Chapitre 3
Représentations des caractères en machine

Objectifs :

- Identifier l'intérêt des différents systèmes d'encodage
- Convertir un fichier texte dans différents formats d'encodage

Vous avez vu que l'architecture des circuits composant un ordinateur impose l'utilisation du langage binaire. Ainsi, tout ce qui est manipulé par l'ordinateur doit être « traduit » pour être représenté dans la machine par une suite de 0 et de 1.

Vous allez découvrir dans ce chapitre comment sont représentés en machine les textes (et donc les caractères) qui sont constamment manipulées par un ordinateur.

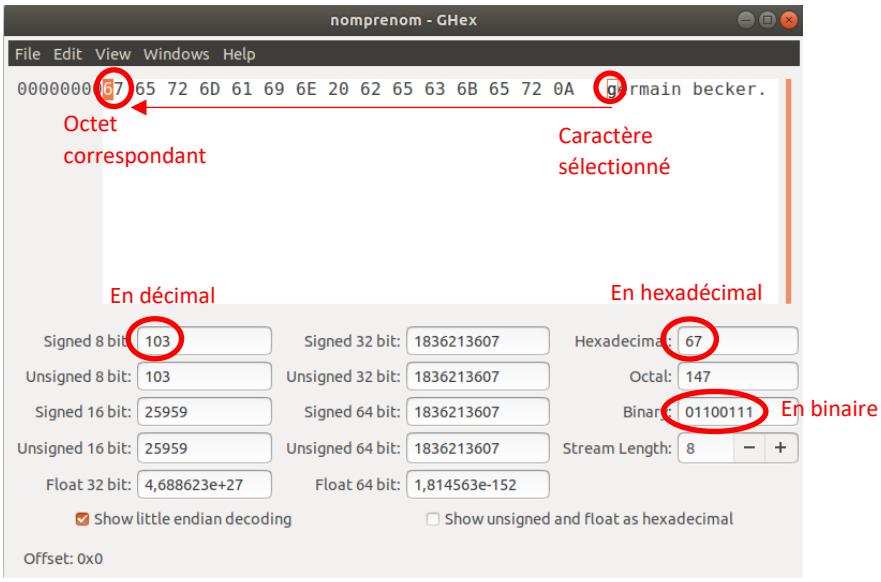
Représentation d'un texte en machine

Bon, vous avez appris à convertir des nombres entiers en binaire. Cela reste une conversion entre des nombres. Les caractères d'un texte, ne sont pas des nombres, et pourtant il faut les représenter en binaire pour que la machine puisse les utiliser, les manipuler, etc.

L'activité qui suit décrit comment on peut représenter des textes en machine.

Activité 1 : Visualiser le contenu d'un fichier texte

1. Lancez un éditeur de texte : gedit sous Linux ou Notepad++ sous Windows.
2. Écrivez ensuite sur une seule ligne (sans retour à la ligne) vos nom et prénom tout en minuscules et sans accent que vous nommerez « nomprenom ».
3. Enregistrez votre fichier texte.
4. Lancez ensuite un éditeur hexadécimal qui permet de visualiser le contenu hexadécimal (et binaire) d'un fichier :
   - GHex (GNOME Hex Editor) sous Linux 
   - HxD Hex Editor sous Windows par exemple. 
5. Ouvrez ensuite votre fichier « nomprenom » avec ce logiciel. Avez GHex vous devez obtenir l'affichage suivant :



Ainsi, vous constatez que votre fichier texte est bien codé par une suite de nombres binaires (ici affichés en hexadécimal par soucis de simplification). Pour comprendre pourquoi la lettre « g » est codée par le nombre binaire 01100111 ($= (67)_{16}$), je vous invite à lire ce qui suit.

Evolution des normes d'encodage

Nous allons maintenant voir par quels nombres sont codés chaque caractère : on parle de l'encodage des caractères. Il existe beaucoup de normes définissant le codage des caractères. Celles-ci sont apparues au fur et à mesure de l'histoire de l'informatique pour pallier aux nouveaux besoins et corriger les défauts des normes précédentes. Nous ne parlerons ici que des trois normes (les plus connues) mais sachez qu'il en existe des centaines.

L'idée est la même pour chacune d'elle :

- une table de caractères associe à chaque caractère son « numéro » de code (ou index, ou point de code) dans la table, qui est un entier positif (donné sous forme décimale, hexadécimale ou binaire) ;
- cet entier est codé par une séquence de bits en machine selon un certain encodage.

ASCII

Avant 1960 de nombreux systèmes de codage de caractères existaient, ils étaient souvent incompatibles entre eux. **En 1960**, l'organisation internationale de normalisation (ISO) décide de mettre un peu d'ordre dans ce bazar en créant la norme ASCII (American Standard Code for Information Interchange). À chaque caractère est associé un nombre binaire sur 8 bits (1 octet). En fait, **seuls 7 bits sont utilisés** pour coder un caractère, le 8ème bit n'est pas utilisé pour le codage des caractères (il vaut toujours 0). Avec 7 bits il est possible de coder jusqu'à 128 caractères ce qui est largement suffisant pour un texte écrit en langue anglaise (pas d'accents et autres lettres particulières).

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Exemple de lecture : la lettre « b » minuscule correspond au caractère $(98)_{10}$ de la table ASCII et correspond au code binaire $(01100010)_2$ (ou $(62)_{16}$).

Exercice 1 :

1. Calculer le code binaire de la première lettre de votre prénom (minuscule non accentuée).
2. Vérifiez la correspondance entre ce tableau et les valeurs affichées par l'éditeur hexadécimal du fichier « nomprenom » de l'activité 1.

ISO-8859-1

La norme ASCII convient bien à la langue anglaise, mais pose des problèmes dans d'autres langues, par exemple le français. En effet l'ASCII ne prévoit pas d'encoder les lettres accentuées et il n'y a plus de « place » dans la table ASCII pour ajouter de nouveaux caractères.

C'est pour répondre à ces problèmes qu'est née la norme ISO-8859-1 **en 1986** qui va doubler le nombre de caractères en utilisant le 8ème bit. Cette norme reprend les mêmes principes que l'ASCII, mais les nombres binaires associés à chaque caractère sont codés **sur 8 bits**, ce qui permet d'encoder jusqu'à 256 caractères. Cette norme va être principalement utilisée dans les pays européens puisqu'elle permet d'encoder les caractères utilisés dans les principales langues européennes (la norme ISO-8859-1 est aussi appelée « **Latin-1** » car elle permet d'encoder les caractères de l'alphabet dit « latin »).

Problème, il existe beaucoup d'autres langues dans le monde qui n'utilisent pas l'alphabet dit "latin", par exemple le chinois ou le japonais ! D'autres normes ont donc dû voir le jour, par exemple la norme "GB2312" pour le chinois simplifié ou encore la norme "JIS_X_0208" pour le japonais.

Cette multiplication des normes a très rapidement posé problème. Imaginons un français qui parle le japonais. Son traitement de texte est configuré pour reconnaitre les caractères de l'alphabet "latin" (norme ISO-8859-1). Un ami japonais lui envoie un fichier texte écrit en japonais. Le français devra modifier la configuration de son traitement afin que ce dernier puisse afficher correctement l'alphabet

japonais. S'il n'effectue pas ce changement de configuration, il verra s'afficher des caractères ésotériques.

Remarque : Une autre norme ISO très connue, appelée **ISO-8859-15** ou encore Latin-9, est presque identique au Latin-1 à 8 caractères près (le Latin-9 contient par exemple les caractères œ, Œ et € qui ne sont pas dans la table Latin-1).

Unicode

Pour éviter ce genre de problème, **en 1991** une nouvelle norme a vu le jour : Unicode

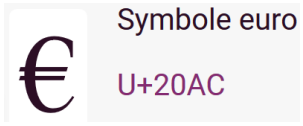
Unicode a pour ambition de rassembler tous les caractères existant afin qu'une personne utilisant Unicode puisse, sans changer la configuration de son traitement de texte, à la fois lire des textes en français ou en japonais

Unicode est uniquement une table qui regroupe tous les caractères existants dans le monde, il ne s'occupe pas de la façon dont les caractères sont codés dans la machine. Unicode accepte **plusieurs systèmes de codage** : UTF-8, UTF-16, UTF-32. Le plus utilisé, notamment sur le Web, est UTF-8.

Pour encoder les caractères Unicode, **UTF-8 utilise un nombre variable d'octets (de 1 à 4 octets)** : les caractères "classiques" (les plus couramment utilisés) sont codés sur un octet, alors que des caractères "moins classiques" sont codés sur un nombre d'octets plus important (jusqu'à 4 octets). Un des avantages d'UTF-8 c'est qu'il est totalement compatible avec la norme ASCII : les caractères Unicode codés avec UTF-8 ont exactement le même code que les mêmes caractères en ASCII.

Exemple de codage en UTF-8 : le caractère « € »

- https://unicode-table.com/fr/#control-character : on tape « € » dans la barre de saisie
- On trouve ainsi son nombre Unicode : U+20AC
- On repère dans le tableau ci-dessous sur combien d'octets se code le nombre U+20AC :



Définition du nombre d'octets utilisés dans le codage (uniquement les séquences valides)

Caractères codés	Représentation binaire UTF-8	Premier octet valide (hexadécimal)	Signification
U+0000 à U+007F	0xxxxxxx	00 à 7F	1 octet, codant 7 bits
U+0080 à U+07FF	110xxxxx 10xxxxxx	C2 à DF	2 octets, codant 11 bits
U+0800 à U+0FFF	11100000 101xxxxx 10xxxxxx	E0 (le 2e octet est restreint de A0 à BF)	3 octets, codant 16 bits
U+1000 à U+1FFF	11100001 10xxxxxx 10xxxxxx	E1	
U+2000 à U+3FFF	11100010 10xxxxxx 10xxxxxx	E2 à E3	
U+4000 à U+7FFF	11100011 10xxxxxx 10xxxxxx	E4 à E7	
U+8000 à U+BFFF	11100100 10xxxxxx 10xxxxxx	E8 à EB	
U+C000 à U+CFFF	11100110 10xxxxxx 10xxxxxx	EC	
U+D000 à U+D7FF	11100111 100xxxxx 10xxxxxx	ED (le 2e octet est restreint de 80 à 9F)	
U+E000 à U+FFFF	11101000 10xxxxxx 10xxxxxx	EE à EF	
U+10000 à U+1FFFF	11110000 1001xxxx 10xxxxxx 10xxxxxx	F0 (le 2e octet est restreint de 90 à BF)	4 octets, codant 21 bits
U+20000 à U+3FFFF	11110000 101xxxxx 10xxxxxx 10xxxxxx		
U+40000 à U+7FFFF	11110001 10xxxxxx 10xxxxxx 10xxxxxx	F1	
U+80000 à U+FFFFF	11110011 10xxxxxx 10xxxxxx 10xxxxxx	F2 à F3	
U+100000 à U+10FFFF	11110100 1000xxxx 10xxxxxx 10xxxxxx	F4 (le 2e octet est restreint de 80 à 8F)	

Source : https://fr.wikipedia.org/wiki/UTF-8

- U+20AC étant compris entre U+2000 et U+3FFF on sait alors qu'il faudra 3 octets pour coder le symbole € en UTF-8.
- La norme indique que les 16 bits de son code binaire seront répartis sur 3 octets de la façon suivante :

Trois '1' pour indiquer que le codage se fait sur trois octets

En vert, les 16 bits réservés au code binaire du caractère
Chaque zone réservée au code binaire est précédée des deux bits '10'

- On convertit ensuite la valeur hexadécimale 20AC en binaire :

Hexadécimal	2	0	A	C
Binaire	**0010**	**0000**	**1010**	**1100**

- On répartit ensuite ces valeurs binaires sur les 16 bits réservés par la norme (zones vertes) :

1110**0010** 10**000010** 10**101100**

- On peut alors convertir ces trois octets en hexadécimal, cela donne : E2 82 AC.

```
>>> hex(int('111000101000001010101100',2))
'0xe282ac'
```

Exercice 2 :

1. Déterminez le code binaire du caractère « é » Unicode codé en UTF-8.
2. Vérifiez en tapant ce caractère dans un fichier texte et en visualisant celui-ci avec un éditeur hexadécimal.

Remarque : L'encodage UTF-8 de la norme Unicode devient universelle, c'est celle qu'il faut utiliser. Si on inspecte l'en-tête d'un fichier HTML, celle-ci contient une ligne définissant la norme d'encodage des caractères utilisée et la quasi-totalité des nouvelles pages Web utilisent l'encodage UTF-8. Les fichiers Python (version 3) que vous écrivez sont également des fichiers texte et ceux-ci sont codés en UTF-8 avec Spyder par exemple.

```
<!DOCTYPE html>

<html>
    <head>
        <!-- en-tête de la page -->
        <!-- encodage des caractères -->
        <meta charset="UTF-8">
        <title>Ma première page</title>
    </head>
```

En-tête d'une page HTML

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 29 11:37:25 2019

@author: germain
"""
```

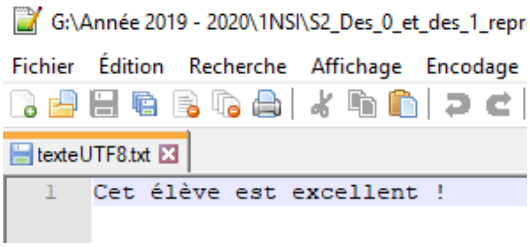
En-tête d'un nouveau script Python avec Spyder

Ces différentes normes peuvent causer des problèmes

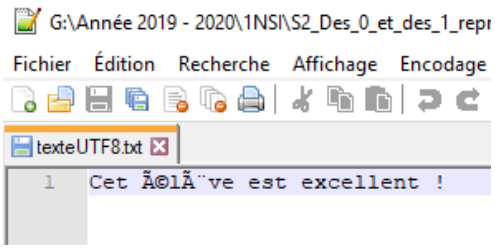
Comme les encodages des caractères diffèrent et qu'il en existe beaucoup, ils sont encore la source d'erreurs d'affichage. En effet, si l'encodage utilisé par un « logiciel » pour décoder le code binaire en machine d'un texte n'est pas le même que celui utilisé pour coder le texte, on peut avoir des surprises dans l'affichage.

Exemple : Si un texte contenant le symbole « é » a été codé en UTF-8 alors ce caractère a été codé sur 2 octets par $(C3A9)_{16}$. Imaginons que l'on souhaite lire ce fichier texte avec un éditeur qui est configuré sur l'encodage ISO-8859-1 (Latin-1) qui prévoit qu'un caractère est codé sur 1 octet. L'éditeur va alors lire deux caractères hexadécimaux C3 puis A9 et afficher les deux caractères correspondants de la table Latin-1, c'est-à-dire « Ã© » à la place du « é ».

Plus concrètement, voici ce que cela peut donner avec l'éditeur Notepad++ :



Enregistrement avec l'encodage UTF-8



Ouverture avec l'encodage Latin-1 (ISO-8859-1)

Ce genre de problèmes d'affichage était très courant, notamment sur le Web. Les logiciels et navigateurs parviennent désormais à mieux déterminer l'encodage utilisé pour afficher correctement les textes. De manière générale, la bonne pratique est d'utiliser UTF-8 comme encodage de vos fichiers.

Activité 2 : Convertir un fichier texte dans différents formats d'encodage avec Python

1. Ecriture d'un fichier

a. Lancez Spyder et recopiez le script suivant et enregistrez le fichier Python.

```
1 fichier = open ("monFichierLatin9.txt", mode="w", encoding="latin9")
2 fichier.write("Ce pull est très cher, il coûte 100 €.")
3 fichier.close()
```

Explications :

- Ligne 1 : on ouvre un fichier appelé « monFichierLatin9.txt » et comme ce fichier n'existe pas, un fichier vide est créé dans le même répertoire que le fichier Python. Ce fichier est ouvert en mode écriture ("w" pour write) et sera encodé en Latin-9.
- Ligne 2 : on écrit cette phrase dans `fichier` avec la méthode `write`.
- Ligne 3 : on ferme le flux d'écriture du fichier.

b. Ouvrez alors le répertoire de votre fichier Python, vous devriez y trouver un fichier nommé « monFichierLatin9.txt ».
Lancez ce fichier avec un éditeur de texte pour en visualiser le contenu.
Si certains caractères ne s'affichent pas correctement, il faut modifier l'encodage utilisé pour la lecture du fichier et sélectionner « Latin-9 » (ou « ISO 8859-15 »).

Avec Notepad++ (sous Windows) : allez sur l'onglet « Encodage » puis sur « Codage des caractères » puis « Langues d'Europe occidentale » puis « ISO 8859-15 »
Avec Gedit (sous Linux) : normalement la détection de l'encodage se fait automatiquement et il n'y a pas de soucis d'affichage.

2. Lecture d'un fichier

a. Retournez sur Sypder et recopiez le script suivant et enregistrez le fichier Python toujours dans le même répertoire.

```
1 fichier = open ("monFichierLatin9.txt", mode="r", encoding = "latin9")
2 contenu = fichier.read()
3 print(contenu)
4 fichier.close()
```

Explications :

- Ligne 1 : on ouvre un fichier existant appelé « monFichierLatin9.txt » ython.
  Ce fichier est ouvert en mode lecture ("r" pour read) avec l'encodage Latin-9 pour la lecture.
- Ligne 2 : on stocke dans la variable `contenu` le contenu du fichier (chaîne de caractères) grâce à la méthode `read`.
- Ligne 3 : on affiche la variable `contenu` dans la console.
- Ligne 4 : on ferme le flux de lecture.

b. Exécutez le script. Vous devez constater que l'affiche se fait convenablement puisque l'encodage utilisé est le même que celui utilisé pour l'écriture du fichier (question 1).

c. Modifier l'encodage pour la lecture du fichier en remplaçant `encoding = "latin9"` par `encoding = "latin1"`. Que constatez-vous ? Comment pouvez-vous l'expliquer ?

d. Mêmes questions avec `encoding = "utf8"`.

3. Conversion du format d'encodage

Complétez le programme suivant pour qu'il convertisse le fichier « monFichierLatin9.txt » en le fichier « monFichierUTF8.txt » encodé en UTF-8.

```
1 fichier_depart = open("...................", mode = "...", encoding = "latin9")
2 fichier_final = open("...................", mode = "...", encoding = "utf8")
3 contenu = ...              # on stocke le contenu de "fichier_depart"
4 fichier_final.write(...)   # on écrit le contenu dans "fichier_final"
5 fichier_depart.close()     # on ferme les deux flux
6 fichier_final.close()
```

On peut de la même manière convertir un fichier texte dans différents formats d'encodage.

Germain BECKER & Sébastien POINT, Lycée Mounier, ANGERS

Activité 3 : Un programme Python pour convertir en majuscules

1. Observez dans la table ASCII le décalage entre un caractère majuscule et minuscule d'une même lettre (non accentuée).

En python, les fonctions `ord` et `chr` permettent de convertir un caractère en son code Unicode décimal et réciproquement. Par exemple :

`ord('a')` → renvoie le code Unicode du caractère 'a'
`chr(125)` → renvoie le caractère dont le code Unicode est 125

```
In [1]: ord('a')
Out[1]: 97

In [2]: chr(97)
Out[2]: 'a'
```

2. Complétez le script Python suivant qui a pour but de demander à l'utilisateur d'entrer une lettre minuscule et qui doit afficher la même lettre mais en majuscule.

```
lettre_min = str(input("Entrez un caractère minuscule : "))
code_min = ord(lettre_min)
code_maj = code_min - ...
lettre_maj = chr(code_maj)
print(lettre_maj)
```

3. Adaptez ce script en une fonction appelée `min2maj` qui prend une lettre minuscule en paramètre et qui renvoie cette lettre mais en majuscule.
4. Que suffit-il de changer pour créer une fonction `maj2min` qui prend une lettre majuscule en paramètre et qui renvoie cette lettre mais en minuscule ?
5. Proposez maintenant un programme Python qui :
   - permet à l'utilisateur de saisir une phrase (majuscules et minuscules mélangées, signes de ponctuations) ;
   - transforme cette phrase pour qu'elle ne contienne que des lettres majuscules et signes de ponctuation ;
   - affiche cette phrase.

   Indications : Une boucle `for` permet de parcourir une chaîne de caractères très facilement (caractère par caractère). Il faudra ajouter des instructions conditionnelles pour tester si chacun des caractères parcourus est une lettre ou un autre type de caractère : pour cela, observez bien les codes Unicode des lettres minuscules.

Ressources :

- David Roche : https://pixees.fr/informatiquelycee/n_site/nsi_prem_texte.html
- E. Bansière & P. Jonin, ressources partagées DIU EIL Nantes, « Comment un ordinateur code-t-il un texte ? »
- Wikipédia : articles sur ASCII, Unicode, UTF-8