Divers codages d'information en informatique.

Le bit, l'octet (le byte), le mot, l'hexadécimal.

Dans un ordinateur, l'information est stockée sous forme de tensions électriques. Soit la tension est inférieure à un seuil, soit elle est supérieure à un seuil. Dans une mémoire flash, elle est stockée sous la forme de présence ou d'absence d'électrons dans un condensateur. Sur un disque dur ou une bande magnétique, l'information est stockée dans des minuscules régions sous forme de magnétisation, soit dans un sens, soit dans l'autre sens. Sur un CD-ROM ou un DVD, elle est stockée sous forme de creux et de bosses, qui réfléchissent différemment la lumière. Autrefois, on stockait l'information sur des bandes perforées, soit il y avait un trou, soit il n'y en avait pas.

Dans tous les cas, l'information élémentaire n'a que deux possibilités, que l'on note généralement **0** et **1**.
Cette information élémentaire est appelée un **bit**, abréviation de **BI**nary uni**T**.
On peut donc représenter l'information stockée dans un ordinateur, un disque dur, etc. par une suite de bits, donc une suite de 0 et de 1.

Pour plus de commodité, on a regroupé ces bits en **octets** ou **bytes**. Un **octet** est une suite de 8 bits. En écrivant ces 8 bits sous forme de 8 chiffres valant soit 0 soit 1, cela correspond à écrire un nombre en **notation binaire**. On dit pour cela que toute l'information est stockée sous forme de nombres dans un ordinateur, ce qui n'est qu'une manière pratique de se représenter la mémoire d'un ordinateur.

Un **bit** n'a que deux valeurs possibles, qui sont zéro ou un.
Un **octet** a $2^8 = 256$ valeurs possibles, qui vont de 0000 0000 à 1111 1111 (0 à 255).

On regroupe souvent les octets par pairs pour former des **mots de 16 bits**.
Un **mot de 16 bits** a $2^{16} = 65'536$ valeurs possibles.
On regroupe souvent les octets par quadruplets pour former des **mots de 32 bits**.
Un **mot de 32 bits** a $2^{32} = 4'294'967'296$ valeurs possibles, soit environ 4 Giga valeurs.
La limitation de mémoire de 4 Giga octets des systèmes d'exploitations de 32 bits vient de là.
On regroupe encore les octets par série de huit, pour former des **mots de 64 bits**.
Un **mot de 64 bits** a 2^{64} valeurs possibles, soit environ $16 \cdot 10^{18}$ valeurs.
Les systèmes d'exploitations 64 bits ne sont pas limités par la taille des mots de 64 bits.

Pour noter un octet ou un mot de 16 ou 32 bits, la notation binaire est désagréable. Pour cela on utilise la notation **hexadécimale**, qui regroupe des bits par série de 4, pour former des nombres allant de 0 à 15.

```
0000 = 0    0001 = 1    0010 = 2    0011 = 3    0100 = 4    0101 = 5    0110 = 6    0111 = 7
1000 = 8    1001 = 9    1010 = A    1011 = B    1100 = C    1101 = D    1110 = E    1111 = F
```

En notation hexadécimale, six "chiffres" sont ajoutés, que l'on note A, B, C, D, E et F.
Ainsi, un octet est noté à l'aide de deux nombres hexadécimaux.
0000 0000 = #00 Le # est une manière d'indiquer la numérotation hexadécimale.
1111 1111 = #FF

Exercice 1
Complétez les notations binaires et hexadécimales :

```
01011101  =                    10000111  =

          = #46                          = #7A

          = #CC                          = #B0
```

Exercice 2
Complétez...

Un mot de 16 bits est représenté par ____ chiffres hexadécimaux.

Un mot de 32 bits est représenté par ____ chiffres hexadécimaux.

#FA18 représente un mot de ____ bits.

Le plus grand mot de 32 bits est noté en hexadécimal par _____

Codage des couleurs

Puisque l'œil humain est composé de 3 sortes de capteurs de couleurs, sensibles soit aux couleurs voisines du rouge, soit du vert, soit du bleu, on peut coder presque toutes les couleurs en indiquant la proportion de rouge, de vert et de bleue.
Une image sur un écran d'ordinateur ou de télévision est constitué de **pixels** (PIXture ELements).
Chaque pixel a une couleur, déterminée par la quantité de rouge, de vert et de bleu. C'est le système **RVB** ou **RGB** (Red Green Blue en anglais).
Chacune des trois quantités est déterminée par un octet, donc la valeur comprise entre 0 et 255.
Donc un pixel est représenté par un nombre hexadécimal de 6 chiffres, les deux premiers indiquant la quantité de rouge, les deux suivants indiques la quantité de vert, les deux derniers celle de bleue.

Exercice 3
Faites la correspondance entre le nombre hexadécimal et la couleur.

#FF0000 ↔ _____ #00FF00 ↔ _____

#0000FF ↔ _____ #888888 ↔ _____

_____ ↔ noir _____ ↔ blanc

_____ ↔ gris foncé _____ ↔ gris clair

_____ ↔ jaune _____ ↔ magenta

_____ ↔ cyan _____ ↔ brun

Codage de caractères, ASCII, iso-8859-1, iso-885-15, Windows-1252, utf-8

Chaque caractère est aussi codé par une séquence de bits. A l'origine, c'était facile, car les américains n'avaient besoin de coder que les 26 caractères alphabétiques minuscules, les 26 majuscules, les 10 chiffres et quelques caractères de ponctuation tels que : , . ; : ' ? ! + - = " ( ) * % & / [ ] { } etc. 96 (=128 – 32) symboles suffisaient amplement.
Donc pour ces caractères, un octet est largement suffisant.
Les 32 premiers nombres, codent des caractères de contrôle, tel que le tabulateur, le retour à la ligne, le saut de ligne, le saut de page, la fin de fichier.
Les nombres allant de 32 à 127 codent les caractères cités précédemment.
Le 8ème bit servait de bit de contrôle, pour tester si une erreur s'était introduite, par exemple lors d'une transmission. Ce bit de contrôle n'est plus utilisé de nos jours.
Ce codage sur 7 bits est le **code ASCII**. Il est **standard** et utilisé par tous les systèmes d'exploitation.

Avec l'apparition du DOS et l'utilisation d'ordinateurs en Europe, divers codages sont apparus pour coder les caractères accentués. Cela c'est fait de manière complètement anarchique, et de nombreux systèmes de codages ont étendus le code ASCII. Microsoft et Apple ont chacun développé leur système de codage.
En 1986, la norme **iso-8859-1** a définie le codage des caractères accentués, sur un octet.
Il définit le codage de 191 caractères.
Il manquait certains caractères, tel que le : œ.
La norme **iso-8859-15** a comblé en 1998 certains manques, en étendant le codage iso-8859-1.
Ce codage se nomme également **Windows-1252**.
L'avantage de ces systèmes de codage est qu'un caractère est codé sur un octet.

Pour les Américains, du nord et du sud et les Européens, cela suffit. Mais pas pour les arabes, les Japonais, le chinois et une majorité du monde ! Pour cela d'autres systèmes de codages ont été définis.
Au début des années 1990, le codage **UTF-8** est apparu. Il est utilisé actuellement dans plus de trois quarts des pages Web. Il a l'*avantage* de pouvoir coder plus d'un million de caractères, dont les caractères arables, japonais, chinois et bien d'autres. Il a le *désavantage* d'utiliser de 1 à 4 octets pour coder un caractère. Il est *compatible avec le codage ASCII*. Donc les caractères de l'ASCII sont codés sur un octet en utf-8, de la même manière qu'en ASCII.
Par contre, les caractères accentués sont codés sur deux octets.
C'est ce système de codage qui est utilisé sur UNIX, Linux et MAC OSX.
D'autres systèmes de codage existent, tels que l'utf-16 et l'utf-32. **Unicode** est une définition encore plus générale.

Exercice 4
Complétez le tableau suivant : (# signifie codage en hexadécimal) c.f. http://www.utf8-chartable.de/

Caractère	code ASCII	code iso-8859-15	code utf-8
A			
	#61 = 97		
			#20 = 32
é			
ñ			
			#c3a6
			#e0a9a9

Codage des nombres entiers

Les nombres peuvent se classer dans trois catégories. Les nombres entiers positifs, les nombres entiers relatifs et les nombres à virgules. L'informatique étant limitée, aucune distinction n'est faite entre les nombres irrationnels et les nombres rationnels. Parmi tous les nombres, très peu peuvent être codés de manière exacte.

Exercice 5
Si on se limite aux **entiers positifs** :
un **octet** peut coder les nombres de 0 à 255.
un **mot de 16 bits** peut coder les nombres de 0 à _____
un **mot de 32 bits** peut coder les nombres de 0 à _____
un **mot de 64 bits** peut coder les nombres de 0 à _____

Le codage des **entiers relatifs**, suit une logique mathématique.
Remarquons que : 255 + 1 = 256 = 1 0000 0000 en binaire.
Si on élimine le 9ème bit, cela donne 255 + 1 = 0, donc 255 représente le nombre entier négatif –1.
Toujours en éliminant le 9ème bit, 254 + 2 = 0, donc 254 représente le nombre entier négatif –2.

Sur un octet, on code les nombres entiers relatifs comme suit :
0 1 2 ... 127 sont les entiers positifs
255 254 ... 129 sont les entiers négatifs, de –1, –2, ... à –127.
128 est une autre manière de coder 0, il correspond à " –0 ".

Sur un **mot de 16 bits**, rappelons que 2^{16} = 65'536.
65'535 + 1 = 0, si on élimine le 17ème bit. Donc 65'535 = –1.
32'769 + 32'767 = 0, donc 32'769 = –32'767.

Sur un **mot de 32 bits**, rappelons que 2^{32} = 4'294'967'296. La moitié vaut : 2'147'483'648.

Le codage des nombres négatifs dépend du nombre d'octets choisi pour coder les nombres entiers.

Exercice 6
Sur un *mot de 32 bits*, à quel entier positif correspond le nombre –1 ?

En se limitant aux *octets*, que donne l'opération suivante, en nombre positif et en nombre négatif.
En nombre négatif, on parlera d'**overflow**, qui signifie que l'on dépasse les capacités de ce codage.
69 + 61

Sur un *mot de 32 bits*, dans le codage des entiers relatifs, quel est le plus grand nombre positif ?

Suggérez la taille d'un mot à considérer pour coder des entiers relatifs plus grand que mille milliards.
Dans ce système, quel sera le plus grand entier ? À quel nombre entier correspondra –1 ?

Dans un système de *mots de 16 bits*, donnez un exemple d'addition provocant un overflow.

Dans un système de *mots de 8 bits*, donnez un sens à l'égalité suivante.
2 + 250 = –4

Codage des nombres à virgule

Le système de codage des nombres à virgules dans un ordinateur est assez complexe. On ne considérera que deux type de codage, en simple précision, sur 32 bits et en double précision, sur 64 bits. Ce dernier type de codage est le plus couramment utilisé de nos jours. Ces codages imitent l'écriture scientifique des nombres.

Codage **simple précision** = à **virgule flottante en 32 bites** : nombre = $s \cdot 2^{e-127} \cdot 1,$mantisse
s = 1 ou –1 suivant la valeur du premier bit.
e = le nombre entier positif représenté par les 8 bits allant du 2ème au 9ème.
La mantisse donne les 23 chiffres binaires après la virgule.

Codage **double précision** = à **virgule flottante en 64 bites** : nombre = $s \cdot 2^{e-1023} \cdot 1,$mantisse
s = 1 ou –1 suivant la valeur du premier bit.
e = le nombre entier positif représenté par les 11 bits allant du 2ème au 12ème.
La mantisse donne les 52 chiffres binaires après la virgule.

Il y a une exception si e = 0, alors le nombre = $s \cdot 2^{-127} \cdot 0,$mantisse en simple précision.
Il y a aussi une exception si e = maximum, le nombre représente + ou – l'infini.

Exemples en simple précision :

```
 1      = 0 01111111 00000000000000000000000
-2      = 1 10000000 00000000000000000000000
13      = 0 10000010 10100000000000000000000
```

Le signe est positif.§
le facteur multiplicateur vaut $2^{130-127} = 2^3 = 8$
1,mantisse vaut $1 + 1 \cdot 1/2 + 0 \cdot 1/4 + 1 \cdot 1/8 + 0 = 13/8$
Donc le nombre vaut bien $8 \cdot 13/8 = 13$.

```
 0,625  = 0 01111110 01000000000000000000000
```

le facteur multiplicateur vaut $2^{126-127} = 2^{-1} = 1/2 = 0,5$
1,mantisse vaut $1 + 0 \cdot 1/2 + 1 \cdot 1/4 + 0 \cdot 1/8 + 0 = 1 + 1/4 = 1,25$
Donc le nombre vaut bien $0,5 \cdot 1,25 = 0,625$

```
 0,1    ≈ 0 01111011 10011001100110011001100
```

le facteur multiplicateur vaut $2^{123-127} = 2^{-4} = 1/16 = 0,0625$
1,mantisse vaut $1 + 2^{-1} + 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-20} + 2^{-21}$
Le produit des deux nombres ci-dessus vaut presque 0,1.
Il n'est pas possible d'écrire 0,1 de manière exacte dans ce système de codage des nombres !

Exemples en double précision :

```
  1      = 0 01111111111 0000000000000000000000000000000000000000000000000000
-13      = 1 10000000010 1010000000000000000000000000000000000000000000000000
 -0,1    ≈ 1 01111111011 1001100110011001100110011001100110011001100110011001
  0,75   = 0 01111111110 1000000000000000000000000000000000000000000000000000
```

le facteur multiplicateur vaut $2^{126-127} = 2^{-1} = 1/2 = 0,5$
1,mantisse vaut $1 + 1/2 = 1,5$
Donc le nombre vaut bien $0,5 \cdot 1,5 = 0,75$

Pour qu'un nombre puisse s'écrire de manière exacte, il faut qu'il puisse s'écrire sous la forme d'un nombre entier divisé par 2 à la puissance un nombre entier.
Donc la plupart des nombres à virgules sont approximés en informatique.

Utilisez des feuilles supplémentaires pour résoudre les exercices qui suivent.

Exercice 7 :
En simple précision, quels sont les nombres représentés par :

```
A = 0 10000011 00010000000000000000000
B = 0 01111101 01000000000000000000000
C = 1 10000011 00000100000000000000000
D = 0 01111111 10011001100110011001100
E = 1 01111101 00110011001100110011001
F = 0 10000000 10010010000111111011010
G = 0 10000000 10010001111010111000010
H = 0 10000000 01011011111100001010100
I = 0 01111111 01101010000010011110011
J = 0 01111111 10011110001101110111100
```

Exercice 8 :
En double précision, quels sont les nombres représentés par :

```
A = 0 10000000000 10010010000111111011010100010001000010110100011000
B = 0 10000000000 01011011111100001010100010110001010001010111011010001
C = 0 01111111111 01101010000010011110011001100111111100111011111001101
D = 0 01111111111 10011110001101110111100110111001011111110100101010000
```

Exercice 9 :
En vous basant sur l'exercice 7, pouvez-vous écrire la représentation binaire en double précision, du nombre 1,6 ? Et du nombre –0,3 ?

Exercice 10 :
Représentez en binaire, simple précision, les nombres suivants :
(Si vous êtes particulièrement curieux, vous pouvez les représenter aussi en double précision)

```
 55
−28
Votre âge
 0,6
 -pi/4
 0,7
Le plus grand nombre possible
Le plus petit nombre positif
```

Exercice 11 :
Écrivez un programme en javascript, ou dans un autre langage, qui fait toutes les conversions de nombres...
C'est un projet en soi !