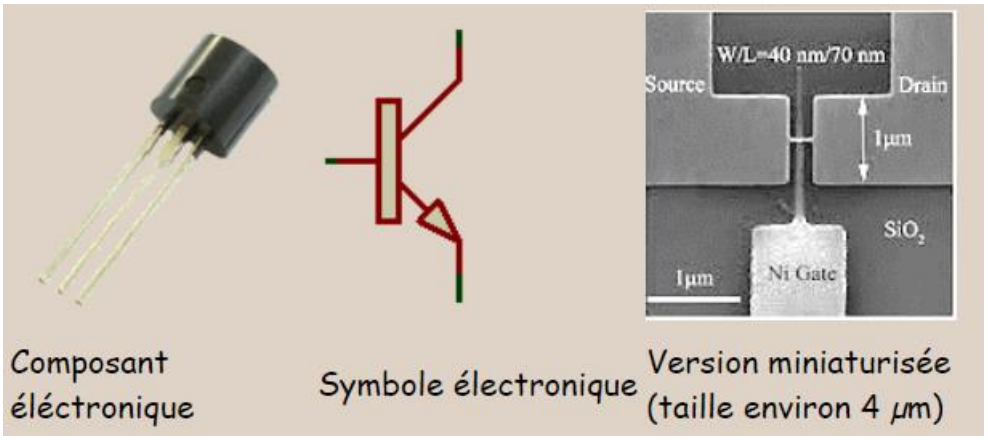
Vus de l'extérieur, les ordinateurs et les programmes que nous utilisons tous les jours permettent de **mémoriser**, de **transmettre** et de **transformer** des nombres, des textes, des images, des sons, etc.

1. Un peu d'électronique :

L'ordinateur actuel existe suite à l'invention d'un composant essentiel : **le transistor**.



Dans son fonctionnement le plus simple le transistor peut prendre deux états :

- Un **état bloqué** : dans cet état, le transistor ne permet pas le passage du courant.
- Un **état passant** (on dit aussi saturé) : le transistor est en état de conduction. Le courant passe.

Dans la pratique on représente ces **2 états** par **les symboles 0 ou 1**

L'informatique utilise des courants électriques, des aimantations, des rayons de lumière...
Chacun de ces phénomènes met en jeu ces deux états possibles :

- Tension nulle ou tension non nulle (5V par ex),
- Aimantation dans un sens ou dans l'autre sens,
- Lumière ou pas de lumière.

De ce fait, un ordinateur ne **manipule que des 0 et des 1**.
Les nombres, les textes, les images, les sons, etc. mémorisés, transmis, transformés par les ordinateurs doivent être **représentés comme des suites de 0 et de 1**.

Une telle valeur, 0 ou 1, s'appelle un **booléen**, un **chiffre binaire** ou encore un **bit** (*binary digit)*.
Une suite de bits, par exemple 0000110100, est appelé **un mot**.

2. Les systèmes de numérations : représentation des entiers naturels :

a) Numération décimale :

Pour que vous compreniez le fonctionnement du binaire, et des systèmes de comptage en général (plus communément appelés **bases**), on va commencer par faire une petite réintroduction à la base 10 que vous connaissez tous.

En effet, tout le monde sait compter en base 10 (décimal).
Mais comment ça marche ? Comment est construit notre système ?
Pour répondre à cette question à l'apparence simple, oubliez tout et reprenons depuis le début : comment avez-vous appris à compter à l'école ?

La numération décimale utilise 10 chiffres : **0, 1, 2, 3, 4, 5, 6, 7, 8 et 9**.
Avec ces derniers, on peut **compter** jusqu'à **9.**
Et si l'on veut aller au-delà de 9, il faut **changer de rang**.

Cela signifie que si le rang des unités est plein, il faut passer à celui des dizaines, puis des centaines, milliers et ainsi de suite.

Par exemple :

à 19, le rang des **unités** est "**saturé**" (plein), car il contient le **chiffre 9**, et il n'y a pas (dans la **base 10**) de **valeur plus élevée**.

Pour passer au **nombre suivant**, il faut donc incrémenter le rang périphérique puis réinitialiser l'état de celui des unités.
Ce qui signifie : j'ai **19**, je ne peux pas mettre **plus de 9 à droite**, donc j'ajoute **1 à celui de gauche** et je remets à **zéro celui de droite**.
On obtient alors **20.**

Le nombre entier va être **composé de rangs** (**unités**, **dizaines**, **centaines**, etc).

Chaque rang vaut le **rang précédent** multiplié par **l'indice de la base** :

- Une **centaine** vaut 10 dizaines, et
- une **dizaine** vaut **10** unités.

Par exemple :

Dans l'image ci-contre, on peut voir le nombre $\mathbf{185_{10}}$
(ici, le $_{10}$ signifie qu'il s'agit d'un nombre, en **base 10**).
Dans ce nombre, on peut voir **trois rangs** : centaines, dizaines et unités.
Pour n'importe quelle base, la valeur d'un rang est égale à $\mathbf{b^n}$, où **b est l'indice de la base** (ici, 10) et $^{\mathbf{n}}$ la **position du rang**.
Ici, les **unités ont la position 0**, les **dizaines la position 1** et les **centaines la position 2**.
Nous pouvons donc écrire :

- $\mathbf{185 = 1\times10^2 + 8\times10^1 + 5\times10^0}$
- L'écriture du nombre $\mathbf{275_{10}}$ se traduit par
  $\mathbf{275 = 2\times10^2 + 7\times10^1 + 5\times10^0}$

1) Ecrire une égalité semblable pour les nombres
   a) 1234
   b) 875

b) Numération binaire :

La ***notation binaire*** est la notation à position en base 2.
C'est la base utilisée en informatique.
Il suffit de **deux chiffres** pour traduire ces états : **0** et **1**.

En binaire, c'est pareil à la différence qu'on utilise le terme **bit** (à la place de rang), qui est la contraction de "**binary digit**", littéralement "chiffre binaire".

Là où tout se complique, c'est que chaque rang en binaire ne peut avoir que **deux valeurs** (binaire = base 2) **différentes** : **0 ou 1**.
Pour la **base 10**, chaque **rang** représente une **puissance de 10**, de même, pour la **base 2**, chaque rang occupe une **puissance de 2**.

Voici comment compter en binaire jusqu'à 10 :

Nombre en décimal	Nombre en binaire	Le pourquoi du comment
0	0	Pour l'instant, ça va.
1	1	Là encore, c'est simple.
2	10	Le premier rang ayant été rempli, on passe au suivant !
3	11	On re-remplit le rang 1.
4	100	Le rang 2 est plein, le rang 1 aussi, qu'à cela ne tienne, on passe au suivant.
5	101	On continue en suivant la même méthode.
6	110	
7	111	
8	1000	On commence le rang 4.
9	1001	On continue comme tout à l'heure.
10	1010	
...		

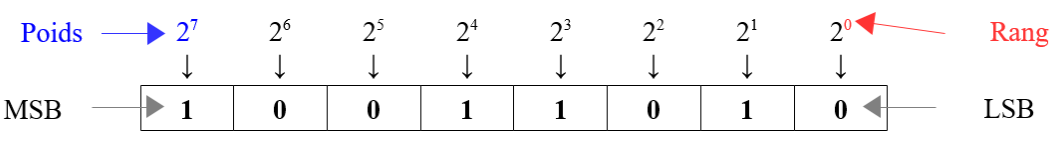
- ***Conversion binaire décimale*** :

Considérons le nombre binaire 10011010.
On le notera ici $1001\ 1010_2$ pour plus de lisibilité.
Le nombre 10011010_2 occupe 8 bits.

L'écriture du nombre $\mathbf{10011010_2}$ se traduit par :

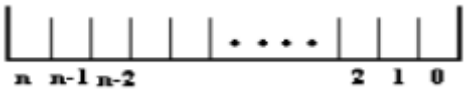
$$\mathbf{10011010} = 1\times2^7+0\times2^6+0\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0$$
$$= 128 + 0 + 0 + 16 + 8 + 0 + 2 + 0 = 154$$

On dit que **10011010 est la représentation en base 2 de 154**.



Voici le schéma d'une mémoire à n+1 bits

Les cases du schéma représentent les bits, le chiffre marqué en-dessous d'une case indique la puissance de 2 à laquelle est associé ce bit (on dit aussi rang du bit).



- Le bit de **rang 0** est appelé le bit de **poids faible.**
- Le bit de **rang n** est appelé le bit de **poids fort**.

2) a) Combien de valeurs peut-on coder avec 1 bit ?
b) Combien de valeurs peut-on coder avec 2 bits ?
c) Combien de valeurs peut-on coder avec 3 bits ?
d) Combien de valeurs peut-on coder avec n bits ?

3) Traduire en écriture décimale les nombres binaires $(1101)_2$; $(1111\ 1111)_2$ et $(1001\ 0110)_2$

4) C'est en $(111\ 1001\ 0000)_2$ qu'a été démontré le théorème fondamental de l'informatique.
Exprimer ce nombre en base 10.

- **Conversion du système décimal vers le système binaire**

Il existe plusieurs méthodes pour obtenir l'expression binaire d'un nombre exprimé en décimal.

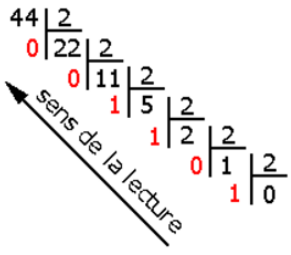
Une des plus simple et plus rapide est la méthode euclidienne.
(En plus, facile à utiliser en programmation)

- On prend le nombre en base 10 (forme normale).
- On le divise par 2 et on note le reste de la division (soit 1 soit 0)
- On refait la même chose avec le quotient précédent, et on met de nouveau le reste de côté.
- On réitère la division, jusqu'à ce que le quotient soit 0.
- Le nombre en binaire apparaît alors : il suffit de prendre tous **les restes** de **bas en haut**.

Exemple :

On a le nombre décimal 44

44 ÷ 2 = 22 + **0**
22 ÷ 2 = 11 + **0**
11 ÷ 2 = 5 + **1**
5 ÷ 2 = 2 + **1**
2 ÷ 2 = 1 + **0**
1 ÷ 2 = 0 + **1**



On trouve alors **$44 = (101100)_2$**

5) Trouver la représentation binaire des nombres décimaux suivants : 187 ; 218 et 1000

6) Un nombre **binaire de huit chiffres** est un **octet**.
Quel est le plus grand nombre que l'on peut représenter avec un octet ?

7) Justifier qu'avec un **mot** de **n** bits, on peut représenter les nombres de 0 à $2^n - 1$.

c) Numération hexadécimale :

E informatique, tout est basé sur le binaire, et étant une base d'indice 2, c'est plus aisé **d'encoder les informations sur un nombre multiple de 2**.

On utilise donc souvent la base 16, appelé **système hexadécimal**
(hexa = 6, déci = 10, 16 = 6 + 10) car 16 est un multiple de 2, et qu'il permet de **représenter 8 bits avec seulement 2 chiffres**.

Ça paraît simple, mais il y a un autre problème : en base 10, on utilise 10 chiffres.
En base 2 (binaire) on utilise seulement 2 chiffres : 0 et 1.
Mais du coup, la base 16 est un regroupement de 4 bits, elle comporte donc 16 caractère :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F.

On peut établir une liste de correspondances entre la base 10 et la base 16 (voire même la base 2) :

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

Notation :

En programmation, pour indiquer la base 16, on place le symbole **$, #** ou les symboles **0×** devant le nombre : **$(3D2)_{16}$ = $3D2 = 0×3D2 = #3D2**.

Exemple d'utilisation :

Le système hexadécimal est un des **modes de codage informatique** des couleurs des **écrans d'ordinateurs**.

000000	0000FF	00FF00	FF0000	FFFF00	FFFFFF
Noir	*Bleu*	*Vert*	*Rouge*	*Jaune*	*Blanc*

- **Conversion du système hexadécimal vers le système décimal**

  L'écriture du nombre **$(23A)_{16}$** se traduit par :
  $23A = 2×16^2 + 3×16^1 + 10×16^0 = 512+48+10 = 570$

  On dit que **23A** est la représentation **en base 16 de 570**.

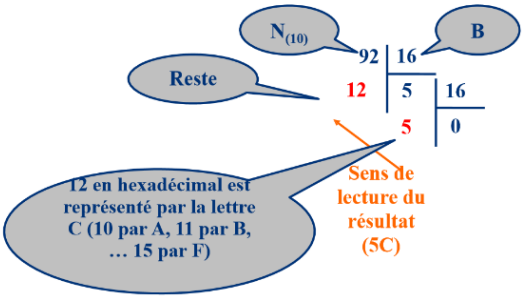
8) Traduire en nombre décimal les nombres hexadécimaux suivant : $(B8C)_{16}$; $(281EF)_{16}$ et $(92)_{16}$

9) A quel nombre décimal correspond le nombre FF ?

- **Conversion du système décimal vers le système hexadécimal**

  Pour trouver l'équivalent hexadécimal d'un nombre décimal, on peut utiliser la méthode des divisions successives par 16. Même méthode qu'avec la base 2.

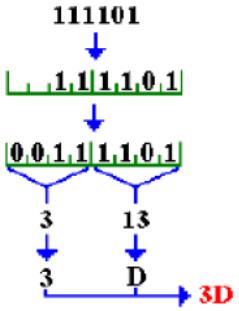
  On trouve alors : **92 = 5C**



10) Traduire en nombre hexadécimaux les nombres décimaux suivant : 965 et 6725

- **Passerelle entre la base 2 et la base 16**

Pour passer de la base 2 à la base 16 :

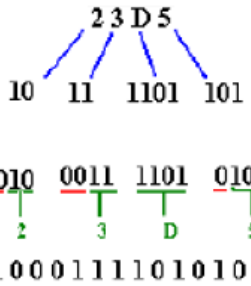


- On décompose ce nombre par **tranches de 4 bits** à partir du bit de poids faible ($2^4 = 16$)
- On complète la dernière tranche (celle des bits de poids forts) par des 0 s'il y a lieu.
- On convertit chaque tranche en son symbole de la **base 16.**
- On réécrit à sa place le nouveau symbole par changements successifs de chaque groupe de 4 bits.

Donc $\mathbf{(111101)_2 = (3D)_{16}}$

Pour passer de la base 16 à la base 2 :

Cette conversion est l'opération inverse de la précédente.



- On convertit chaque symbole hexadécimal du nombre en son écriture binaire (nécessitant au plus 4 bits).
- Pour chaque tranche de 4 bits, on complète les bits de poids fort par des 0 s'il y a lieu.
- On regroupe toutes les tranches de 4 bits à partir du bit de poids faible, sous forme d'un seul nombre binaire.

Donc $\mathbf{(23D5)_{16} = (10001111010101)_2}$

11) Convertir $(1100011010)_2$ et $(0011\ 1111\ 0101)_2$ en base 16

12) Convertir $(45A)_{16}$ et $(7CF21)_{16}$ en base 2.

3. Utilisation de Python

a) Fonctions Python :

En *Python*, quelques fonctions permettent le passage d'une représentation à une autre.
Tout d'abord, pour passer une représentation décimale à une représentation binaire ou hexadécimale, *Python* dispose des deux fonctions :

▹ bin

qui reçoit un **entier naturel** et renvoie une **chaîne de caractères** associée à la **représentation binaire** de cet entier ;
cette chaîne **débute systématiquement par le préfixe 0b**

```
>>> bin(123)
    '0b1111011'
```

▷ hex

qui procède de la même façon et renvoie une **chaîne de caractères** associée à la **représentation hexadécimale** de l'entier, avec le **préfixe 0x**.

```
>>> hex(123)
    '0x7b'
```

Les **opérations inverses** peuvent se faire à l'aide de la **fonction int** qui reçoit **deux arguments** :

- une **chaîne de caractères** représentant un **entier dans une base donnée** et
- un **entier** représentant **cette base**.

```
>>> int('0b1111011', 2)
    123

>>> int('0x7b', 16)
    123
```

b) Programmer des fonctions de conversion en python :

On se propose de construire plusieurs fonctions qui réalisent ces opérations de conversions.

- Partant d'un **type entier**, en représentation décimale, ces fonctions doivent renvoyer une chaîne de caractères associée à **la représentation de l'entier** dans les bases **2 et 16**.

- Partant d'une **chaîne de caractères** représentant un **entier binaire** ou **hexadécimal**, ces fonctions doivent renvoyer l'entier associé en **représentation décimale**.