Contenus	Capacités attendues
Écriture d'un entier positif dans une base $b \geqslant 2$	Passer de la représentation d'une base dans une autre.
Représentation binaire d'un entier relatif	Évaluer le nombre de bits nécessaires à l'écriture en base 2 d'un entier, de la somme ou du produit de deux nombres entiers. Utiliser le complément à 2.
Représentation approximative des nombres réels : notion de nombre flottant	Calculer sur quelques exemples la représentation de nombres réels : 0.1, 0.25 ou 1/3.
Valeurs booléennes : 0, 1. Opérateurs booléens : and, or, not. Expressions booléennes	Dresser la table d'une expression booléenne.
Représentation d'un texte en machine. Exemples des encodages ASCII, ISO-8859-1, Unicode	Identifier l'intérêt des différents systèmes d'encodage. Convertir un fichier texte dans différents formats d'encodage.

Types et valeurs de base

1. Représentation d'un entier positif

1.1. Le décimal

Le système décimal est celui dans lequel nous avons le plus l'habitude d'écrire. Chaque chiffre peut avoir 10 valeurs différentes : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. De ce fait, le système décimal a pour **base 10**. Tout nombre écrit dans le système décimal vérifie la relation suivante :

$745 = 7 \times 100 + 4 \times 10 + 5 \times 1$

$745 = 7 \times 10 \times 10 + 4 \times 10 + 5 \times 1$

$745 = 7 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$

Chaque chiffre du nombre est à multiplier par une puissance de 10 : c'est ce que l'on nomme le **poids du chiffre.**

L'exposant de cette puissance est nul pour le chiffre situé le plus à droite et s'accroît d'une unité pour chaque passage à un chiffre vers la gauche.

$12\,435 = 1 \times 10^4 + 2 \times 10^3 + 4 \times 10^2 + 3 \times 10^1 + 5 \times 10^0$.

Cette façon d'écrire les nombres est appelée **système de numération de position.** Dans notre système conventionnel, nous utilisons les puissances de 10 pour pondérer la valeur des chiffres selon leur position, cependant il est possible d'imaginer d'autres systèmes de nombres ayant comme base un nombre entier différent.

A vous de jouer : Décomposer le nombre 1 045 612 en puissance de 10.

1.2. Le binaire

Dans le système binaire, chaque chiffre peut avoir 2 valeurs différentes : 0, 1 (**appelé « bit »**). De ce fait, le système a pour base 2. Tout nombre écrit dans ce système vérifie la relation suivante :

$(10\ 110)_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
$(10\ 110)_2 = 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1$
donc $\mathbf{(10110)_2 = (22)_{10}}$

Pour un nombre binaire composé donc de bits on appelle « **bit de poids fort** » **le bit le plus à gauche** du nombre binaire, soit le bit avec la plus grande valeur décimale et « **bit de poids faible** » **le bit le plus à droite** du nombre binaire, soit le bit avec la plus petite valeur décimale.

Voici le tableau des « 2^n » :

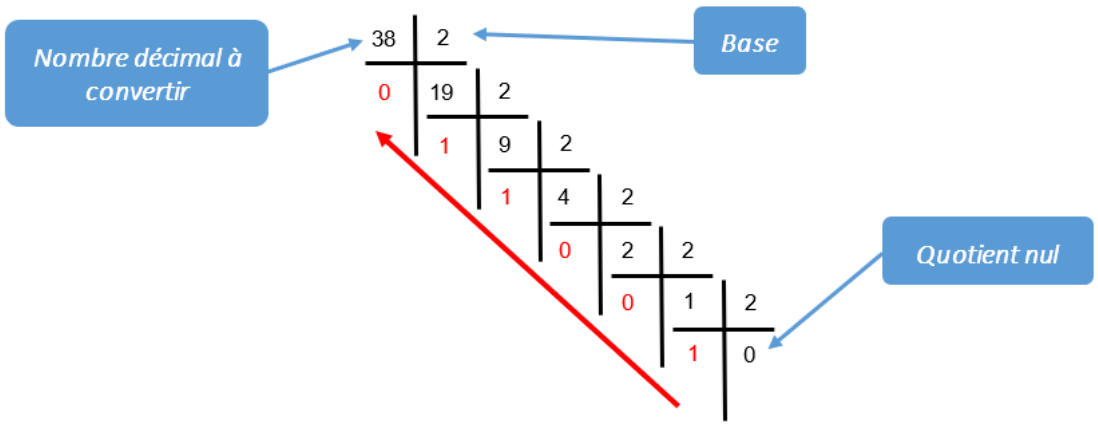
2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}
1	2	4	8	16	32	64	128	256	512	1024	2048	4096

1.3. Conversion décimal-binaire

Pour convertir un nombre décimal en un nombre binaire, on peut utiliser la méthode de la division successive. Elle consiste à effectuer une division euclidienne par 2 (base binaire = base 2) du nombre décimal, garder le reste et recommencer l'opération en divisant le quotient obtenu jusqu'à ce que ce dernier soit nul.

Les restes de ces divisions composent le nombre binaire. Le bit de poids faible (LSB) étant le premier reste et le bit de poids fort (MSB) étant le dernier reste.

Exemple pour la conversion du nombre décimal 38 :



$$(38)_{10} = (100110)_2$$

A vous de jouer : Convertir $(66)_{10}$, $(136)_{10}$ et $(301)_{10}$ en binaire.

1.4. Conversion binaire-décimal

Pour convertir un nombre binaire en décimal on effectue la décomposition du nombre binaire avec les valeurs de 2^n.

Exemple pour la conversion du nombre binaire 100110 :

$$1\times 2^5+0\times 2^4+0\times 2^3+1\times 2^2+1\times 2^1+0\times 2^0=32+4+2=38$$

$$(100110)_2=(38)_{10}$$

A vous de jouer : Convertir $(1010)_2$, $(110\ 1100)_2$ et $(1\ 1101\ 0011)_2$ en décimal.

1.5. L'hexadécimal

Le système hexadécimal utilise les 16 symboles suivants : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. De ce fait, le système a pour **base 16**.
Un nombre exprimé en base 16 pourra se présenter de la manière suivante : $(5AF)_{16}$.

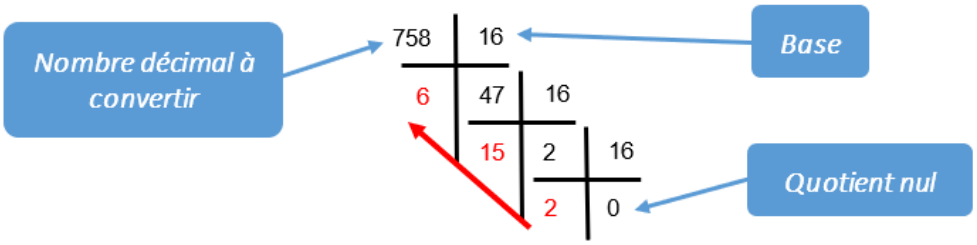
Le nombre $(5AF)_{16}$ peut se décomposer comme suit :
$(5AF)_{16} = 5 \times 16^2 + A \times 16^1 + F \times 16^0$

En remplaçant A et F par leur équivalent en base 10, on obtient :
$(5AF)_{16} = 5 \times 16^2 + 10 \times 16^1 + 15 \times 16^0$
$(5AF)_{16} = 5 \times 256 + 10 \times 16 + 15 \times 1$
donc **$(5AF)_{16} = (1455)_{10}$**

1.6. Conversion décimal-hexadécimal

La conversion d'un nombre décimal en un nombre hexadécimal s'effectue de la même manière que la conversion en un nombre binaire. Il suffit de remplacer la base 2 (base binaire) en base 16 (base hexadécimale).

Exemple pour la conversion du nombre décimal 758 :



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

$$(758)_{10}=(2F6)_{16}$$

A vous de jouer : Convertir $(75)_{10}$, $(525)_{10}$ et $(5\ 456)_{10}$ en hexadécimal.

1.7. Conversion hexadécimal-décimal

Pour convertir un nombre hexadécimal en décimal on effectue la décomposition du nombre hexadécimal avec les valeurs de 16^n.

Exemple pour la conversion du nombre hexadécimal 2CA4 :

$$2\times16^3+12\times16^2+10\times16^1+4\times16^0=11428$$

$$(2CA4)_{16}=(11428)_{10}$$

A vous de jouer : Convertir $(2A)_{16}$, $(100)_{16}$ et $(5DF)_{16}$ en décimal.

1.8. Conversion binaire-hexadécimal

Pour coder un caractère hexadécimal il suffit de 4 bits binaires.

Base 10	Base 16	Base 2
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Pour convertir un nombre binaire en hexadécimal il suffit donc de regrouper les bits par 4 et de convertir chaque groupe de 4 bits en hexadécimal.

Exemple pour la conversion du nombre binaire 101010 :

$(10)_2 = (2)_{16}$ et $(1010)_2 = (A)_{16}$ donc $\mathbf{(101010)_2 = (2A)_{16}}$

A vous de jouer : Convertir $(1110)_2$, $(111\ 0101)_2$ et $(1110\ 1011\ 0001)_2$ en hexadécimal.

1.9. Conversion hexadécimal-binaire

Pour convertir un nombre hexadécimal en binaire on utilise donc la même technique. Chaque caractère hexadécimal correspond à 4 bits binaires.

Exemple pour la conversion du nombre hexadécimal B6 :

$(B)_{16} = (1011)_2$ et $(6)_{16} = (0110)_2$ donc $\mathbf{(B6)_{16} = (10110110)_2}$

A vous de jouer : Convertir $(2A)_{16}$, $(100)_{16}$ et $(5DF)_{16}$ en décimal.

1.10. Calcul du nombre de bit nécessaire pour représenter un entier positif

Pour déterminer le nombre de bits permettant de coder un nombre décimal il existe une formule simple.

2^n (« 2 puissance n ») détermine le nombre de combinaison possible avec n bits. Par exemple avec 6 bits on a au total 2^6 soit 64 combinaisons possibles. On peut donc coder les nombres décimaux allant de 0 à 63.

Si l'on cherche maintenant à déterminer combien il faut de bits pour coder le nombre décimal 102 513 il faut poser l'équation suivante :

$$2^n = 102\ 513$$

Pour la résoudre on utilise le logarithme népérien (noté ln, fonction que vous verrez en Maths).

$$\ln(2^n) = \ln(102\ 513)$$

$$n.\ln(2) = \ln(102\ 513)$$

$$n = \ln(102\ 513)/\ln(2)$$

$$n = 16{,}64\ldots$$

Donc il faut 17 bits pour pouvoir coder le nombre 102 513.

1.11. La taille d'un fichier numérique

Un fichier numérique est codé en binaire. La taille de ces fichiers se mesure donc en bits. Une autre unité est utilisée : l'octet. L'octet correspond à 8 bits.

Pour simplifier la lecture on utilise des préfixes d'unités :

Préfixe	Exa (E)	Péta (P)	Téra (T)	Giga (G)	Méga (M)	kilo (k)
Coefficient multiplicateur	10^{18}	10^{15}	10^{12}	10^9	10^6	10^3

Exemple :

1 Mo = 1000 ko

1000 ko = 8000 kb

Attention, en anglais « octets » se traduit par « bytes » et « bits » par « bits ».

32 Méga-octets s'abrège en anglais par 32 MB (32 Mégabytes) et 32 Méga-bits par 32 Mb.

1.12. Exercices

1) Convertir $(156)_{10}$ en binaire.
2) Convertir $(10100110)_2$ en décimal.

3) Convertir $(2340)_{10}$ en binaire.
4) Convertir $(1100111010101100)_2$ en décimal.
5) Convertir $(120)_{10}$ en hexadécimal.
6) Convertir $(AF)_{16}$ en décimal.
7) Convertir $(3092)_{10}$ en hexadécimal.
8) Convertir $(B35F)_{16}$ en décimal.
9) Convertir $(11101100)_2$ en hexadécimal.
10) Convertir $(7C)_{16}$ en binaire.
11) Convertir $(110100000110101)_2$ en hexadécimal.
12) Convertir $(6ABC)_{16}$ en binaire.
13) Convertir 2560 Mb en Gb.
14) Convertir 0,65 Go en Mo.
15) Convertir 156 Gb en Go.
16) Convertir 2,5 Mo en Mb.
17) Convertir 33 To en Gb.
18) Convertir 2530 Mb en Go.
19) Convertir 0,56 Mo en kb.
20) Convertir 9 532 000 Mb en To.

2. Représentation d'un entier relatif et opération

Comme pour les nombres en base 10 il est possible de représenter les nombres binaires négatifs et d'effectuer des opérations avec.

2.1. L'addition

Pour additionner deux nombres binaires on utilise la méthode que pour deux nombres en base 10.

En binaire : 0 + 0 = 0 0 + 1 = 1 1 + 0 = 1 1 + 1 = 10

Exemple :

```
  1 1     retenues
  1 0 1
+ 1 1 1
-----------
1 1 0 0
```

A vous de jouer : Faire l'addition binaire de $(1011)_2$ avec $(101)_2$ et $(10\ 0101)_2$ avec $(1101)_2$.

2.2. Le produit

Pour la multiplication de deux nombres binaires on procède aussi de la même manière que pour la multiplication de deux nombres en base 10.

En binaire : 0 x 0 = 0 0 x 1 = 0 1 x 0 = 0 1 x 1 = 1

Exemple :

```
   1010
 × 1011
------
   1010
  1010
 0000
1010
------
1101110
```

A vous de jouer : Faire la multiplication binaire de $(1011)_2$ avec $(101)_2$ et $(10\ 0101)_2$ avec $(1101)_2$.

2.3. Le complément à deux

Pour représenter un nombre binaire négatif, la méthode la plus élémentaire, **jamais utilisée en pratique**, consiste à utiliser l'un des bits (le premier, donc le bit de poids fort) pour représenter le signe. Si l'on prend l'exemple d'un octet (8 bits), on a alors un bit de signe et sept bits pour coder la valeur absolue (un nombre indépendamment de son signe).

Sur 8 bits on peut représenter des nombres signés de -127 à +127. Lorsque le bit le plus à gauche est un 0, le nombre est positif. 01111111, ici le résultat est +127 comme en binaire pur. Lorsque le bit le plus à gauche est un 1, le nombre est négatif. 11111111, ici le résultat est -127.

L'inconvénient c'est que le zéro a deux représentations 00000000 et 10000000. Plus ennuyeux encore, l'addition classique ne fonctionne plus.

Pour éviter ces problèmes on utilise donc la méthode du « complément à deux ». En voici les règles :

- On définit le nombre de bits qui seront utilisés pour cette représentation (souvent 8, 16, 32 ou 64 bits). En prenant soin de prendre un nombre de bit suffisamment grand pour coder toutes les valeurs que l'on souhaite coder. 8 bits permettront de coder 256 valeurs donc de -128 à 127. 14 devra s'écrire $(0000\ 1110)_2$.
- Les nombres négatifs sont obtenus en deux étapes :
  - On inverse les bits de l'écriture binaire de sa valeur absolue (opération binaire NOT). Cela s'appelle le complément à un.
  - On ajoute 1 au résultat (les dépassements sont ignorés).

Nombre de 8 bits			
Lu en hexadécimal	Lu en binaire	Lu en décimal signé	Lu en décimal non signé
7F	0111 1111	+127	127
7E	0111 1110	+126	126
...	...	...	...
10	0001 0000	+16	16
0F	0000 1111	+15	15
0E	0000 1110	+14	14
0D	0000 1101	+13	13
0C	0000 1100	+12	12
0B	0000 1011	+11	11
0A	0000 1010	+10	10
09	0000 1001	+9	9
08	0000 1000	+8	8
...	...	...	...
02	0000 0010	+2	2
01	0000 0001	+1	1
00	0000 0000	+0	0
FF	1111 1111	-1	255
FE	1111 1110	-2	254
FD	1111 1101	-3	253
FC	1111 1100	-4	252
FB	1111 1011	-5	251
FA	1111 1010	-6	250
F9	1111 1001	-7	249
...	...	...	...
86	1000 1001	-122	134
85	1000 0101	-123	133
84	1000 0100	-124	132
83	1000 0011	-125	131
82	1000 0010	-126	130
81	1000 0001	-127	129
80	1000 0000	-128	128

2.4. Exercices

1) Faire l'addition binaire de $(100100)_2$ et $(110011)_2$.
2) Faire l'addition binaire de $(11101000)_2$ et $(110111)_2$.
3) Faire la multiplication binaire de $(1001)_2$ et $(11)_2$.
4) Faire la multiplication binaire de $(101)_2$ et $(11100)_2$.
5) Convertir $(-78)_{10}$ du décimal signé en binaire sur 8 bits.
6) Convertir $(00101100)_2$ en décimal signé.
7) Convertir $(-289)_{10}$ du décimal signé en binaire sur 16 bits.
8) Convertir $(11101110)_2$ en décimal signé.
9) Convertir $(-102)_{10}$ du décimal signé en binaire sur 8 bits.
10) Convertir $(-7042)_{10}$ du décimal signé en binaire sur 16 bits.
11) Convertir $(10111101)_2$ du binaire signé en décimal.
12) Convertir $(11100010\ 01110100)_2$ du binaire signé en décimal.

3. Représentation d'un nombre réel

En mathématiques, un nombre réel est un nombre qui peut être représenté par une partie entière et une liste finie ou infinie de décimales

En informatique, on les nomme « nombres flottants ».

3.1. Les nombres flottants

Pour représenter la partie entière d'un nombre en binaire on a vu qu'il fallait répartir cette partie entière entre les valeurs de 2^n.

Pour la partie décimale d'un nombre en binaire il s'agit de la répartir entre les valeurs 2^{-n} en partant cette fois-ci de n = -1 (avec n=0, 2^n donne toujours 1).

2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
1	0,5	0,25	0,125	0,0625	0,03125	0,015625	0,0078125	0,00390625

Par exemple $\mathbf{(4,125)_{10}}$ se représente par $\mathbf{(100,001)_2}$.

Il existe une méthode (similaire la division euclidienne par 2 pour passer un nombre entier de décimal à binaire) qui facilite la recherche de la conversion.

Exemple pour la conversion du nombre décimal 5,1875 :

La partie entière **5** se représente en binaire par **101**. Pour représenter le **",1875"** :

- On multiplie 0,1875 par 2 : **0,1875 x 2 = 0,375**. On obtient 0,375 que l'on écrira **0 + 0,375**
- On multiplie 0,375 par 2 : **0,375 x 2 = 0,75**. On obtient 0,75 que l'on écrira **0 + 0,75**
- On multiplie 0,75 par 2 : **0,75 x 2 = 1,5**. On obtient 1,5 que l'on écrira **1 + 0,5** (quand le résultat de la multiplication par 2 est supérieur à 1, on garde uniquement la partie décimale)
- On multiplie 0,5 par 2 : **0,5 x 2 = 1,0**. On obtient 1,0 que l'on écrira **1 + 0,0** (la partie décimale est à 0, on arrête le processus)

On obtient une succession de "**a + 0,b**" ("0 + 0,375", "0 + 0,75", "1 + 0,5" et "1 + 0,0"). Il suffit maintenant de "prendre" tous les "a" (dans l'ordre de leur obtention) afin d'obtenir la partie décimale de notre nombre **: 0011**

$$\mathbf{(5,1875)_{10} = (101,0011)_2}$$

3.2. Exercices

1) Convertir $(31,625)_{10}$ en binaire.
2) Convertir $(1001,011)_2$ en décimal.
3) Convertir $(11,296875)_{10}$ en binaire.
4) Convertir $(11,10101)_2$ en décimal.
5) Convertir $(0,1)_{10}$ en binaire.

4. Algèbre booléenne

L'algèbre de Boole, ou calcul booléen, est la partie des mathématiques qui s'intéresse aux opérations et aux fonctions sur les variables logiques. Elle fut inventée par le mathématicien britannique George Boole. Aujourd'hui, l'algèbre de Boole trouve de nombreuses applications en informatique et dans la conception des circuits électroniques.

Vidéos sur l'histoire de la logique (base de l'informatique) :

https://www.youtube.com/watch?v=_yEHa6dIKgg

https://www.youtube.com/watch?v=lZQgr6Lwjhg

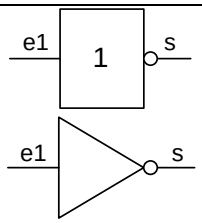
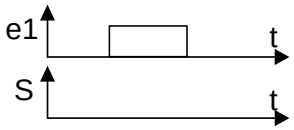
Une variable booléenne ne peut prendre que deux valeurs, notées 0 et 1. Ces variables peuvent servir à constituer une information binaire (oui/non, vrai/faux, égal/différent, marche/arrêt, allumé/éteint, …) ou à décrire l'état physique d'un composant (alimentation d'un composant, action sur un bouton, …).

La valeur 0 représente l'état physique d'un composant non alimenté, ou ne recevant pas d'action physique. La valeur 1 représente l'état physique d'un composant alimenté. La valeur booléenne est donnée par l'état logique (haut ou bas) de la grandeur physique qui la porte : de l'électricité.

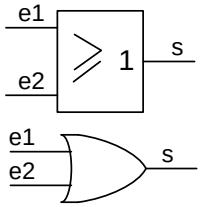
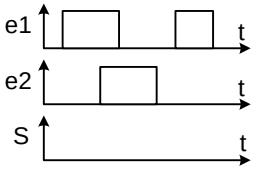
La plupart du temps, l'état logique HAUT correspond à une tension « proche » de 5V et donc à une valeur booléenne de 0 et l'état logique BAS à une tension « proche » de 0V et une valeur booléenne de 0.

Il existe alors plusieurs opérateurs (constitués de transistor) permettant d'effectuer des opérations sur ces variables booléennes.

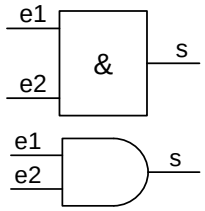
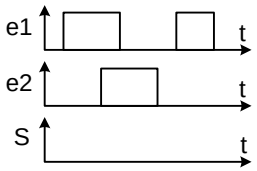
4.1. L'opérateur NOT

Table de vérité	Symbole logique	Équation logique	Schéma électrique	Chronogramme
e1 | **S** 0 | 1 1 | 0		$S = \overline{e1}$		

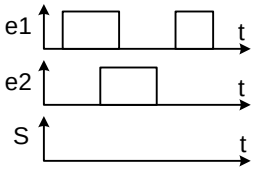
4.2. L'opérateur OR

Table de vérité	Symbole logique	Équation logique	Schéma électrique	Chronogramme
e1 | **e2** | **S** 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 1		$S = e1 + e2$		

4.3. L'opérateur AND

Table de vérité	Symbole logique	Équation logique	Schéma électrique	Chronogramme
e1 **e2** **S** 0 0 0 0 1 0 1 0 0 1 1 1		$S = e1.e2$		

4.4. L'opérateur XOR

Table de vérité	Symbole logique	Équation logique	Schéma électrique	Chronogramme
e1 **e2** **S** 0 0 0 0 1 1 1 0 1 1 1 0				

4.5. Exercices

1) Dresser la table de vérité de l'équation booléenne $S = \bar{a}.\bar{b}$
2) Dresser la table de vérité de l'équation booléenne $S = \overline{a + b}$
3) Dresser la table de vérité de l'équation booléenne $S = a + b.\bar{c}$
4) Dresser la table de vérité de l'équation booléenne $S = \bar{a}.b + a.\bar{c} + b.c$

5. Représentation d'un texte en machine

Nous avons vu que l'ordinateur est seulement capable de traiter du binaire. Pour pouvoir traiter des fichiers numériques il faut donc passer par un encodage des données en binaire.

5.1. ASCII

Le code ASCII (pour American Standard Code for Information Interchange) est une norme informatique de codage de caractères alphanumériques. ASCII comprend 128 combinaisons de 7 bits historiquement qui définissent 95 caractères imprimables, dont les chiffres arabes de 0 à 9, les lettres de l'alphabet latin non accentuées, minuscules et majuscules, des symboles de ponctuation et quelques caractères mathématiques simples.

Il convient bien pour coder les textes en anglais, mais est insuffisant pour le français, notamment en raison de l'**absence des caractères accentués**. Il est à noter aussi que les 32 premières combinaisons ne sont pas imprimables ainsi que la dernière, 127 : toutes correspondent à des commandes de terminal informatique. Ainsi, la combinaison 127 correspond à la commande pour effacer.

Le code ASCII a été étendu sur 8 bits pour incorporer certains caractères accentués (notamment les accents français). On voit ci-dessous la deuxième partie du code ASCII dit *étendu* codé sur 8 bits, la première partie étant le code ASCII historique sur 7 bits qui code en décimal de 0 à 127, en hexadécimal de 00h à 7Fh. L'extension code donc en décimal de 128 à 255 et en hexadécimal de $(80)_{16}$ à $(FF)_{16}$.

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

5.2. 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.

Cette multiplication des normes a très rapidement posé des problèmes. 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.

5.3. Unicode

Unicode est un standard informatique qui permet des échanges de textes dans de **très nombreuses langues**, à un niveau mondial. Il est développé par le **Consortium Unicode**, qui vise au codage de texte écrit en donnant à tout caractère de n'importe quel système d'écriture un nom et un identifiant numérique, et ce de manière unifiée, quelle que soit la plateforme informatique ou le logiciel utilisés.

À ce jour, Unicode référence plus de **137 000 caractères** dans une centaine d'écritures.

La principale caractéristique d'UTF-8 (variante d'Unicode) est qu'elle est **rétro compatible** avec la norme ASCII, c'est-à-dire que tout caractère ASCII se code en UTF-8 sous forme d'un unique octet, identique au code ASCII. Par exemple, « **A** » (A majuscule) a pour code ASCII 65 et se code en UTF-8 par l'octet 65.

5.4. Exercices

1) Coder en hexadécimal le mot « Bienvenue » grâce au codage ASCII.
2) En ASCII, que faut-il faire pour passer une majuscule en minuscule ?

6. En langage Python 3

Types de variables - fonctions de conversions

- ***Type integer → Nombre entier***
  int( ) → *convertit si possible un décimal ou texte en entier*
- ***Type float → Nombre décimal***
  float( ) → *convertit si possible un entier ou texte en décimal*
- ***Type string → Chaine de caractères (texte)***
  suite de signes définie en la délimitant par des guillemets
  str( ) → *convertit un nombre en chaîne*
- ***Type boolean → Logique***
  *ne prend que deux valeurs : **True** et **False***
- ***Affectation =***
  *x =... → lire « x **prend la valeur**....»* ***astuces***

```
x="12" # x est dy type sting
y=3    # y est du type integer
z=x+y   # erreur
z=int(x)+y  # donne 15
z=x+str(y)  # donne "123"
```

```
a=(1+1>2)   # a booléen qui vaut False
b=(2*3==6)  # b booléen qui vaut True
```

```
a,b=12,15 # équivaut à a=12  et b =15
x+=2      # équivaut à x=x+2
x-=1      # équivaut à x=x-1
```

Entrées, sorties console, opérations numériques

- ***Entrée → input (" message") : lit un texte saisi au clavier .***
  Renvoie donc toujours une chaîne de caractères.
  *conversion possible en nombre par **int( ) ou float( )***
- ***Sortie en console → print(, , ...) : affiche en console***
  les valeurs de tout type en les séparant par une tabulation.
- ***Opérations sur les nombres***
  / → *division décimale*
  // → *quotient de la division entière*
  % → *reste de la division entière*
  ******** → *puissance (remarque : $a**0.5 = \sqrt{a}$)*
  abs( ) → *valeur absolue*
  round(x,d) → *arrondi le nombre x à d décimales*

```
a=input("Texte?")   # a   type string
b=float(input("Nombre?")) # b type float
```

```
a=45*2
print("Coût=",a,"€") # affiche "Coût= 90 €"
```

```
d=11/4  # d vaut 2.75
q=11//4  # q vaut  2
r=11%4  # r vaut 3 car 11= 2*4+3
p=4**3  #  p vaut 64
r=16**0.5 # r vaut racine(16) soit 4
x=abs(7-10) # x vaut 3
y=round(4/3,4) # y vaut 1,3333
```

Chaînes de caractères

- **Concaténation +** → attache les textes pour n'en former qu'un
- **Caractères d'échappement**
  le signe \ permet de transformer le caractère qui suit
  \n → saut de ligne (new). **\t** → tabulation
  \" ou **\'** → guillemet qui ne ferme pas la chaine
- **longueur d'une chaine :**
  len( ) → renvoie le nombre de caractères d'une chaine, espaces compris.
- **Indexation** Chaque caractère de la chaîne est indexé (numéroté) **en commençant par 0**
  Chaine**[i]** → renvoie le caractère de rang i
  ☛ **astuces** :
  MaChaine**[-1]** → dernier caractère
  MaChaine**[-2]** → avant dernier caractère,etc…
  MaChaine**[i : j]** → caractères indéxés de i à j-1
  ☛ **Attention : on en peut pas modifier un caractère d'une chaîne par son index, seulement le lire !**
- **Code ASCII**
  chr(x**)** → renvoie le caractère de code ASCII x
  ord(char **)** → renvoie le code ASCII du caractère char
  ☛ **chr(10)** ou **chr(13)** → saut de ligne. **chr(9)** → tabulation

```
T1="Ceci est" # variable type string
T2=" un test." # variable type string
T=T1+T2   # T vaut "Ceci est un test"
```

```
T="ceci est \nun test"# Imprime: ceci est
print(T)              #          un test
T="Il m\'a dit:\"Attention\""
print(T) # imprime : Il m'a dit:"Attention"
```

```
T="Ceci est un test"
L=len(T)    # entier valant 16
print(T[0]) # écrit 'C'
x=T[2]   # x vaut  'c'
y=T[L]   # erreur de dépassement
z=T[L-1] # z vaut 't' (dernière lettre)
z=T[-1]  # z vaut 't' (dernier index)
z=T[-2]  # z vaut 't' (avant dernier index)
z=T[1:6] # z vaut 'eci e' (indexe 1 à 5)
```

```
T="Pithon"
T[1]="y" # erreur: affectation non autorisée
```

```
x=chr(65) # x vaut 'A'
y=ord('B') #  y vaut 66
```