


PARTIE 2 : représentation des entiers relatifs

1. Première approche intuitive

Une première idée serait de coder le signe + par un 0 et le signe - par un 1.

Ainsi $6_{10} = 110_2$ se coderait alors en 0110_2 . Mais -6_{10} se coderait alors en 1110_2 .


 Exercice 1 : Représenter 14_{10} en binaire. Que constatez-vous ?

Par conséquent, nous allons devoir nous fixer un nombre de bits n comme espace mémoire. On conserve alors le premier pour le signe et les $n - 1$ autres pour le codage de la valeur absolue du nombre.

Ainsi que 8 bits : -6_{10} se code 10000110_2 .

 Exercice 2 : Déterminer le codage en binaire sur 8 bits puis sur 9 bits, des nombre suivants :

1. $11_{10} =$
2. $-35_{10} =$
3. $26_{10} =$
4. $-42_{10} =$

 Exercice 3 : Sur 8 bits, comment codera-t-on 0 ?

Un premier problème se pose avec cette méthode : le chiffre 0_{10} a deux représentations !

 Exercice 4 :

1. Coder avec cette méthode les nombres 17_{10} et -17_{10} sur 8 bits
2. Effectuer l'addition binaire de ces deux nombres

L'addition binaire ne donne pas des résultats corrects !
Il faut donc oublier cette méthode !!!

2. Le complément à deux

2.1. Cas des entiers naturels



PROPRIÉTÉ 1 : La représentation d'un nombre entier naturel en binaire se fait de la même manière que dans le cours précédent.

Si on dispose de n bits, le premier sera 0 pour indiquer que l'entier est positif. Et les $n - 1$ autres seront le codage en binaire de l'entier.

Cela signifie aussi que si l'on dispose de n bits, nous ne pourrons pas dépasser un entier plus grand que $2^{n-1} - 1$ puisque nous ne disposons que de $n - 1$ bits.

✓ **EXEMPLE 1** : Sur 5 bits, on a $12_{10} = 01100_2$. Si on en dispose que de 4 bits, on ne peut pas coder 12 en binaire si nous considérons les entiers relatifs. En effet, $12 > 2^{4-1} - 1$

2.2. Cas d'entiers négatifs

PREMIÈRE MÉTHODE

1. On code la valeur absolue du nombre en binaire
2. On remplace tous les 0 par des 1 et tous les 1 par des 0
3. On ajoute 1

✓ **EXEMPLE 2** : Codage de -12 sur 8 bits en complément à 2.

1. Codage de la valeur absolue : $12_{10} = 0000\ 1100_2$ sur 8 bits
2. Changement des bits : $0000\ 1100_2$ devient $1111\ 0011_2$
3. On ajoute 1 :

						1	1	
	1	1	1	1	0	0	1	1
+	0	0	0	0	0	0	0	1
	1	1	1	1	0	1	0	0

Il est possible de vérifier le résultat en additionnant -12 et 12 en binaire :

	1	1	1	1	1	1			
		1	1	1	1	0	1	0	0
+	0	0	0	0	1	1	0	0	
	1	0	0	0	0	0	0	0	0

Nous sommes **sur 8 bits**, le 1 en première position doit être enlevé. **Le résultat est bien 0.**

DEUXIÈME MÉTHODE

Une méthode beaucoup plus rapide permet de faire cette opération :

1. On code la valeur absolue du nombre en binaire
2. On recopie les valeurs des bits de droite à gauche jusqu'au premier 1 inclus.
3. Ensuite on inverse les bits jusqu'au dernier (celui de gauche)

✓ **EXEMPLE 3** : Codage de -12 sur 8 bits en complément à 2.

1. Codage de la valeur absolue : $12_{10} = 0000\ 1100_2$ sur 8 bits
2. On recopie les bits de droite à gauche jusqu'au premier 1 inclus : $0000\ 1100_2$ devient $0000\ 1100_2$
3. On inverse les bits restants à gauche : $0000\ 1100_2$ devient $1111\ 0100_2$

 **Exercice 5** : Déterminer la représentation des valeurs suivantes et faites la vérification avec l'addition binaire :

1. -19 codé sur 8 bits en complément à 2
2. -72 codé sur 16 bits en complément à 2
3. -124 codé sur 8 bits en complément à 2

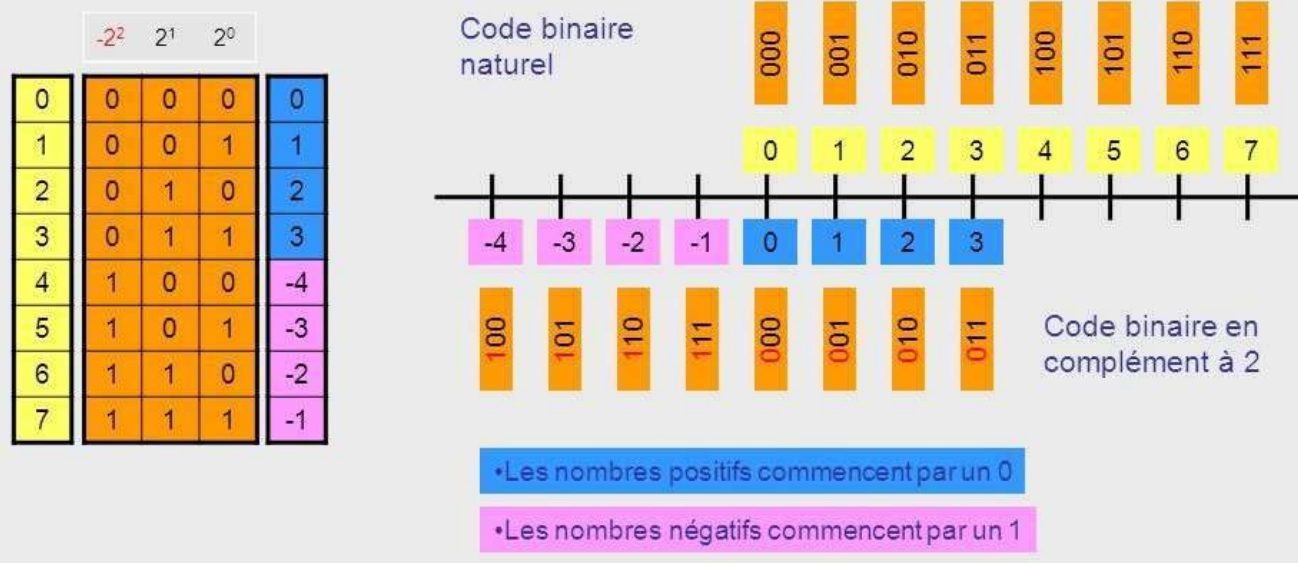


PROPRIÉTÉ 2 : Avec n bits, nous pouvons coder :

2^n valeurs comprises entre -2^{n-1} et $2^{n-1} - 1$

Notation en complément à 2

Principe



✏ Exercice 6 :

1. Sur 8 bits, donner un encadrement des entiers relatifs codables
2. Même question sur 9 bits ?

PARTIE 3 : Représentation des flottants

1. Les nombres dyadiques



DÉFINITION 1 : Nombres décimaux

Un nombre décimal est un nombre s'écrivant sous la forme $\frac{x}{10^n}$ où x est un entier relatif.

Visuellement un nombre décimal est un nombre acceptant un nombre fini de chiffres après la virgule

✓ EXEMPLE 1 :

Nombre	4	-3	0.25	$\frac{1}{3}$	$\frac{1}{4}$	π	$\frac{10}{3}$
Décimal ?	OUI	OUI	OUI	NON	OUI	NON	NON



DÉFINITION 2 : Nombre dyadiques

Par analogie avec les nombres décimaux, on appelle nombres dyadiques les nombres s'écrivant sous la forme $\frac{x}{2^n}$ avec x entier relatif et n entier naturel.

Exercice 1 : Dire si les nombres suivants sont dyadiques

Nombre	$\frac{13}{4}$	$\frac{1}{5}$	2.5	25	$\frac{10}{3}$	π	3.125
Dyadique ?							



DÉFINITION 3 : Développement dyadique d'un nombre

On appelle développement dyadique d'un nombre l'écriture binaire de ce nombre.

Ce développement dyadique est l'écriture en base 2 d'un nombre.



PROPRIÉTÉ 1 : Écriture binaire d'un nombre dyadique

Pour obtenir le développement dyadique d'un nombre qui peut s'écrire sous la forme $\frac{x}{2^n}$ on prend le nombre binaire correspondant à x et on insère une virgule avant le n -ième bit en partant de la fin.

✓ EXEMPLE 2 : Écrire le développement dyadique de 2.5

- 2.5 est un nombre dyadique car $2.5 = \frac{5}{2} = \frac{5}{2^1}$ donc ici $x = 5$ et $n = 1$
- On détermine l'écriture binaire de 5 : $5_{10} = 101_2$
- On insère une virgule avant le n ième bit en partant de la fin : $2.5_{10} = 10.1_2$

 Exercice 2 : Écrire le développement dyadique de 31.25_{10} :

2. De l'écriture décimale à la notation binaire

La méthode précédente est un peu fastidieuse. De plus les nombres non dyadiques ne peuvent pas être codés. Nous allons donc utiliser une nouvelle méthode.

✓ EXEMPLE 3 : Essayons de convertir un nombre : 5.1875 en binaire.

1. Conversion de la partie entière : $5_{10} = 101_2$
2. Multiplications successives par 2 de la partie décimale

$$0.1875 \times 2 = 0.375 = \underline{0} + 0.375$$

$$0.375 \times 2 = 0.75 = \underline{0} + 0.75$$

$$0.75 \times 2 = 1.5 = \underline{1} + 0.5$$


$$0.5 \times 2 = 1 = \underline{1} + 0.0$$

Dès que la multiplication par 2 donne une partie décimale nulle on s'arrête.

On a donc récupéré un certain nombre de parties entières que l'on va assembler : 0011

Il suffit ensuite de regrouper la partie entière et la partie décimale. Ainsi :

$$5.1875 = 101,0011_2$$

 Exercice 3 : Conversion de 12.6875 en binaire

1. Conversion de la partie entière : $12_{10} =$
2. Multiplications successives par 2 de la partie décimale :

Exercice 4 : Convertir en binaire les nombres suivants :

7.09375

13.325

Que remarquez-vous ?

Exercice 5 : Convertir en binaire le nombre suivant :

4,125

3. De l'écriture binaire à la notation décimale

Il est possible de retrouver une représentation décimale à partir d'une représentation en binaire.

✓ EXEMPLE 4 : Convertir $100,0101_2$

1. Convertir la partie entière : $100_2 = 4_{10}$
2. Convertir la partie décimale. On utilise un tableau des puissances de 2 négatives.

Rang du bit	-1	-2	-3	-4	-5	-6
Puissance de 2	$2^{-1} = 0.5$	$2^{-2} = 0.25$	$2^{-3} = 0.125$	$2^{-4} = 0.0625$	$2^{-5} = 0.03125$	$2^{-6} = 0.015625$
Bit	0	1	0	1	0	0
Valeur	0	0.25	0	0.0625	0	0

Il suffit ensuite d'additionner toutes les valeurs calculées :

$$100,0101_2 = 4 + 0.25 + 0.0625 = 4.3125_{10}$$

Exercice 6 : Trouver la représentation décimale de 101,001

1. Convertir la partie entière :

2. Convertir la partie décimale. On utilise un tableau des puissances de 2 négatives.

Rang du bit	-1	-2	-3	-4	-5	-6
Puissance de 2	$2^{-1} = 0.5$	$2^{-2} = 0.25$	$2^{-3} = 0.125$	$2^{-4} = 0.0625$	$2^{-5} = 0.03125$	$2^{-6} = 0.015625$
Bit						
Valeur						

Exercice 7 : Convertir en décimal les nombres suivants :

1101,100101

1000,11101

4. Notation scientifique


En base dix, il est possible d'écrire les très grands nombres et les très petits nombres grâce aux "puissances de dix" (exemples $6,02 \cdot 10^{23}$ ou $6,67 \cdot 10^{-11}$).

Il est possible de faire exactement la même chose avec une représentation binaire, puisque nous sommes en base 2, nous utiliserons des "puissances de deux" à la place des "puissances dix" (exemple $101,1101 \cdot 2^{10}$).

✓ EXEMPLE 5 : Pour passer d'une écriture sans "puissance de deux" à une écriture avec "puissance de deux", il suffit décaler la virgule : $1101,1001 = 1,1011001 \cdot 2^{11}$.

Pour passer de $1101,1001$ à $1,1011001$ nous avons décalé la virgule de 3 rangs vers la gauche d'où le 2^{11} . (Attention de ne pas oublier que nous travaillons en base 2 le « 11 » correspond bien à un décalage de 3 rangs de la virgule, car $11_2 = 3_{10}$).

✓ EXEMPLE 6 : Si l'on désire décaler la virgule vers la droite, il va être nécessaire d'utiliser des "puissances de deux négatives". $0,0110_2 = 1,10 \cdot 2^{-10}$. Nous décalons la virgule de 2 rangs vers la droite, d'où le « -10 ».

 Exercice 8 : Trouver l'écriture binaire scientifique de :

1. $1011,0111101_2$

2. 0.0745_{10}

Compléments sur la norme IEEE-754 (hors programme)

L'IEEE 754 est une norme pour la représentation des nombres à virgule flottante en binaire. Elle est la norme la plus employée actuellement pour le calcul des nombres à virgule flottante dans le domaine informatique.

Cette norme définit notamment 2 formats pour représenter des nombres à virgule flottante :

- *Simple précision* (32 bits : 1 bit de signe, 8 bits d'exposant (-126 à 127), 23 bits de mantisse),



- *Double précision* (64 bits : 1 bit de signe, 11 bits d'exposant (-1022 à 1023), 52 bits de mantisse).



✓ EXEMPLE 6 bis : nombre à virgule flottante simple précision : codage du nombre décimal $10,8125_{10}$

Pour le bit de signe : le signe - est codé par un 1 et le signe + par un 0.

Dans l'exemple, le bit de signe sera donc 0.

Ensuite, pour utiliser la norme on doit écrire le nombre à représenter sous la forme binaire $1,XXXX \times 2^{\text{exp}}$, comme nous avons pu le voir précédemment.

$$10,8125_{10} \Rightarrow 1010,1101 = 1,0101101 \times 2^{11}$$

La mantisse est la partie désignée par XXXX. Si elle ne fait pas 23 bits, on la complète par des 0 à droite.

Dans notre exemple, la mantisse est égale à 0101101. On notera donc 0101101 00000000 00000000 sur 23 bits.

L'exposant est codé sur 8 bits. Il peut être positif ou négatif. La norme impose de procéder à un décalage de 127 (en simple précision) de sorte à pouvoir coder sur ces 8 bits tous les exposants entiers de -127 à +128 (soit 256 valeurs différentes). On ajoute donc systématiquement 127 à la valeur de l'exposant avant de la coder en binaire. L'exposant 0 sera donc représenté par le nombre 127 en binaire, l'exposant 128 par 255, l'exposant - 127 sera lui codé comme l'entier 0.

Dans notre exemple, l'exposant est égal à 3, il sera donc représenté par le nombre 130 en binaire, soit 1000 0010.

Remarque :

Il peut parfois être nécessaire de rajouter des zéros pour avoir les 8 bits exigés. On les ajoutera à gauche cette fois-ci.

En appliquant la norme IEEE 754, le nombre $10,8125$ sera donc représenté par les 32 bits suivants :

0100 0001 0010 1101 0000 0000 0000 0000

🔥 Exercice 8 bis : Convertir les nombres réels suivants en binaire simple précision en suivant la norme IEEE

754 :

$$G = -8,125_{10}$$

$$H = 0,375_{10}$$

$$I = 0,1_{10}$$

✓ EXEMPLE 6 ter :

On peut également faire le travail inverse. Pour interpréter un nombre codé en virgule flottante, on utilisera la « formule suivante » : $\text{valeur} = \text{signe} \times \text{mantisse} \times 2^{\text{exposant} - \text{décalage}}$

Par exemple, on peut convertir en décimal le nombre 0011 1110 1000 0000 0000 0000 0000 0000.

Le bit de signe est égal à 0 : signe +

La mantisse est égale à 000 0000 0000 0000 0000 0000, soit 0 si on élimine les zéros « inutiles » à droite.

L'exposant décalé vaut 0111 1101 soit 125. Il faut alors retirer 127 l'exposant est donc égal à - 2.

En base 10, le nombre cherché est donc égal à $1,0 \times 2^{-2} = 1 \times 0,25 = 0,25$



Exercice 8 ter : Convertir le nombre flottant simple précision suivant :

J = 1 10000001101010000000000000000000₂

K = 00111101110011001100110011001100₂

PARTIE 4 : Encodage

Quelle que soit la nature de l'information traitée par un ordinateur (image, son, texte, vidéo), elle est toujours représentée sous la forme d'un ensemble de nombres binaires.

Le codage de l'information permet d'établir une correspondance entre la représentation externe de l'information et sa représentation binaire.

1. Le codage Morse

Un des premiers systèmes de codage est le code Morse :

- Il a été développé par Samuel Morse et ses collaborateurs (1837)
- Il a servi pour la transmission de signaux par télégraphe
- Le système de codage est fait avec des séquences signal court/long

Le code Morse est au départ principalement utilisé par les militaires pour transmettre des informations, souvent chiffrées.

Le signal peut être transporté via un signal radio, des signaux électriques à travers un câble télégraphique ou des signaux visuels (lumière).

L'idée est de coder les caractères fréquents avec des séquences simples et les caractères rares par des séquences compliquées.

Par exemple le mot

Bonjour

Se code par :

· · · · / - - - / - · / · - - - / - - - / · - - / · · · /

Traduire les codes suivants :

· - · / - - - / - · - / · / · /	
· - · / · - · / - - - / · - · / - - / · - / - /	
· · / - - - / · - - / · /	
- - - / · - - / · - · / - - - / · - - / · · / · /	

Code morse international

1. Un tiret est égal à trois points.
2. L'espacement entre deux éléments d'une même lettre est égal à un point.
3. L'espacement entre deux lettres est égal à trois points.
4. L'espacement entre deux mots est égal à sept points.

A · · -	U · · -
B - · · ·	V · · -
C - · - ·	W - · -
D - · -	X - · · -
E ·	Y - · -
F · · - ·	Z - - ·
G - ·	
H · · · ·	
I · ·	
J - · -	
K - · -	
L · - ·	
M - -	
N - ·	
O - - -	
P · - - ·	
Q - · - ·	
R · - ·	
S · · ·	
T -	
	1 · · - - -
	2 · · - -
	3 · · -
	4 · ·
	5 ·
	6 ·
	7 ·
	8 ·
	9 ·
	0 ·

2. Le codage 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 8e bit n'est pas utilisé pour le codage des caractères. 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 1 : Le « A » majuscule est codé par 65₁₀ ou 41₁₆.

🔧 Exercice 1 : Coder en ASCII le mot suivant : Bonjour

	Décimal	Hexadécimal	Binaire
B			
o			
n			
j			
o			
u			
r			

3. Codage 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.

C'est pour répondre à ce problème qu'est née la norme ISO-8859-1. 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 "latin1" 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.

C'est source de grande confusion pour les développeurs de programmes informatiques car un même caractère peut être codé différemment suivant la norme utilisée.

ISO-8859-1																
	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	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTs	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	ı	¢	£	¤	¥	¦	§	¨	©	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	ø	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

 Exercice 2 : Décoder le texte suivant : (SP correspond à un espace)

Ligne	Code ISO 8859-1	
1	4C 61 20 43 69 67 61 6C	
2	65 20 65 74 20 6C 61 20	
3	46 6F 75 72 6D 69 2E 20	

4. Codage 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.

On attribue à chaque caractère un nom, un numéro (appelé point de code) et un bref descriptif qui seront les mêmes quelle que soit la plate-forme informatique ou le logiciel utilisés.

Unicode est uniquement une table qui regroupe tous les caractères existant au 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 : 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 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.

Le codage se fait suivant 3 variantes : UTF-8, UTF-16 et UTF-32.

On retrouve ici le codage 8 bits :

<http://sebastienguillon.com/test/jeux-de-caracteres/windows-ascii-fr.html>

Sous Windows touche alt + n° exemple touches alt+1 = ☺

Sous Word : saisir le code unicode puis touches alt+c exemple symbole euro (Unicode =20AC) = €



MÉTHODE 1 : Codage en UTF-8

— On transforme le code point (numéro nominatif) en binaire

— On code entre 1 et 4 octets selon la table suivante :

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Les « xxx » sont les chiffres de la représentation du code point en base 2.

✓ EXEMPLE 2 : Codage en UTF-8 de la lettre « é ».

• « é » correspond au code point U+00E9 d'après la table. Cela correspond à la deuxième ligne du tableau. On peut donc coder en utilisant deux octets.

• D'après le tableau on a besoin de 11 bits pour le codage de E9.

• $00E9_{16} = 000\ 1110\ 1001_2$

• Les « xxx » dans le tableau correspondent aux bits de la représentation binaire de $E9_{16}$

Finalement « é » se code par :

1100 0011 10 10 1001



Exercice 3 : Coder en UTF-8 le symbole Ø



Exercice 4 : Coder en UTF-8 le symbole →




Exercice 5 : Coder en UTF-8 le symbole ↑

5. Unicode et Python

En python il existe des fonctions capables de transformer les caractères en nombre et les nombres en caractère.

```
>>> chr(65) # Transforme une valeur decimale en caractere
'A'
>>> ord('A') # Transforme un caractere en valeur decimale
65
```

 Exercice 6 : Créer une fonction `creerAlphabet()` qui renvoie toutes les lettres de l'alphabet

```
>>> creerAlphabet()
a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```


6. Cryptographies – le code de César

L'empereur romain Jules César utilisait pour communiquer avec ses espions une méthode de chiffrement de ses messages simple mais efficace : il décalait chaque lettre de l'alphabet de 3 caractères vers la droite.

Les lettres minuscules et les signes typographiques n'existaient pas, et l'alphabet latin n'a pas de caractères accentués. Pour complexifier le décodage, Jules César ne mettait pas d'espaces entre les mots.

Le codage de l'alphabet était le suivant :

```
alphabet clair → A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
alphabet codé → D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

 Exercice 7 : Créer un programme Python `decodageCesar(motCode :str)` qui demande un mot (ou une phrase) codée avec le code de César, et qui vous affiche le résultat décodé.

```
def decodageCesar(motCode:str):
# decodageCesar('ERQMRXU ') donne BONJOUR
```

Astuces :

- Il est possible de parcourir facilement les caractères d'une chaîne de caractère en utilisant la structure suivante :


```
mot = 'abcd'
for lettre in mot:
    print (lettre)
```


7. Codage de Freeman

Le codage de Freeman sert principalement à coder des images ou des formes.

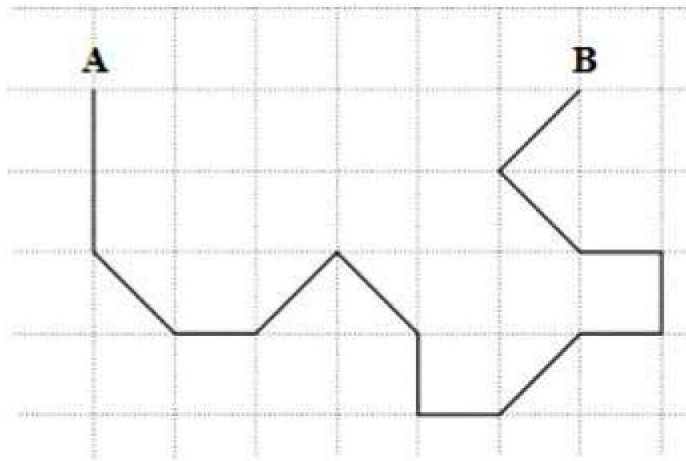
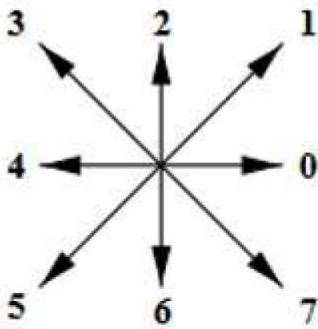
Principe :

- Codage avec un nombre limité de bits
- Constitution d'une chaîne de codes à partir d'un pixel initial


 **MÉTHODE 2 :** On choisit un pixel initial du contour et un sens de parcours. On code la direction qui permet de passer d'un pixel du contour à son voisin immédiat. On continue jusqu'à revenir au pixel initial.

Ce codage permet de coder 8 directions uniquement. C'est donc un système de codage sur 3 bits















✓ **EXEMPLE 3 :** Codage de la figure suivante



Le premier déplacement est vers le bas. Le premier code est donc 6.

 **Exercice 8 :**

A. Compléter la table de codage

Déplacement	Codage	Déplacement	Codage	Déplacement	Codage
	6				
					
					
					
					

B. Donner alors le codage du déplacement

C. Coder ce déplacement en hexadécimal

Il est possible d'utiliser 4 bits pour optimiser le codage. On peut maintenant coder des déplacements de 2 unités.

