \\\hline jusqu'à \textbf{U+007F} ($2^7 - 1$)& $0bbbbbbb$ & latin de base (ASCII)\\\hline

jusqu'à \textbf{U+07FF} ($2^{11} - 1$) & $110bbbbb$ $10bbbbbb$& alphabets d'Europe et du Moyen-Orient\\\hline jusqu'à \textbf{U+FFFF} (2^{16-1}) & $1110bbbb$ $10bbbbbb$ $10bbbbbb$& La quasi-totalité des alphabets actuels\\\hline jusqu'à \textbf{U+10FFFF} ($2^{21} - 1$) & ... & tous les caractères\\\hline \end{tabular} \end{center} \noindent Cette norme permet donc d'être rapide pour les alphabets courants. Elle

présente le défaut de ne pas pouvoir aller directement chercher le 10ème caractère d'une phrase puisque le nombre d'octets par lettre est variable. L'UTF-16, lui, est codé sur 4 octets ou 2 octets en fonction du code du caractère. La norme UTF-32 utilise elle 32 bits en permanence. Cela consomme bien plus de mémoire mais permet de trouver très rapidement le xème caractère d'une chaîne de caractères.

\noindent L'UTF-8 est utilisé par $82,2\%$ des sites web en décembre 2014, puis 87.6% en 2016 et enfin $95,2\%$ en octobre 2020.\

\noindent Par sa nature, UTF-8 est d'un usage de plus en plus courant sur Internet, et dans les systèmes devant échanger de l'information.

\noindent Il s'agit également du codage le plus utilisé dans les systèmes GNU / Linux et compatibles pour gérer le plus simplement possible des textes et leurs traductions dans tous les systèmes d'écritures et tous les alphabets du monde.

\noindent Concrètement, UTF-8 est utilisé par quasi tous les serveurs Web. Aujourd'hui, il n'y a plus de questions à se poser : \textbf{choisissez systématiquement l'encodage utf-8 pour vos travaux}, et comme \textit{explicit is better than implicit}, expliquez clairement dans tous vos codes l'encodage utilisé : \coche \begin{itemize} \item en HTML : \begin{lstlisting}[language = html, numbers=none] \end{lstlisting} \item au début d'un script Python, avec le shebang : \begin{lstlisting}[language = html, numbers=none]

```
-- coding: utf-8 --
```

```
\end{lstlisting}
```

\end{itemize} \begin{ExerciceNomme}{Jouer avezc les encodages} \begin{enumerate} \item On considère la liste Python suivante : \texttt{[233, 112, 97, 116, 97, 110, 116, 32, 33]}. Quel est le message caché ? \caserep{\linewidth}{1} \item\begin{enumerate} \item Créez une page vide en HTML avec la code suivant dans Notepad++, en vérifiant que l'encodage est bien UTF-8 : \begin{lstlisting} [language = HTML]

```
</body>
```

\end{lstlisting} \item Insérez la phrase suivante dans le corps du document: \og Portez ce vieux whisky au juge blond qui fume sur son île intérieure, à côté de l'alcôve ovoïde, où les bûches se consument dans l'âtre, ce qui lui permet de penser à la caenogénèse de l'être dont il est question dans la cause ambiguë entendue à Moÿ, dans un capharnaüm qui, pense-t-il, diminue çà et là la qualité de son œuvre. \fg \item Ouvrez ensuite le fichier dans un navigateur. \item Remplacez maintenant la ligne \ \texttt{}\ par la ligne \texttt{} \item Ouvrez de nouveau le fichier à l'aide d'un navigateur. \caserep{\linewidth}{6} \item Ouvrez maintenant le fichier HTML avec le logiciel \texttt{HexEditorNeo}. Que retrouve-t-on ? \caserep{\linewidth}{4} \end{enumerate} \end{enumerate} \end{ExerciceNomme}

Unknown environment 'footnotesize'

\end{document}