Chapitre 2 : Représentation des données

1 représentation des nombres

1.1 Un peu d'histoire

Histoire

Que ce soient des images, des vidéo, des photos, des fichiers de traitement de texte..., toute donnée stockées sur un ordinateur ou transmise vers un ordinateur, ne l'est que par l'intermédiaire de circuits électroniques : les transistors.
Les transistors ne peuvent se trouver que dans deux états : sous tension (on note cela l'état 1) et hors tension (on note cela l'état 0).
En 1970, le premier microprocesseur,Intel 4004, comptait environ 2000 transistors. Aujourd'hui, le microprocesseur Intel Core i7 en compte 2.3 milliard !
Cette unité de stockage est appelée bit, qui peut prendre les valeurs conventionnelles 0 et 1.
Ainsi, toute action informatique est une suite d'opérations sur des paquets de 0 et de 1, regroupés par huit : les octets.
Les processeurs récents sont des processeurs 32 bits ou 64 bits ; ce qui signifie qu'ils disposent de 32 bits ou 64 bits pour stocker un nombre.

1.2 Représentation d'un entier positif en base 2

1.2.1 rappel sur la base 10

Exemple

$135_{10} =$

1.2.2 La base 2

Exemple

$135_{10} =$

Définition 2.1

Soit n un entier naturel.
Il existe un entier p et $a_1, a_2, a_3 ... a_p$ des nombres entiers égaux à 0 ou 1 tels que :
$n = (a_0 a_1 a_2 ... a_p)_2 = a_0 \times 2^p + a_1 \times 2^{p-1} + ... + a_p \times 2^0$.
$(a_0 a_1 a_2 ... a_p)_2$ est la représentation de n en base 2.

Savoir-Faire 2.1
Savoir passer d'un nombre représenté en base 2 à sa valeur en base 10

...	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
...								

Savoir-Faire 2.2
Savoir passer d'un nombre représenté en base 10 à sa représentation en base 2

- On divise par 2 le nombre donné. On note le quotient et le reste qui est 0 ou 1.
- Puis on divise par 2 le quotient, on note le reste.
- On recommence l'étape 2 jusqu'au moment où le quotient est égal à 0.
- On note ensuite les restes obtenus, en commençant par le dernier et en remontant jusqu'au premier.

Combien d'entier peut-on représenter avec un codage en binaire de n chiffre(s) ?

1.2.3 Représentation d'un entier positif en base 16

Écrire en binaire est fastidieux et source d'erreur quand il y a de grandes séries de bits.
On utilise alors le système hexadécimale (base 16). Les nombres sont écrits à l'aide de 16 symboles :0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F avec (A, B, C, D, E, F) valant respectivement en décimal (10,11, 12, 13, 14, 15)
On a donc le tableau suivant :

Base 10	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Base 16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Pour indiquer qu'un nombre est en base 16 on peut utiliser

- l'indice 16 à la fin du nombre : $(AA)_{16}$
- On place \$ devant la nombre : \$AA
- On place les symboles $0x$ devant le nombre : $0xAA$

Exemple

$219_{10} = 11011011_2 = DB_{16} = \$DB = 0xDB$ .....
.....
.....
.....
.....
.....
.....
.....

Exercice 2.1
Combien d'entiers peut-on représenter avec un hexadécimal de 4 chiffres ?

Savoir-Faire 2.3
Savoir passer d'un nombre hexadécimal (représenté en base 16) à sa représentation décimale (en base 10)

...	16^4	16^3	16^2	16^1	16^0
...					

1. $5D_{16} =$
2. $F3C_{16}$

Savoir-Faire 2.4
Savoir passer d'un nombre représenté en base 10 à sa représentation en base 16

1. 978_{10}
2. 184_{10}
3. 252_{10}

☛ Méthode :

- On divise par 16 le nombre donné. On note le quotient et le reste qui est entre 0 et 15.
- Puis on divise par 16 le quotient, on note le reste.
- On recommence l'étape 2 jusqu'au moment où le quotient est égal à 0.
- On note ensuite les restes obtenus (en remplaçant bien sûr 10 par A, 11 par B...et 15 par E), en commençant par le dernier et en remontant jusqu'au premier.

Savoir-Faire 2.5
Savoir passer d'un nombre hexadécimal (représenté en base 16) au nombre binaire (représentation en base 2

1. $F4_{16}$
2. $45E_{16}$
3. $E8E_{16}$

Méthode :

- On convertit en base 2 et sur 4 bits chaque chiffre donné en hexadécimal.
- On réunit ensuite les nombres obtenus pour obtenir la conversion en base 2.

Savoir-Faire 2.6
Savoir passer d'un nombre représenté en base 2 à sa représentation en base 16

1. 01101111_2
2. 10101111110_2
3. 1101011001_2

Méthode :

- On regroupe les bits par 4 (en ajoutant éventuellment des zéros à gauche)
- On convertit en base 16 chaque groupe de 4 bits en binaire

Exercice 2.2
L'adresse MAC $ac :87 :a3 :a8 :c0 :f2 est codée sur combien de bits ?
Convertissez l'adresse MAC en binaire.

1.3 Représentation des entiers relatifs

1.3.1 Introduction

Pour traiter les nombres négatifs, l'ordinateur ne dispose pas du signe moins.
Il faut donc mettre en place une convention pour représenter en binaire des nombres entiers négatifs.
Il existe plusieurs méthodes, et notamment :

- Représentation du bit signé
- le complément à 2
- Représentation en excédent

1.3.2 La représentation des entiers à l'aide du bit signé

Pour coder un entier relatif, une méthode simple consiste à utiliser le bit de poids fort pour indiquer le signe :

- 0 pour un nombre positif
- 1 pour un nombre négatif

Exemples

12_{10} = ...

-12_{10} = ...

Remarque

Il se pose deux problèmes :

- Il y a deux zéros : Sur 4 bits, 0000 et 1000
- Cette représentation induit de la complexité pour des opérations simples (addition de deux nombres de signes différents par exemple)
- Essayez de faire $2 + (-2)$ pour en avoir le coeur net :-)

1.3.3 Le complément à deux

Comme précédemment, on convient que :

- Bit de signe à 0 pour un entier positif
- Bit de signe à 1 pour un entier négatif

Exemple sur 8 bits :

s								valeur décimale	compl à 2^7
1	1	1	1	1	1	1	1	-1	128-127
1	1	1	1	1	1	1	0	-2	128 -126
1	1	1	1	1	1	0	1	-3	128 -125
1				...					
1	1	1	1	0	1	0	0	-12	128-116
1				...					
1	0	0	0	0	0	0	1	-127	128-1
1	0	0	0	0	0	0	0	-128	128-0
0	1	1	1	1	1	1	1	127	
0	1	1	1	1	1	1	0	126	
0				...					
0	0	0	0	0	0	1	0	2	
0	0	0	0	0	0	0	1	1	
0	0	0	0	0	0	0	0	0	

Le complément à 2 d'un nombre binaire s'obtient de la façon suivante :
Voici un exemple pour -12 :

Poids en binaire	128	64	32	16	8	4	2	1
Valeur absolue en binaire								
Inversion des bits								
On ajoute 1								

Donc -12 est représenté par 11110100 sur 8 bits, en complément à 2.

Exercice 2.3
Trouver le complément à 2 de -59 avec une mémoire de 8 bits.
Faire l'addition binaire $59 + (-59)$

Savoir-Faire 2.7

Savoir passer d'un entier relatif à sa représentation en binaire en complément à 2.

- Représenter -8_{10} sur 8 bits, en complément à 2.
- Représenter -121_{10} sur 8 bits, en complément à 2.
- Représenter -100_{10} sur 8 bits, en complément à 2.
- Représenter -4_{10} sur 3 bits, en complément à 2.

Exercice 2.4
Trouver le complément à 2 de -59 avec une mémoire de 8 bits.
Faire l'addition binaire $59 + (-59)$

Savoir-Faire 2.8

Savoir passer d'un entier binaire en complément à 2 à sa représentation en base 10.
Trouvez la représentation décimale des entiers relatifs dont la représentation binaire sur 8 bits est 0000 0000, 1000 0000, 0111 1111 et 1001 1001.

Savoir-Faire 2.9

Savoir effectuer des opérations avec les entiers relatifs

On considère les opérations suivantes :

- $49 + 25$
- $35 + 65$
- $45 - 12$
- $67 - 13$
- $29 - 84$

Pour chaque calcul :

1. Effectuer le calcul directement avec les représentations décimales (trop facile :-))
2. Représenter chaque opérande en binaire sur 8 bits, en complément à deux.
3. Effectuer l'opération binaire
4. Représenter le résultat de l'opération en base 10 et comparer avec le résultat attendu.

1.4 Représentation d'un réel en base 2

1.4.1 Virgule fixe

Le codage des nombres à virgules fixe nous permettra ensuite d'étudier le codage avec virgule flottante.

Dans le système décimal, écrire 56.375 signifie :
$56.375 = 5 \times 10^1 + 6 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}$
La nouveauté est ici la présence des puissances de 10 négatives pour les chiffres après la virgule.
Il en est de même en binaire, avec des puissances de 2 :

Exemple

On désire coder en virgule fixe le nombre réel 56.375.

1. Commencer par coder la partie entière : 56

2^6	2^5	2^4	2^3	2^2	2^1	2^0
$= 64$	$= 32$	$= 16$	$= 8$	$= 4$	$= 2$	$= 1$
0	1	1	1	0	0	0

   56 se décompose de façon unique $56_{10} = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
   Donc $56_{10} = 111000_2$

2. En ce qui concerne la partie fractionnaire : 0.375
   Le fonctionnement est identique, mais avec des exposants de 2 négatifs :

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
$= 0.5$	$= 0.25$	$= 0.125$	$= 0.0625$	$= 0.03125$	$= 0.015625$
0	1	1			

   0.375 se décompose de façon unique $0.375 = 0 \times 0.5 + 1 \times 0.25 + 1 \times 0.125$
   Donc $0.375 = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$
   Donc $0.375_{10} = 0.011_2$

☞ **Attention :** Seconde méthode pour la partie fractionnaire :

- $0.375 \times 2 = 0.75 = 0(partie\ entière) + 0.75(partie\ fractionnaire)$
- $0.75 \times 2 = 1.5 = \quad 1(partie\ entière) + 0.5$
- $0.5 \times 2 = 1 = \quad 1(partie\ entière) + 0(partie\ fractionnaire)$

Finalement, on retrouve bien $0.375_{10} = 0.011_2$

3. On peut ensuite conclure :
$56.375_{10} = 111000.011_2$

Savoir-Faire 2.10

coder en virgule fixe les nombres suivants :

- 123.6875_{10}
- 14.5_{10}
- 435_{10}
- 171.78515625_{10}

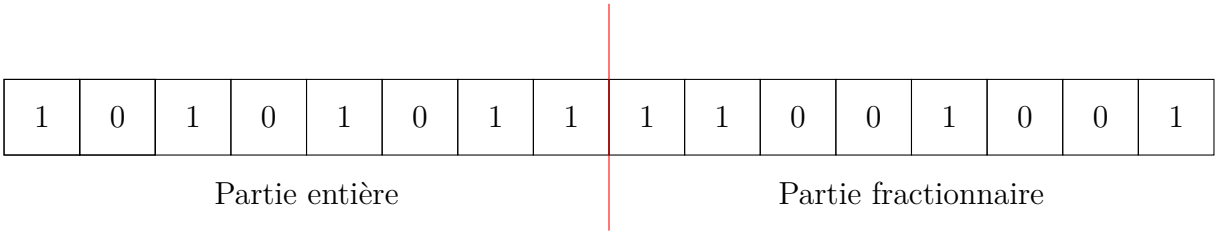
Savoir-Faire 2.11

Décoder maintenant les nombres suivants (virgule fixe)en nombres décimaux :

- 11.101_2
- 100111.101_2
- 1001.111_2

1.4.2 Virgule flottante

Si on reprend le dernier exemple $171,78515625_{10}$ est codé en 10101011.11001001_2, c'est à dire en la succession des bits suivants :



Le trait rouge représente la virgule. La position de cette virgule est à entrer dans le préambule. Ici , on a un codage en virgule fixe "8 × 8"

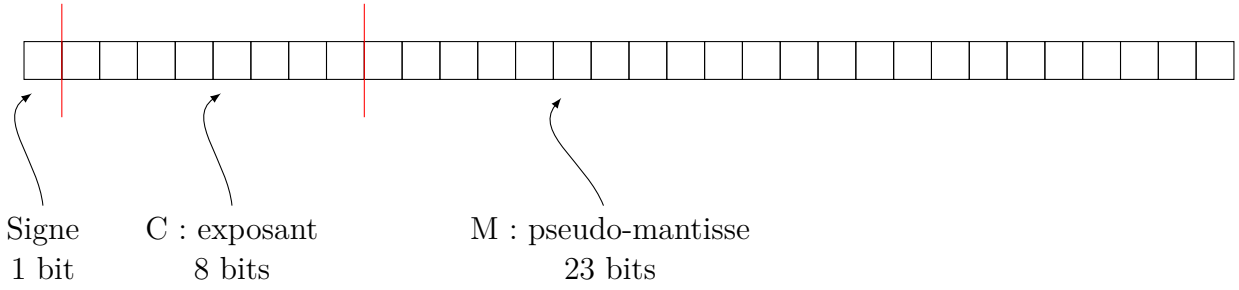
- **Avantages :** L'addition de deux nombres en virgules flottante est très facile et rapide en nombre de calculs : cela se passe comme pour l'addition de deux entiers binaires, en ajoutant juste au "bon endroit" la virgule.
- **Inconvénients :**
  - Le nombre de bits nécessaires pour coder le nombre n'est pas fixe, et dépend du nombre à coder.

- Il faut connaître à l'avance (lors de l'écriture du programme) l'ordre de grandeur des nombres à manipuler afin de positionner au mieux la virgule ; une fois la position de la virgule choisi, on ne peut plus changer. Ceci est un inconvénient majeur : ce type de connaissance a priori existe pour certaines applications, mais dans le cas général ce n'est pas le cas.

☞ Pour pallier à ce manque de flexibilité, le concept de virgule flottante est nécessaire !

Il existe différente façon de coder un nombre en virgule flottante.

L'une d'elle propose ce format (format IEEE754 simple précision) :



Principe :

- Le signe est codé 1 lorsque le nombre est négatif, 0 sinon.

- Il faut ensuite mettre le nombre est sous la forme $\pm 1.M \times 2^C$:
  - On commence par coder le nombre souhaité en utilisant la méthode de la virgule fixe
  - On représente ensuite ce nombre de la forme $1.M \times 2^C$
    - C correspond à l'exposant codé en excédent à 127 (sur 8 bits).
    - M correspond à la pseudo-mantisse (pseudo, car le '1.' n'est pas codé, c'est toujours 1 et on fait donc l'économie d'un bit)(sur 23 bits)

Exemple

On veut ici coder en virgule flottante simple précision le nombre $171,78515625_{10}$.

- Le premier bit est 0 (signe positif)
- On a vu que $171,78515625_{10} = 10101011.11001001_2$.
  Exprimons ce nombre sous la forme $1.M \times 2^C$:
  $10101011.11001001 = 1.1110110100110011 \times 2^7$ (décalage de7 chiffres vers la gauche)
- On a donc $C = 7 + 127$(excédent)$= 134$ donc $C = 10000110_2$.
- On trouve ensuite M, en enlevant $'1.'$ à 1.1110110100110011 : $M = 1110110100110011$.
  On complète ensuite avec des zéros pour arriver à 23 bits : $M = 11101101001100110000000$

- On a donc : $171,78515625_{10} = 01000011001010111100100100000000_2$

Savoir-Faire 2.12

Savoir coder un nombre décimal en utilisant la virgule flottante simple précision.

Coder en virgule flottante simple précision les nombres suivants :

- 123.6875_{10}
- 14.5_{10}
- 435_{10}
- $171,78515625_{10}$
- 0.25
- 0.1 (plus difficile)
- $\frac{1}{3}$ (plus difficile)

Savoir-Faire 2.13

Savoir décoder en nombre décimal un nombre écrit en virgule flottante simple précision.

Décoder en virgule flottante simple précision les nombres suivants : Décode en nombre décimal les nombres binaires suivants, exprimés avec une virgule flottante simple précision :

- $01000000001011100001010001111010_2$
- $01000011000100100001000000000000_2$
- $11000101101100011011001100000000_2$
- $00111000111100001011110010111110_2$

2 Représentation des caractères

On parle ici de codage des caractères, c'est-à-dire qu'il faut disposer d'un système de code permettant de traduire des caractères (lettres) en nombres.

2.1 Code ASCII

American Standard Code For Information Interchange –> sur 7 bits, donc 2^7 représentations possibles

Tableau de conversion ASCII

				b6	0	0	0	0	1	1	1	1
				b5	0	0	1	1	0	0	1	1
				b4	0	1	0	1	0	1	0	1
b3	b2	b1	b0	Hex	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	C	FF	FS	,	<	L	\	l	|
1	1	0	1	D	CR	GS	-	=	M	]	m	}
1	1	1	0	E	SO	RS	.	>	N	^	n	~
1	1	1	1	F	S1	US	/	?	O	-	o	DEL

A vous de jouer ! 2.1

Décoder le message suivant :
1000010 1110010 1100001 1010110 1101111 0100001

A vous de jouer ! 2.2

Coder le message "Hello JB"

1. en binaire
2. En hexadécimal

2.2 code ASCII étendu

Le codage ASCII est souvent complété par des correspondances supplémentaires afin de permettre l'encodage informatique d'autres caractères, comme les caractères accentués par exemple. Cette norme s'appelle ISO-8859 et se décline par exemple en ISO-8859-1 lorsqu'elle étend l'ASCII avec les caractères accentués d'Europe occidentale –> sur 8 bits donc 2^8 représentations possibles.

2.3 La norme Unicode

L'Unicode désigne un système de codage utilisé pour l'échange de contenus à l'échelle internationale, dans différentes langues, en faisant fi de la langue, de la plateforme et du logiciel utilisé pour cet échange.
l'Unicode permet de coder l'ensemble des caractères couramment utilisés dans les différentes langues de la planète en spécifiant un nombre unique pour chacun de ces caractères.
Quelques langages ont un très grand nombre de caractères comme le chinois, le japonais ou le

coréen. Il a donc fallu utiliser au moins deux octets au lieu d'un pour les représenter. Avec deux octets, on dispose de 256*256 = 65536 codes.

Le consortium Unicode a pour devise **"un seul nombre pour chaque caractère, quelque soit la plate-forme, le programme ou le langage"**.

Chaque symbole d'écriture est représenté par une valeur hexadécimale préfixée par « U+ ». Consulter la norme unicode

Contrôles C1 et supplément latin-1 [modifier | modifier le code]

PDF : en [archive] v · d · m	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+0080	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
U+0090	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
U+00A0	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	- SHY	®	¯
U+00B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
U+00C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
U+00D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
U+00E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
U+00F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Exemple de table unicode

2.4 L'UTF-8

Pour résoudre le problème du doublement (ou triplement) de la taille des fichiers, Ken Thompson, un des pères d'UNIX, a littéralement inventé le 2 septembre 1992 le codage UTF-8, une autre manière de coder les caractères Unicode. UTF-8 code chaque caractère sur 8, 16, 24 ou 32 bits L'idée a été de coder les caractères les plus utilisés de 0 à 127 sur un octet, de coder les caractères de 128 à 1023 sur 2 octets et au-dessus de 1023 sur 3 octets et ce jusqu'à 4 octets. il est devenu le standard de l'Internet, donc de l'informatique. Par exemple, l'UTF-8 est devenu le codage par défaut des documents en XML donc en XHTML.
⚠ Il ne faut pas confondre Unicode qui identifie des caractères par des nombres et l'encodage utf8 qui va associer à (la plupart) des caractères Unicode une suite d'octets. D'autres encodages des caractères Unicode comme utf-8 par exemple donneront en général une autre suite d'octets.

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

Principe de l'UTF-8

Techniquement, il s'agit de coder les caractères Unicode sous forme de séquences de un à quatre codets d'un octet chacun. La norme Unicode définit entre autres un ensemble (ou répertoire) de caractères. Chaque caractère est repéré dans cet ensemble par un index entier aussi appelé « point de code ». Par exemple le caractère « €» (euro) est le 8365e caractère du répertoire Unicode, son index, ou point de code, est donc 8364 (0x20AC) (on commence à compter à partir de 0).
Le codage bianire du symbole € est donc 0010000010101100 (16 bits). On utilise donc la trame 1110xxxx 10xxxxxx 10 xxxxxx
Ainsi, le symbole € est codé en UTF-8 par 111000101000001010101100

Caractère	Unicode (hexadécimal)	Valeur scalaire		Codage UTF-8	
		décimal	binaire	binaire	hexadécimal
[NUL]	U+0000	0	0000000	00000000	00
[US]	U+001F	31	0011111	00011111	1F
[SP]	U+0020	32	0100000	00100000	20
A	U+0041	65	1000001	01000001	41
é	U+00E9	233	00011 101001	11000011 10101001	C3 A9
€	U+20AC	8 364	0010 000010 101100	11100010 10000010 10101100	E2 82 AC

Exemples de codage UTF-8

Remarque

En html, on utilise le codage utf-8 pour permettre au navigateur de lire correctement le document. On insère dans le "head" :

```
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
```

2.5 Python et les codes ASCCI

Python intègre nativement des méthodes et fonctions permettant de convertir un caractère en représentation ASCII décimale (entre 0 et 127) et vice-versa :
La fonction ord() donne la valeur décimale ASCII d'un caractère :

`ord('B') affiche 66`

chr() renvoie le caractère de valeur décimale ASCII donnée :

`chr(65) affiche 'A'`

Exercice 2.5
Écrire un script Python qui affiche les 256 caractères de la table ASCII étendue. De quel type sont ces caractères ?

Exercice 2.6
Soit la liste suivante : [233, 112, 97, 116, 97, 110, 116, 32, 33], qui correspond à la liste des codes Unicode d'une chaîne. Laquelle ?

Vous écrirez une fonction python decodage(lst) :

```
def decodage(lst):
    '''lst est une liste d'entier contenant les codes
    La fonction renvoie une chaine de caractères
    qui correspond au décodage.
    '''
```

Unicode et encoding :

- unicode : chaque caractère est codé par un codepoint représenté par un nombre hexadécimal, par exemple U+00E8
- encoding : convertit l'unicode logique en stockage physique sur des octets, avec différentes solutions. Un de ces codages est UTF-8 (le plus standard et utilisé)
- en python3, les strings (type str) sont en unicode

La méthode encode() permet d'encoder un caractère en utf-8 :

```
>>> mot_code = 'hello'.encode()
>>> mot_code
>>> b'hello'
>>> mot_code[0]
104
```

...et 104 correspond bien au caractère h en unicode !

A l'inverse, la méthode decode() permet de décoder une suite d'octet en utf-8.

```
>>> mot_code = 'é'.encode()
>>> mot_code
b'\xc3\xa9'
```

Ici, on n'est plus avec un caractère ASCII donc il faut retrouver le caractère qui a pour encodage utf-8 c3a9 en binaire.
⚠ En Python 3, le type bytes est un tableau d'entiers

A vous de jouer ! 2.3

On considere :
⚠ Le début de l'exercice se fait sans l'utilisation de la console Python

```
>>> mot = b'\xf0\x9f\x98\xb1'
```

- Déterminer son encodage utf8 binaire
- En déduire sa valeur unicode binaire
- En déduire sa valeur unicode décimale
- Faire une recherche sur le Web pour trouver ce caractère
- Vérifier avec python

3 Représentation des booléens

Histoire

Georges Boole sort ' Mathematical Analysis of Logic' en 1847, qui est une structure algébrique a seulement deux états : 0 et 1. A cette époque, tout cela reste théorique mais sera grandement utilisé par la suite. Bien que le Binaire soit apparu en 3000 avant J.C. en Chine pour calculer des périodes religieuses, les Français ne l'utiliseront qu'en 1600 avec la table de Harriot(mathématicien et astronome anglais) et Leibniz y consacra un article important en 1703.Il fallut attendre 1930 pour que Claude Shannon démontre qu'avec des interupteurs fermés pour 'vrai' et ouvert pour 'faux', on peut effectuer des opération logiques en associant le nombre 1 pour vrai et 0 pour faux. C'est lui qui va populariser le mot 'bit' créé par John Tukey.
En 1937, Georges Stibitz et Samuel Williams fabriquent un calculateur avec des relais, basé sur le système binaire, capable de réaliser les 4 opérations avec des nombres décimaux en entrée en 1 minute ! Ils ont même établi un système de communication pour piloter une machine à des centaines de kilomètres. D'autres calculateurs avaient été fabriquées bien avant mais étaient mécaniques et utilisaient des engrenages. La Pascaline de Blaise Pascal, à 16 ans, est capable de soustraire et additionner. Gottfried Wilhelm Leibniz conçoit en 1694 une machine capable d'en plus multiplier et diviser. D'autres modèles mécaniques suivirent, le dernier fut une calculatrice mécanique de 1948 capable de faire une multitude de calculs mathématiques rapidement. Elle sera utilisée jusqu'en 1970, année de mise en marché de la calculatrice dite conventionnelle que l'on connaît aujourd'hui.

C'est l'arrivée du transistor en 1947 qui va bouleverser les travaux sur les calculateurs. Jusqu'alors on utilisait les relais ou les lampes. Les lampes étaient 1000 fois plus grandes et consommaient 1000 fois plus.

3.1 Qu'est ce qu'un transistor ?

Que fait un transistor ? Un transistor a trois broches : le collecteur, la base et l'emetteur. Le courant dans l'emetteur est égal à la somme des courants du collecteur et de la base. Si la base n'est pas alimentée, le courant ne passe pas et une faible tension dans la base laissera passer le courant...passe, passe pas,....1,0..

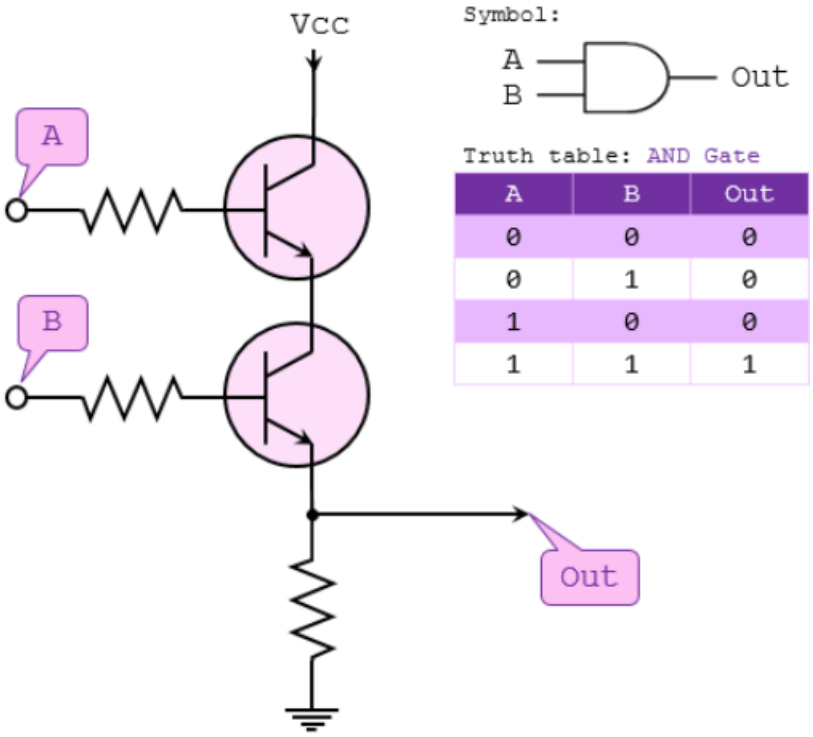
Tansistor

3.2 Fonctionnement

On a un comportement très particulier :

- en appliquant une petite tension à la base, le courant principal entre le collecteur et l'émetteur peut passer
- en supprimant cette tension à la base, le courant entre le collecteur et l'émetteur est coupé.

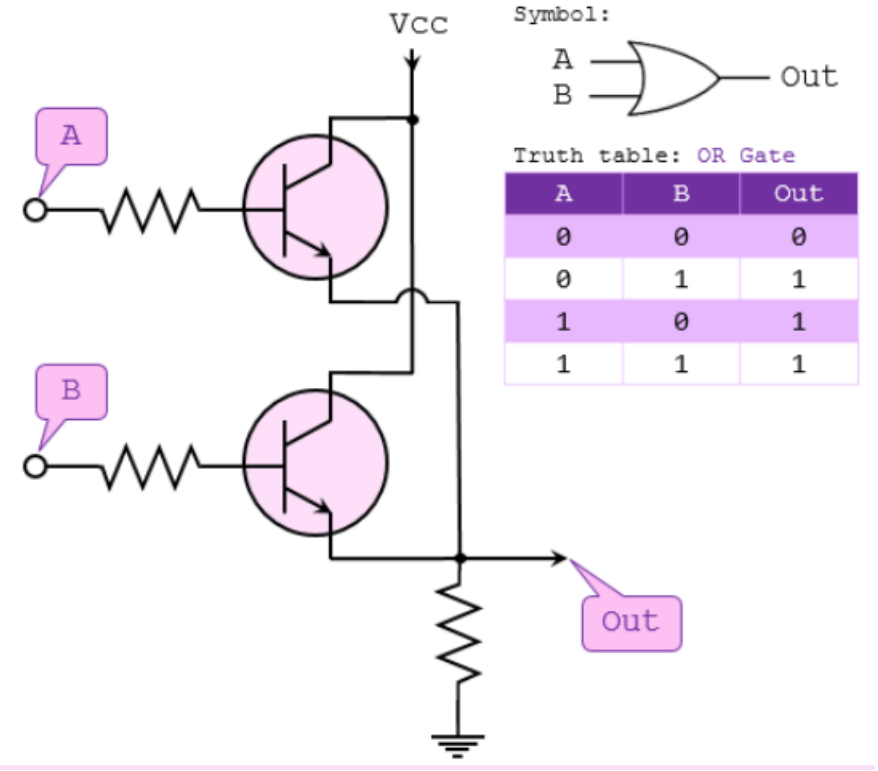
3.3 Porte AND



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

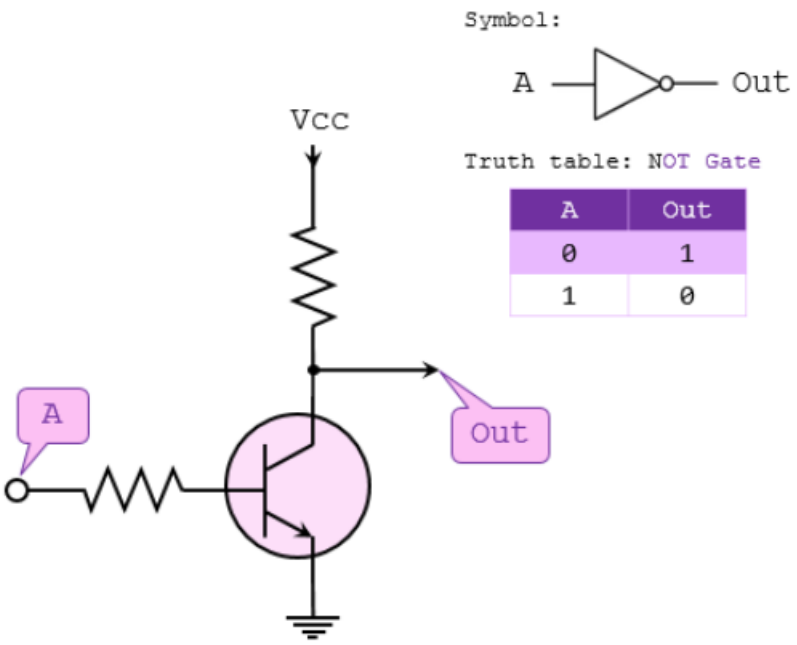
Porte AND

3.4 Porte OR



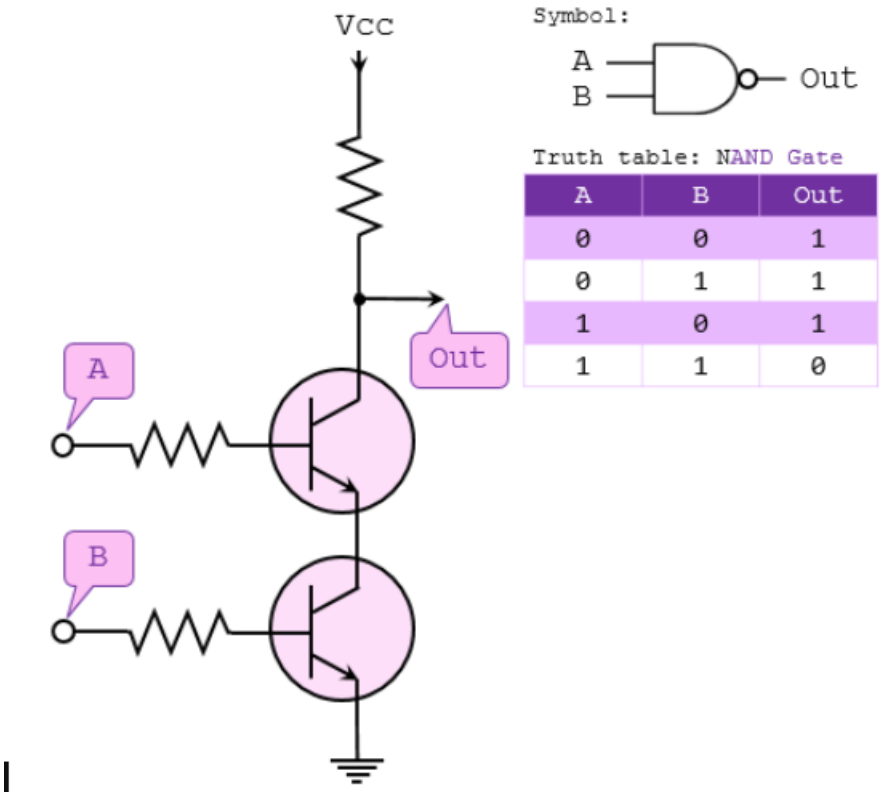
Porte OR

3.5 Porte NOT



Porte OR

3.6 Porte OR



Porte NOT AND

3.7 Porte XOR

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

3.8 Loi de Moore

La multiplication du nombre de transistors a suivi loi de Moore : "le nombre de transistors par circuit de même taille allait doubler, à prix constant, tous les ans. Il rectifia par la suite en portant à dix-huit mois le rythme de doublement. Il en déduisit que la puissance des ordinateurs allait croître de manière exponentielle, et ce pour des années." Depuis 2014 la loi de Moore n'est plus complétement vérifiée et subit un ralentissement.

Exercice 2.7
Faire un tableau à 5 lignes. Placer a et b dans les deux premieres colonnes et y inscrire toutes les combinaisons possibles de 0 et 1 pour a et b dans les 4 lignes qui suivent. Ajouter des colonnes pour ecrire not(a) and b, not(a) or b, not(a) or not(b), not(a and b). Comparer les deux dernières colonnes. Que concluez vous ?

Exercice 2.8
Faire de même avec (a or b) and (a and b), et (a or b) or (a and b). Comment peut-on écrire ces deux règles plus simplement ?

Exercice 2.9
Soit, liste1=[3,5,7] et liste2=[10,3,18,25].
Donner les valeurs de test1, test2 et test3 et test 4, avec :
test1= 3 in liste1 and 3 in liste2
test2= 10 in liste1 or 10 liste2
test3= 10 in liste1 and 10 in liste2
test4= 20 in liste1 and 20 in liste2

Exercice 2.10
Faire un tableau pour (not(a) and b) or (a and not(b))

Exercice 2.11
En python, le "AND" se note "&", le "OR" se note "|", le "XOR" se note "".
Réaliser ce genre de fonctions :

```
>>> tableET()
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
```

Table AND

Exercice 2.12
Cet exercice montre une des applications : le filtrage.
adresse IP :
11000000.10101000.00000001.00000010
masque de sous réseau :
11111111.11111111.11111111.00000000
L'adresse du réseau est obtenu en utilisant l'opérateur & pour chaque bit :
adresse du réseau :
11000000.10101000.00000001.00000000
Retrouver ce résultat avec Python, en urilisant les opérateur & et |.

```
0b11000000101010000000000100000010
0b11111111111111111111111100000000
0b11000000101010000000000100000000
```

Le résultat attendu

Exercice 2.13
Réaliser une fonction jeu() qui :

- Pose à l'utilisateur le résultat d'un tableau logique parmi AND, OR et XOR.
- L'utilisateur saisit sa réponse
- Un message est renvoyé pour dire à l'utilisateur si la réponse est correcte.

```
>>> jeu()
 1 & 0 =
entrer votre réponse :0
 bravo
>>> jeu()
 0 ^ 1 =
entrer votre réponse :1
 bravo
>>> jeu()
 0 | 1 =
entrer votre réponse :1
 bravo
```

FIGURE 1 – Exemple